TensorFlow

Time series fundamentals

+

Milestone Project 3

BitPredict 💰📈

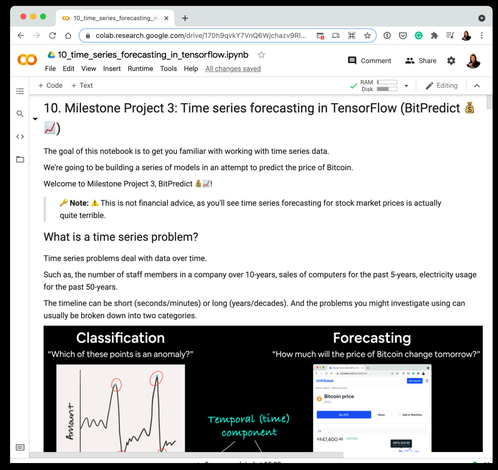
⚠️ This is not financial advice

(Spoiler: by the end of this module you'll see where even deep learning models fail to predict in open systems)

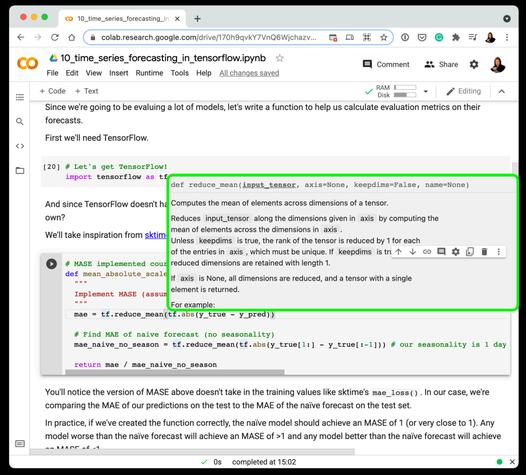
Where can you get help?

- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask (don't forget the Discord chat!)

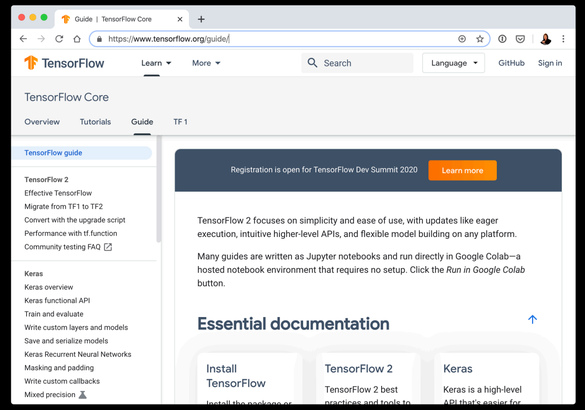
(yes, including the "dumb" questions)



"If in doubt, run the code"



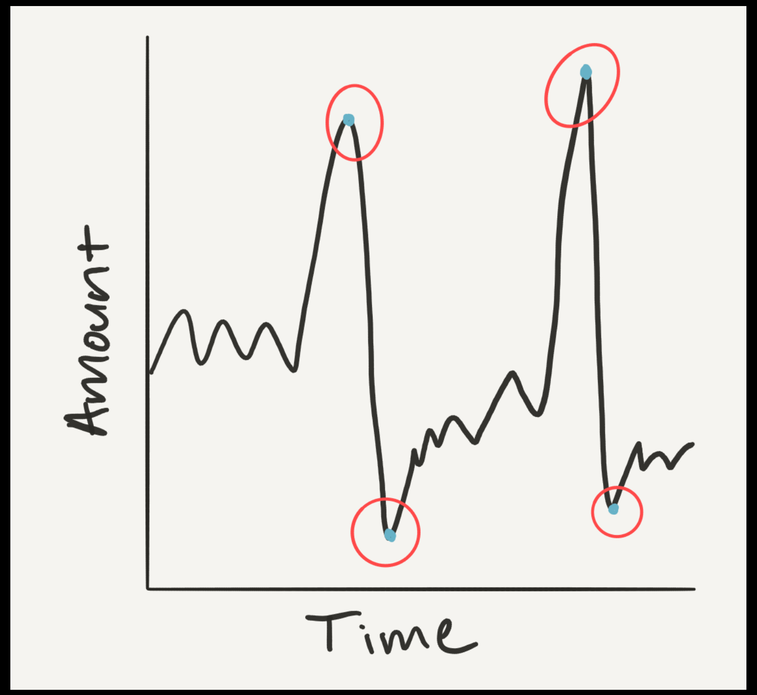
stack overflow



Example Time Series Problems

Classification

“Which of these points is an anomaly?”



“What electronic device is this?”

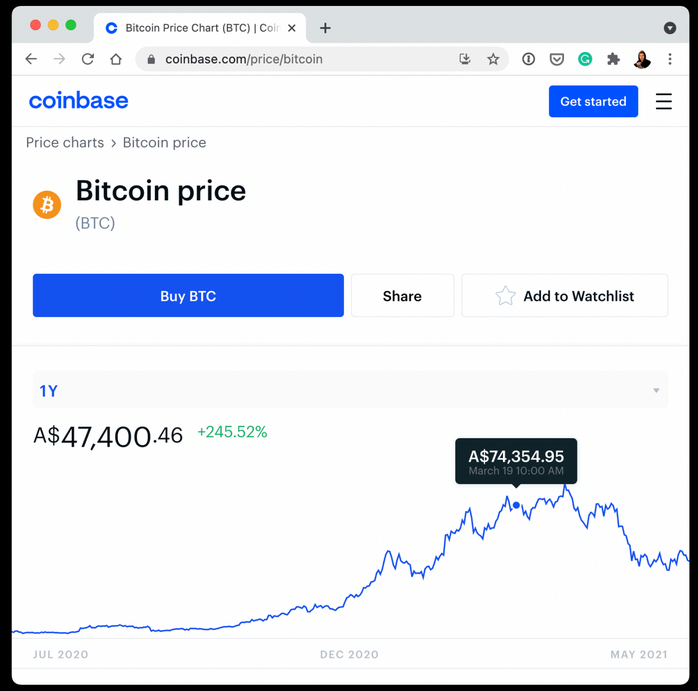
“Are these heartbeats regular?”

Output: discrete

Temporal (time) component

Forecasting

“How much will the price of Bitcoin change tomorrow?”



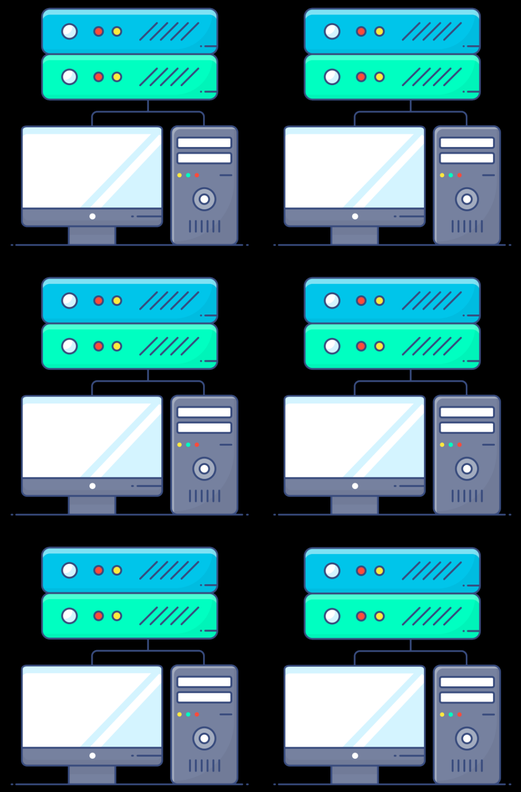
“How many computers will we sell next year?”

“How many staff do we need for next week?”

Output: continuous

Example Forecasting Problems at Uber

Market demand

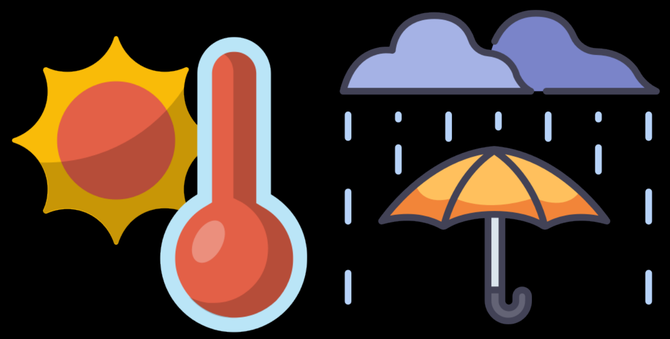


Compute requirements

Marketing campaigns

Source: Forecasting at Uber blog post

Example Forecasting Problems in daily life



TECH \ TRANSPORTATION \ CARS

Lyft's president says 'majority' of rides will be in self-driving cars by 2021

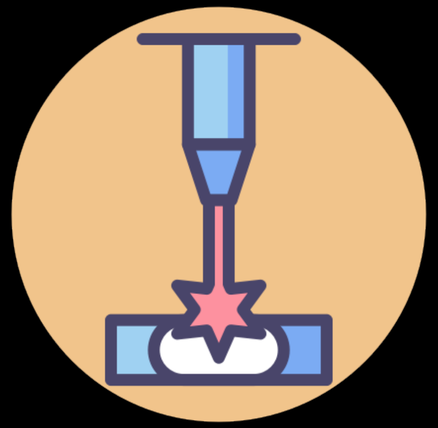
Also say bye-bye to personal car ownership

By Andrew J. Hawkins | @andyjayhawk | Sep 18, 2016, 9:15am EDT

"I think there is a world market for maybe five computers."
— Thomas Watson, president of IBM, 1943

(there are many more of these)

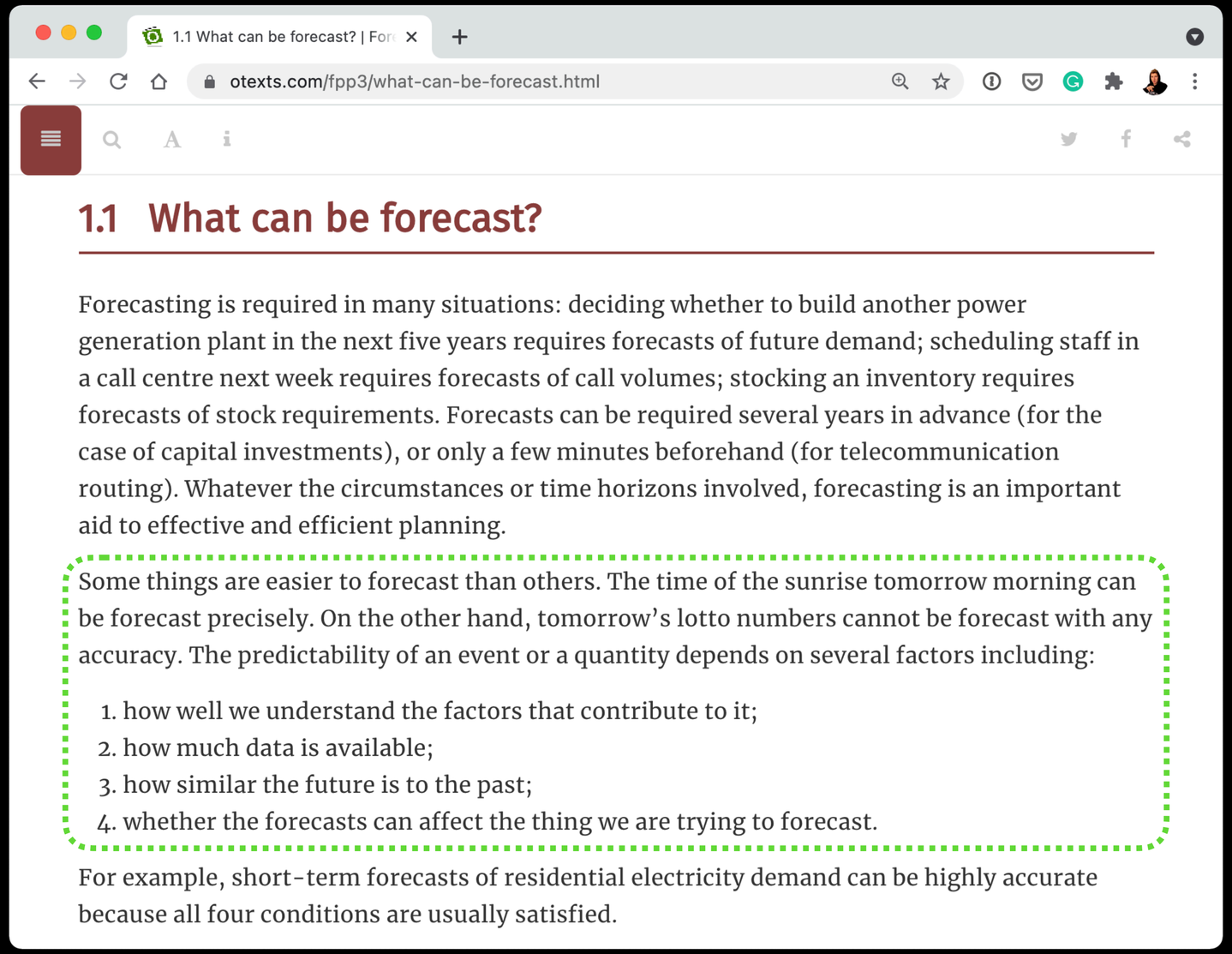
What can we predict?



What do these things have in common?

(Massive impact but unplanned, unpredicted)

What can we predict?



Source: Forecasting: Principles and Practice 3rd edition chapter 1.1

What we're going to cover

(broadly)

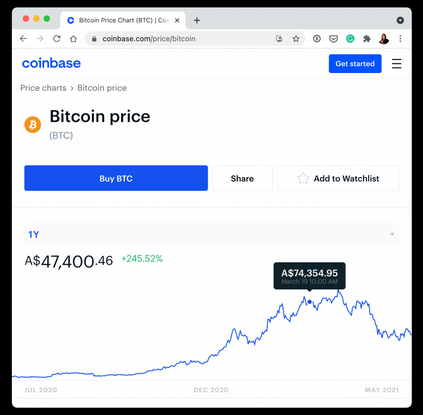
- Downloading and formatting time series data (the historical price of Bitcoin)
- Writing a preprocessing function to prepare our time series data
- Setting up multiple time series modelling experiments
- Building a multivariate model to take in multivariate time series data
- Replicating the N-BEATS algorithm using TensorFlow
- Making forecasts with prediction intervals
- Demonstrating why time series forecasting can be BS with the turkey problem 🦃

(we'll be cooking up lots of code!)

How:

Time series inputs and outputs

(Forecasting wise)



BitPredict

A$47,744

Actual output

"What will the price of Bitcoin be tomorrow?"

```
[[37888, 39124, 40563…],
[39124, 40563, 42458…],
[40563, 42458, 44899…],
…,
```

Numerical encoding

(Time series data is often already numerical)

Inputs

Machine Learning Algorithm

(often already exists, if not, you can build one)

Outputs

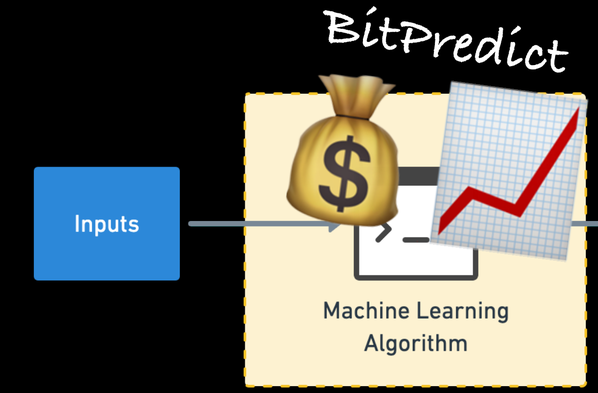
```
[[40669],
[43443],
[46986],
…,
```

Predicted output

(comes from looking at lots of these)

BitPredict input and output shapes

```
# Window for one week
[[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795],
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033],
[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049],
...]
```




```
# Predict the next day
[[123.033],
[124.049],
[125.961],
...]
```

`[batch_size, window_size]`

```
Shape = [None, 7]
        or
Shape = [32, 7]
```

(32 is a very common batch size)

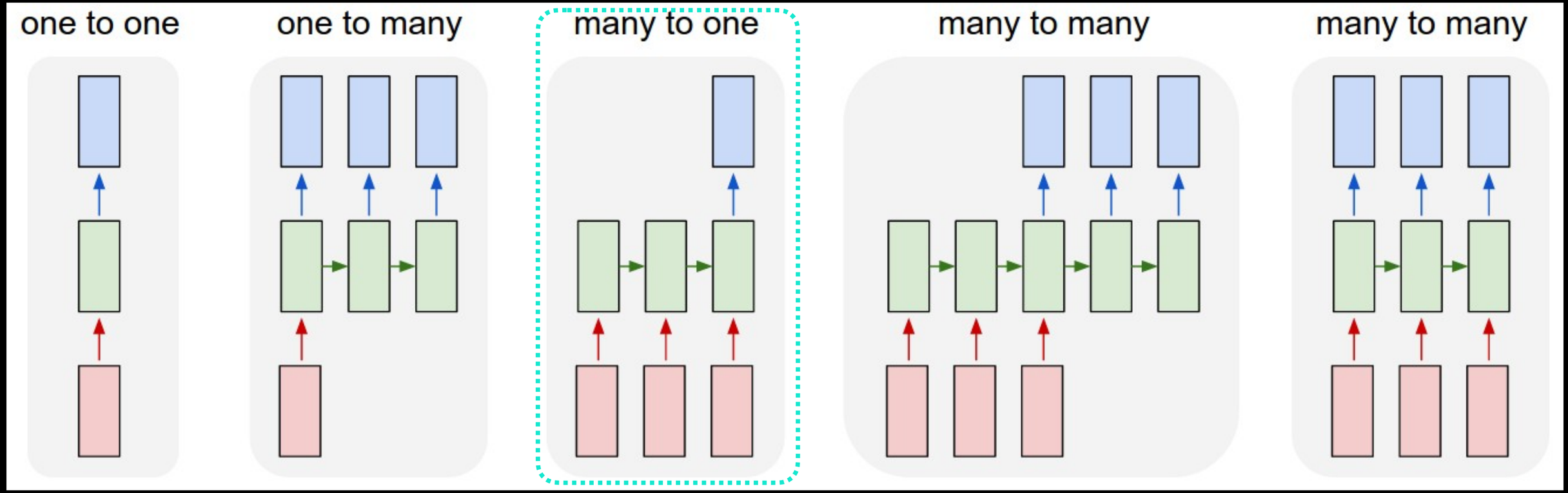
These will vary depending on the problem you're working on/what window size & horizon you decide to use.

`[horizon]`

```
Shape = [1]
```

Sequence problems

(also called seq2seq)

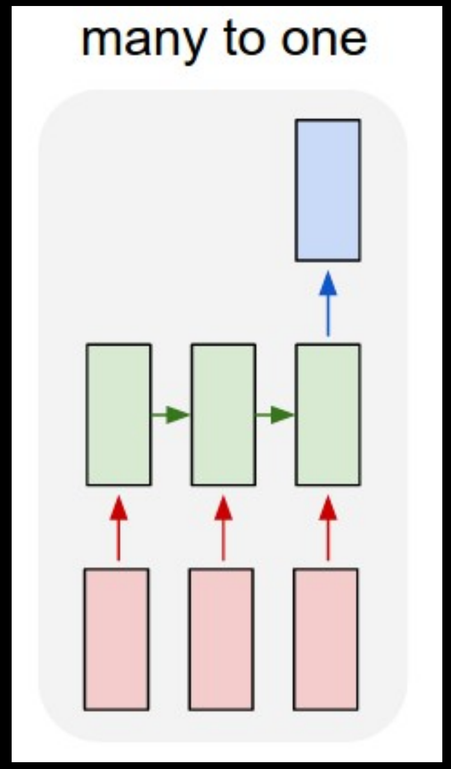


Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

BitPredict: seq2seq problems

Output [123.033] Horizon = 1

One day forecast for Bitcoin price



Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

One week of Bitcoin prices

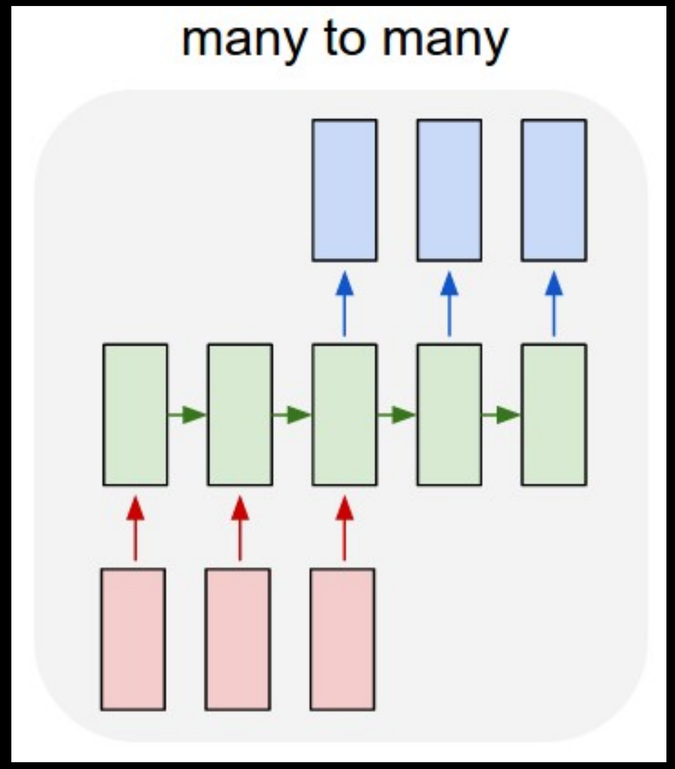
Window size = 7

Input [123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795]

BitPredict: seq2seq problems

3 day forecast of Bitcoin prices

Output `[123.033, 124.049, 125.961]` Horizon = 3



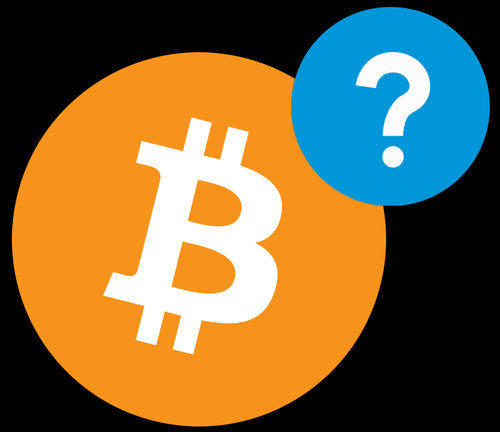
Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

One week of Bitcoin prices

Window size = 7

Input `[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795]`

Let’s try and predict the price of Bitcoin!



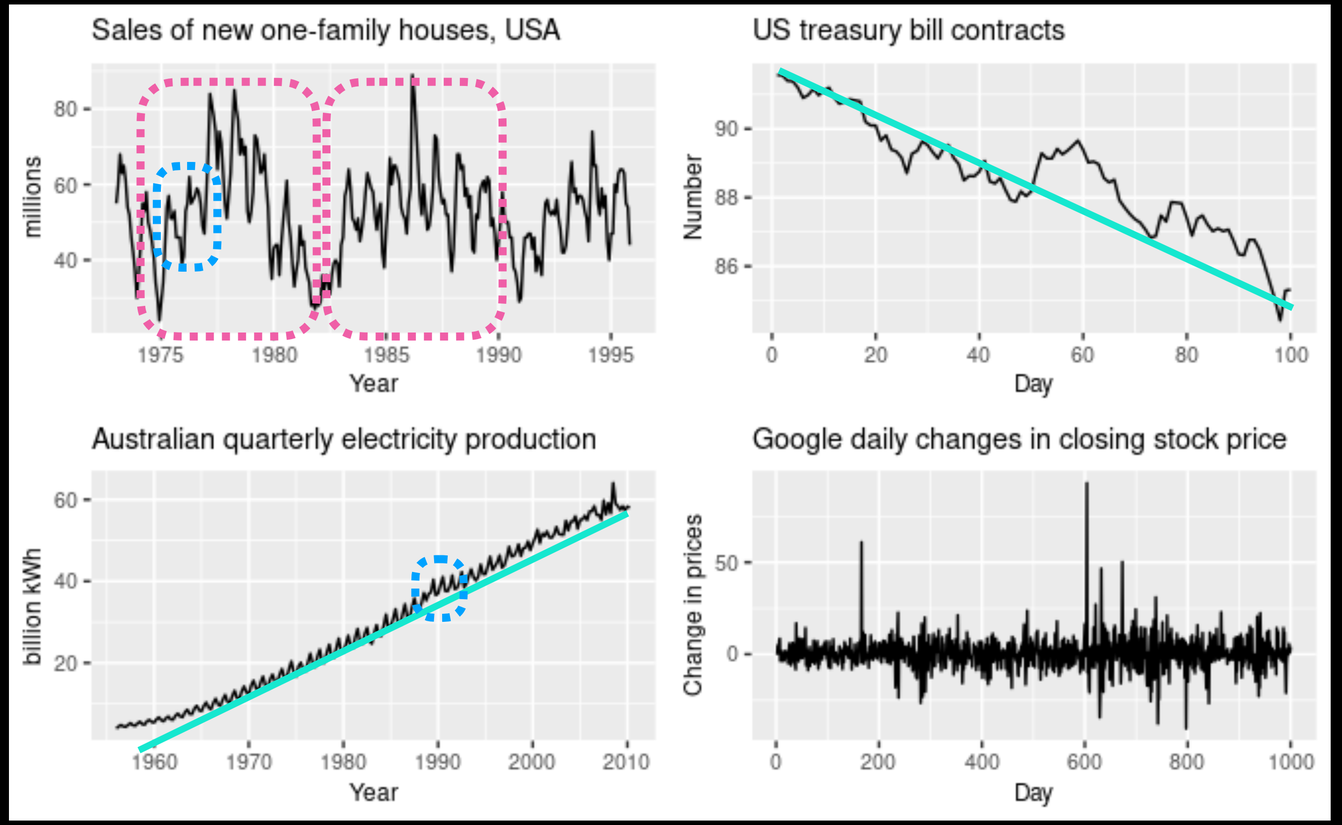
Types of time series

(various patterns)

Trend — time series has a clear long-term increase or decrease (may or may not be linear)

Seasonal — time series affected by seasonal factors such as time of year (e.g. increased sales towards end of year) or day of week

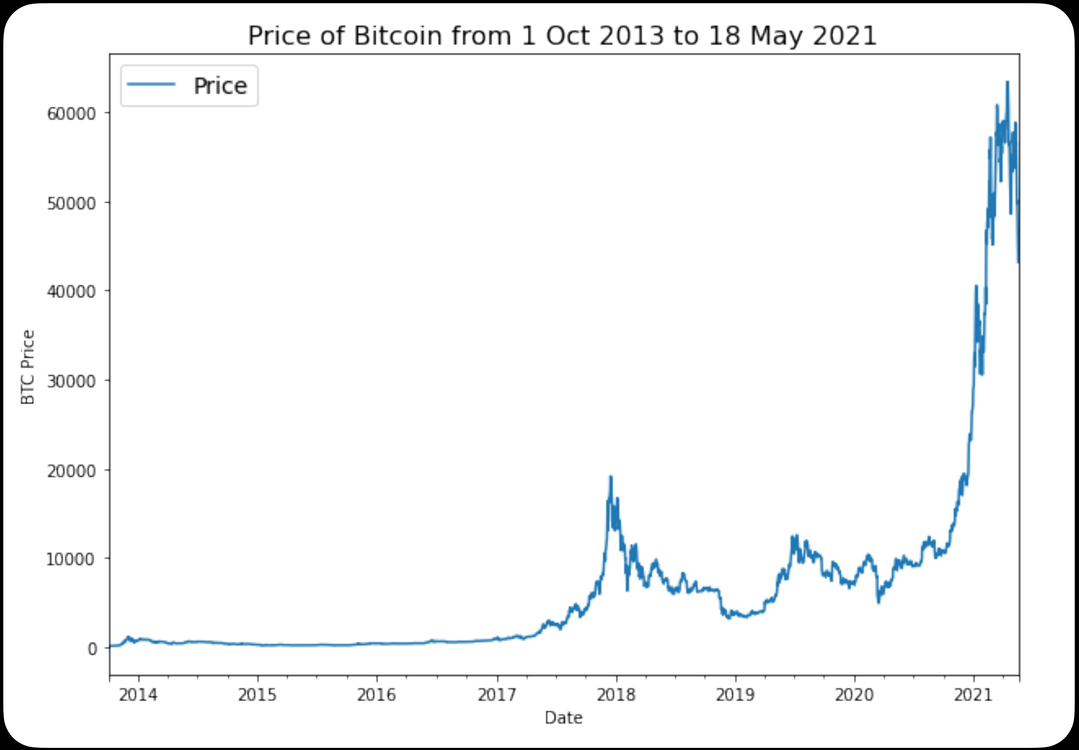
Cyclic — time series shows rises and falls over an unfixed period, these tend to be longer/more variable than seasonal patterns



Source: Figure 2.3: Four examples of time series showing different patterns from Forecasting: Principles and Practice 3rd Edition.

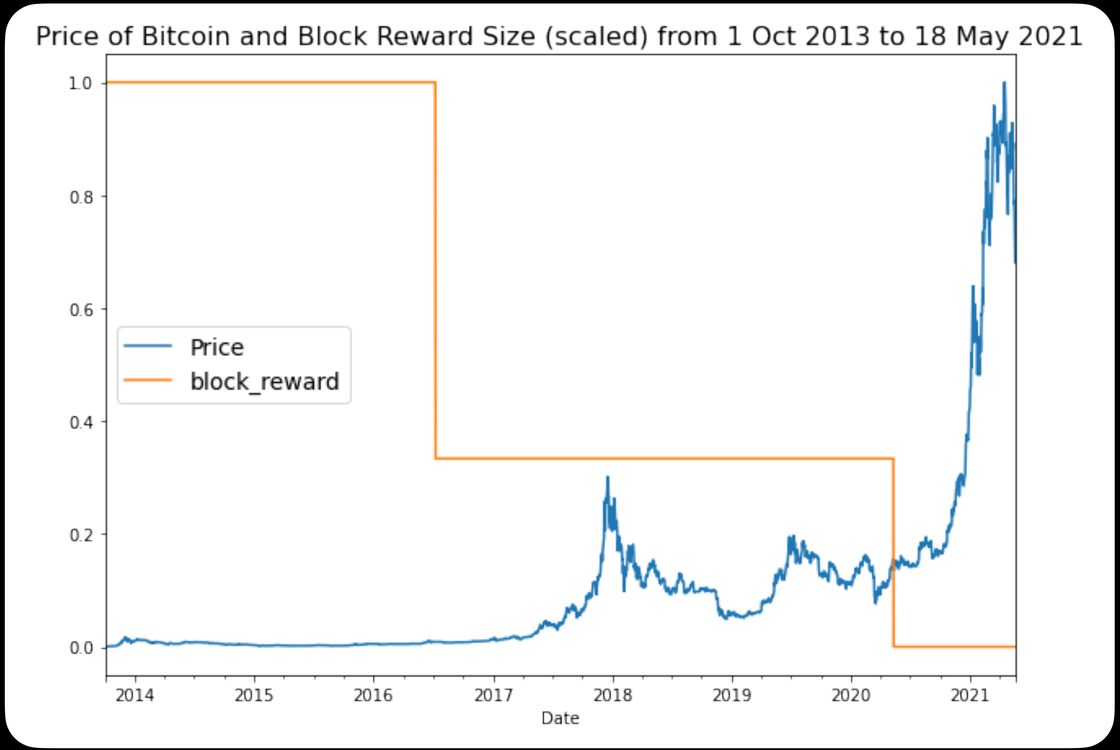
No patterns (random, likely not predictable)

Univariate and Multivariate time series data



Univariate

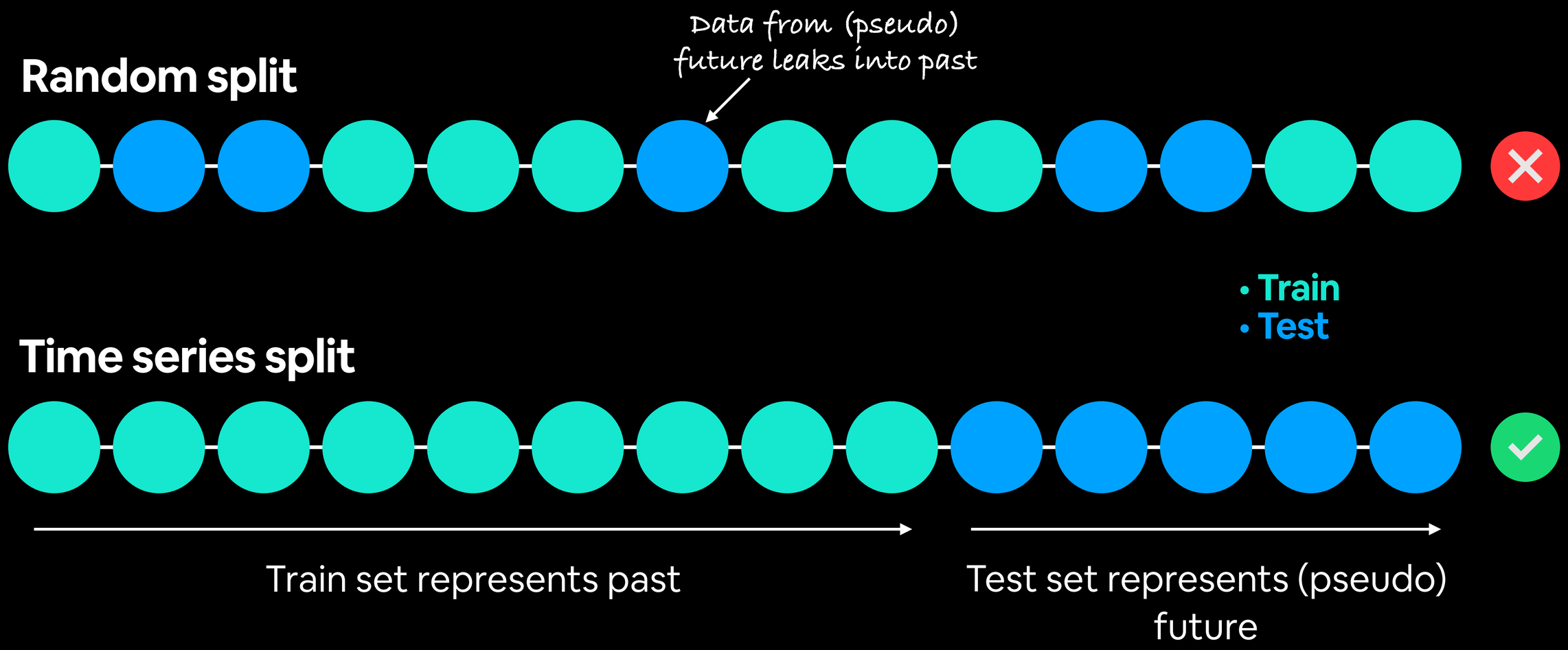
Only one variable (using the price of Bitcoin to predict the price of Bitcoin).



Multivariate

More than one variable (using the price of Bitcoin as well as the block reward size to predict the price of Bitcoin).

Time series train & test sets



(some common)

Regression evaluation metrics

Metric Name	Metric Forumla	TensorFlow code	When to use
Mean absolute error (MAE)	$\text{MAE} = \frac{\sum_{i=1}^{n} \lvert y_i - x_i \rvert}{n}$	`tf.keras.losses.MAE( )` or `tf.metrics.mean_absolute_error( )`	As a great starter metric for any regression problem.
Mean square error (MSE)	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$	`tf.keras.losses.MSE( )` or `tf.metrics.mean_square_error( )`	When larger errors are more significant than smaller errors.
Huber	$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \textbf{for } \lvert y - f(x) \rvert \le \delta, \\ \delta \lvert y - f(x) \rvert - \frac{1}{2}\delta^2 & \textbf{otherwise.} \end{cases}$	`tf.keras.losses.Huber( )`	Combination of MSE and MAE. Less sensitive to outliers than MSE.

(some common)

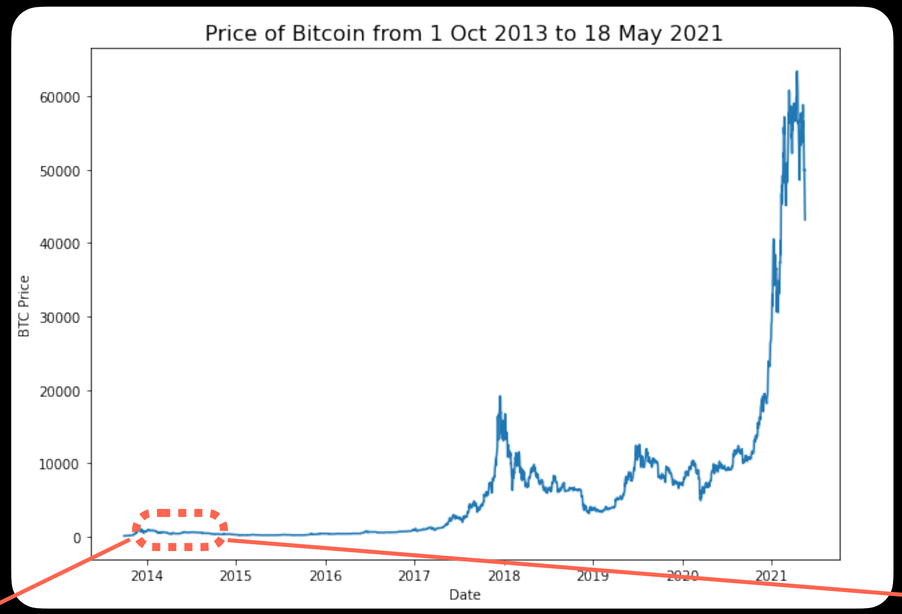
Time series forecasting evaluation metrics

Metric Name	Metric Forumla	TensorFlow code	When to use
Mean absolute error (MAE)	$\text{MAE} = \frac{\sum_{i=1}^{n} \lvert y_i - x_i \rvert}{n}$	`tf.keras.losses.MAE( )` or `tf.metrics.mean_absolute_error( )`	As a great starter metric for any regression problem.
Mean square error (MSE)	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$	`tf.keras.losses.MSE( )` or `tf.metrics.mean_square_error( )`	When larger errors are more significant than smaller errors.
Root mean squared error (RMSE)	$\text{RMSE} = \sqrt{\text{MSE}}$	`tf.sqrt(tf.metrics.mean_square_error( ))`	Similar to MSE but as interpretable as MAE (RMSE is in same units as target).
(Symmetric) Mean absolute percentage error (MAPE/ sMAPE)	$\text{MAPE} = \textbf{mean}(\lvert 100e_t/y_t \rvert)$	`tf.keras.metrics.mean_absolute_percentage_error()`	Recommend not to use by Hyndman & Koehler.
Mean absolute scaled error (MASE)	$q_j = \frac{e_j}{\frac{1}{T-1}\sum_{t=2}^{T} \lvert y_t - y_{t-1} \rvert}.$	Custom or See sktime's `mase_loss()`	A scaled error is **>1 if the forecast is worse than the naive** and **<1 if the forecast is better than the naive**.

Source: For more explanations and details on time series forecasting evaluation metrics see Forecasting: Principles and Practice chapter 5.8

Windows and horizons

Windows and horizons



Turning a time series into a supervised learning problem

```
# Window for one week with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] -> [123.033]
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033] -> [124.049]
[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049] -> [125.961]
```

Window size = 7

Data

Horizon = 1

Label

Windows and horizons

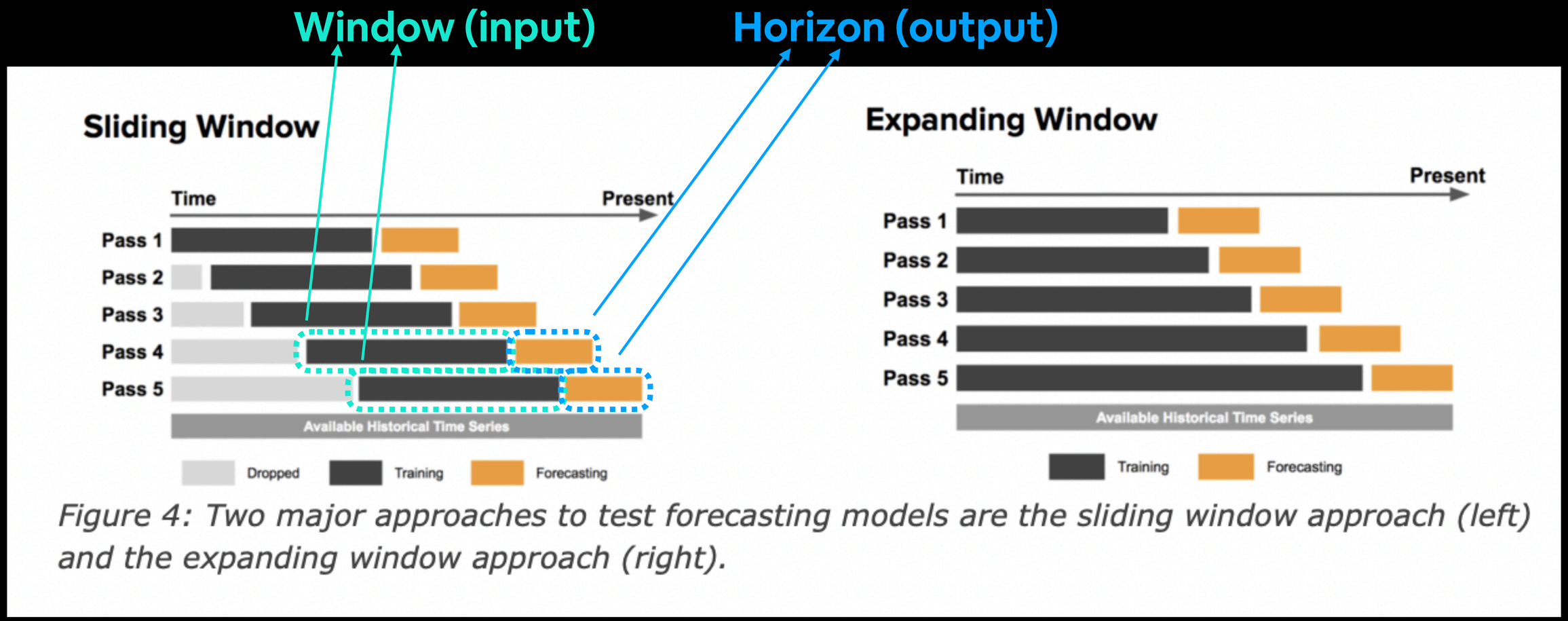


Figure 4: Two major approaches to test forecasting models are the sliding window approach (left) and the expanding window approach (right).

Source: Forecasting at Uber blog post

Windows and horizons

```
# Window for one week with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] -> [123.033]
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033] -> [124.049]
[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049] -> [125.961]
```

Window size = 7

Horizon = 1

(These values are tuneable hyperparameters)

"Can we predict the next _____ given the past _____?"

Window size (input) — number of time steps of historical data used to predict horizon

Horizon (output) — number of time steps to predict into the future

Data

Label

BitPredict: inputs and outputs

```
# Window for one week with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] -> [123.033]
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033] -> [124.049]
[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049] -> [125.961]
```

Window size = 7

Horizon = 1

(These values are tuneable hyperparameters)

"Can we predict the price of Bitcoin tomorrow given the past week of prices?"

Window size (input) — number of time steps of historical data used to predict horizon

Data

Horizon (output) — number of time steps to predict into the future

Label

BitPredict: inputs and outputs

```
# Window for one month with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655,
121.795, 123.033, 124.049, 125.961, 125.279, 125.927,
126.383, 135.241, 133.203, 142.763, 137.923, 142.951,
152.551, 160.338, 164.314, 177.633, 188.297, 200.701,
180.355, 175.031, 177.696, 187.159, 192.756, 197.400] -> [196.024]

[125.455, 108.584, 118.674, 121.338, 120.655, 121.795,
123.033, 124.049, 125.961, 125.279, 125.927, 126.383,
135.241, 133.203, 142.763, 137.923, 142.951, 152.551,
160.338, 164.314, 177.633, 188.297, 200.701, 180.355,
175.031, 177.696, 187.159, 192.756, 197.400, 196.024] -> [198.048]
```

Window size = 30

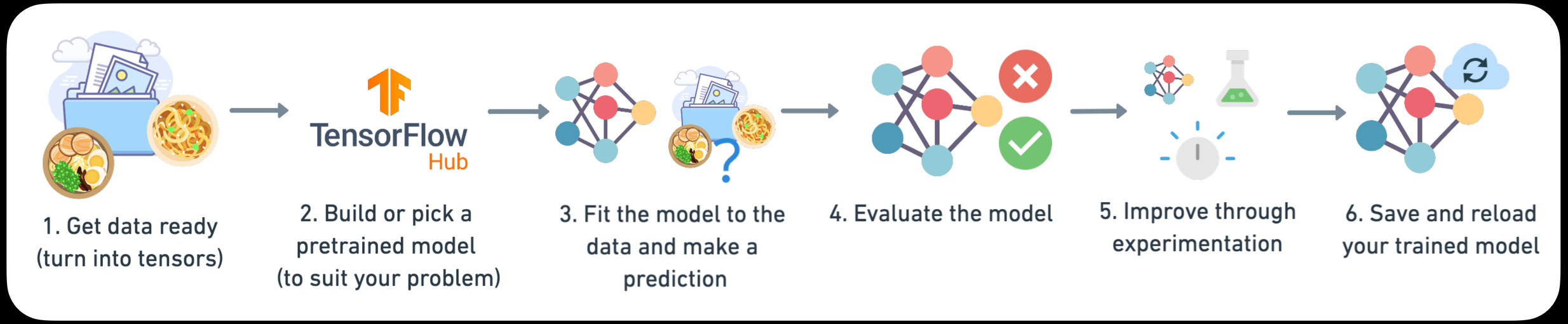
Horizon = 1

(These values are tuneable hyperparameters)

Data

Label

Steps in modelling with TensorFlow



1. Turn all data into numbers (neural networks can't handle text/natural language)
2. Make sure all of your tensors are the right shape (pad sequences which don't fit)

Experiments we're running

Experiment	Model
0	Naive model (baseline)
1	Dense model (window size = 7, horizon = 1)
2	Same architecture as model 1 (window size = 30, horizon = 1)
3	Same architecture as model 1 (window size = 30, horizon = 7)
4	Conv1D
5	LSTM
6	Same architecture as model 1 (but with multivariate data)
7	N-BEATS algorithm
8	Ensemble (multiple models stacked together)
9	Future prediction model
10	Same architecture as model 1 (but with turkey 🦃 data introduced)

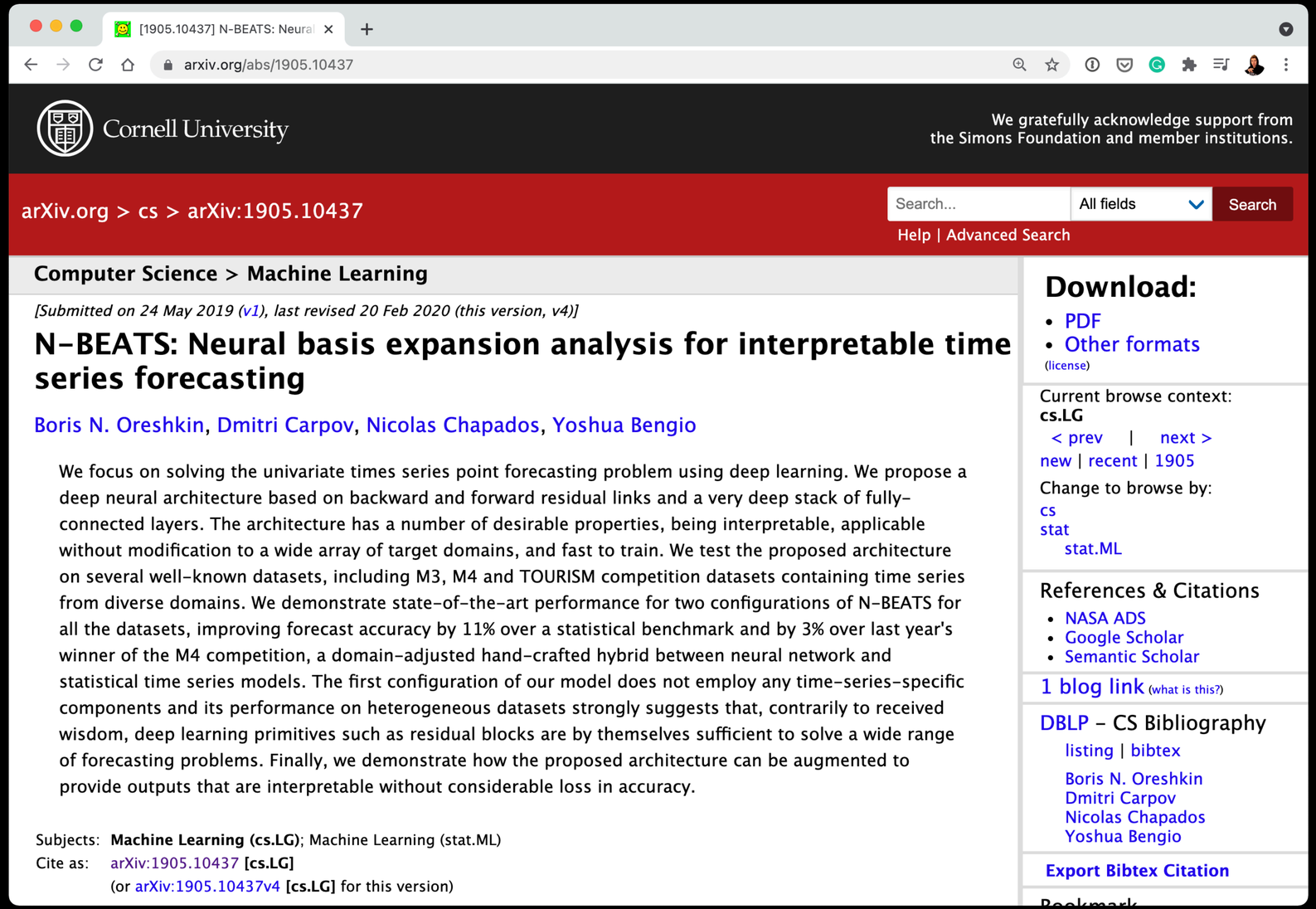
```
59 Machine Learning Engineer*
60 ----
61
62 1. Download a paper
63 2. Implement it
64 3. Keep doing this until you have skills
```

- George Hotz

*Machine Learning Engineering also involves building infrastructure around your models/ data preprocessing steps

What we're doing

Replicating this paper



Source: https://arxiv.org/abs/1905.10437

Source: Figure 1 (proposed architecture) N-BEATS paper

Window Horizon

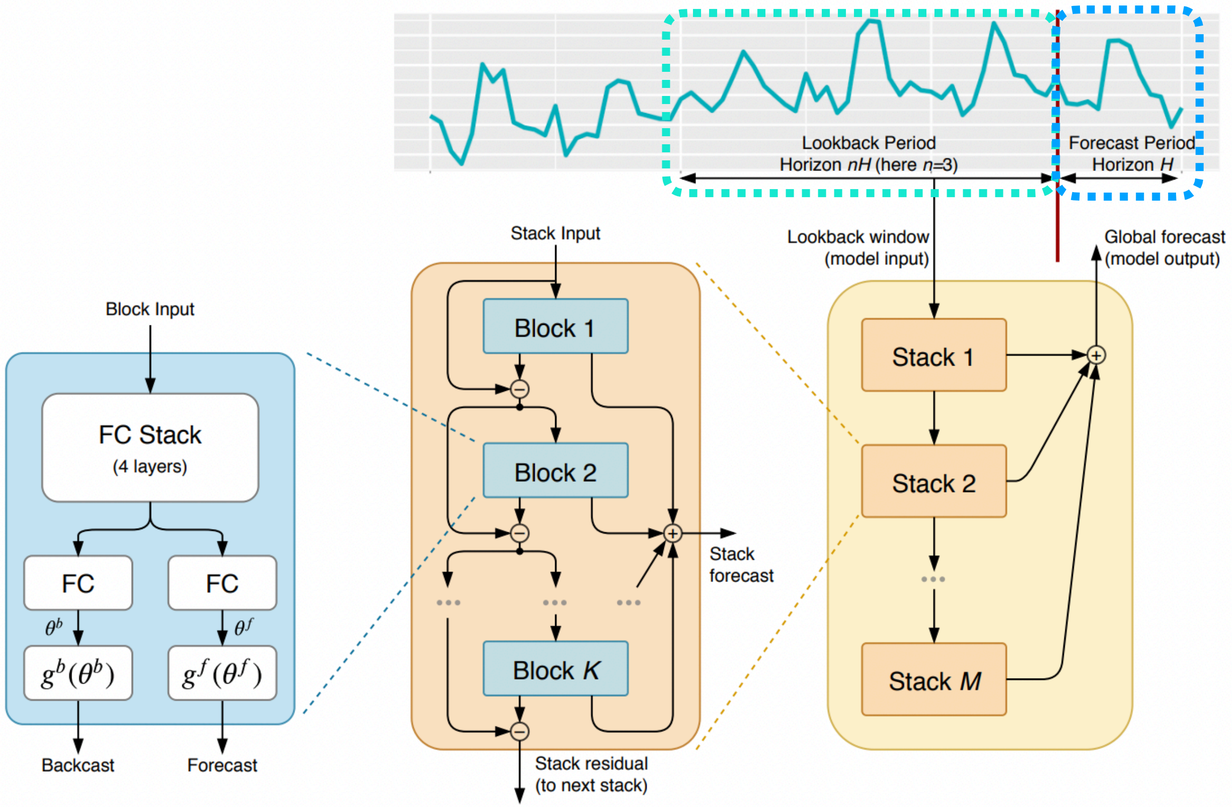
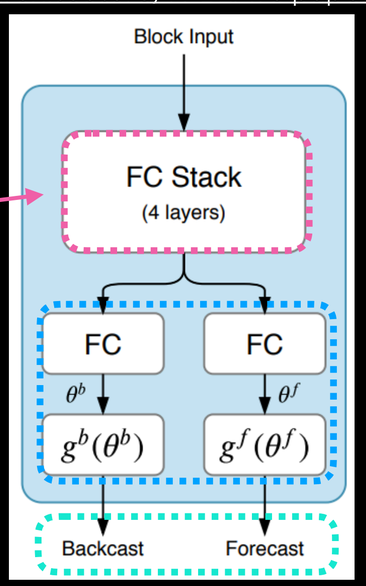


Figure 1: Proposed architecture. The basic building block is a multi-layer FC network with ReLU nonlinearities. It predicts basis expansion coefficients both forward, θ^f, (forecast) and backward, θ^b, (backcast). Blocks are organized into stacks using doubly residual stacking principle. A stack may have layers with shared g^b and g^f. Forecasts are aggregated in hierarchical fashion. This enables building a very deep neural network with interpretable outputs.

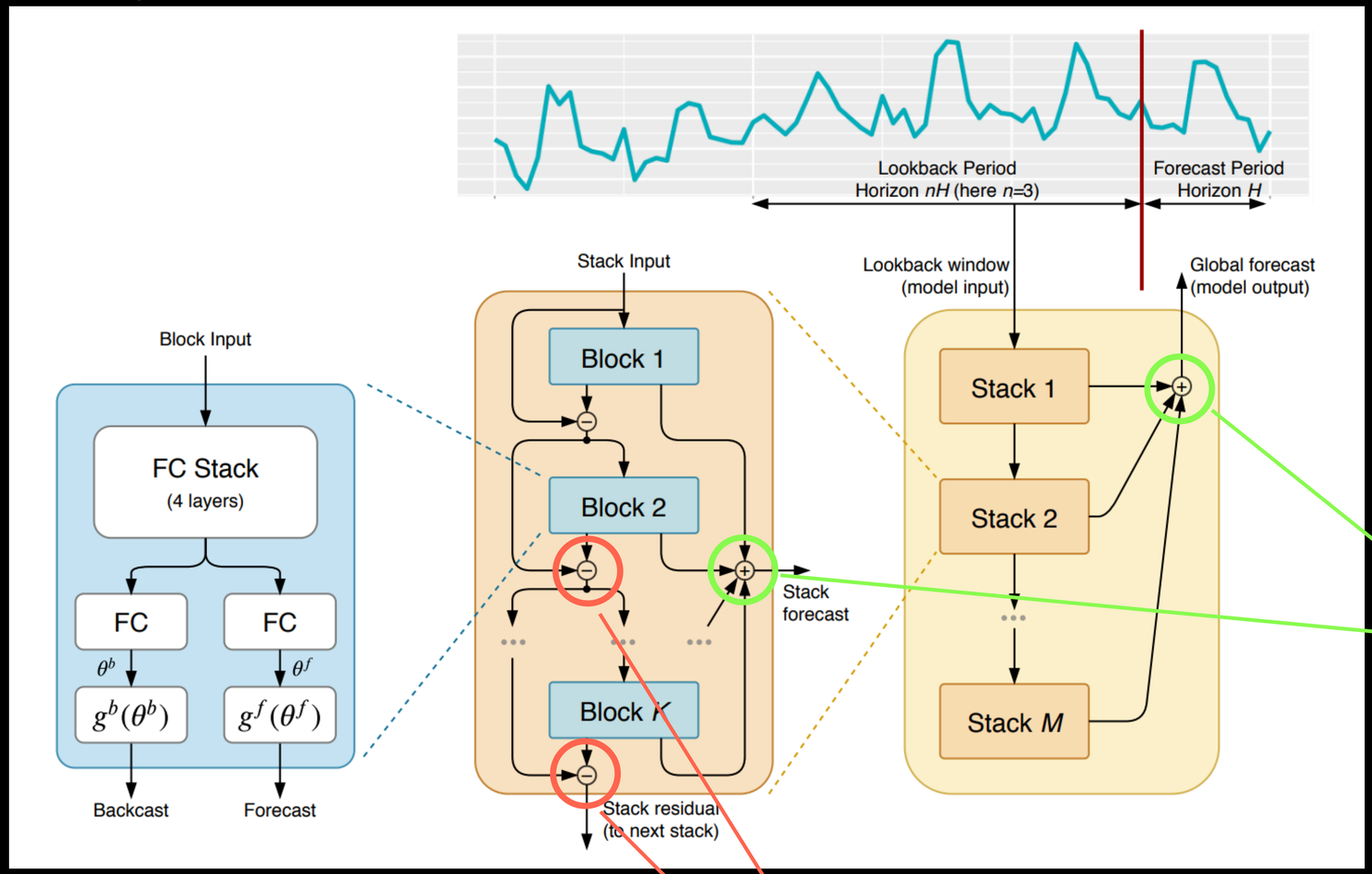
```python
# Create NBeatsBlock custom layer
class NBeatsBlock(tf.keras.layers.Layer):
  def __init__(self, # the constructor takes all the hyperparameters for the layer
               input_size: int,
               theta_size: int,
               horizon: int,
               n_neurons: int,
               n_layers: int,
               **kwargs): # takes care of all of the arguments for the parent class (input_shape, trainable, name)
    super().__init__(**kwargs)
    self.input_size = input_size
    self.theta_size = theta_size
    self.horizon = horizon
    self.n_neurons = n_neurons
    self.n_layers = n_layers

    # Block contains stack of 4 fully connected layers each has ReLU activation
    self.hidden = [tf.keras.layers.Dense(n_neurons, activation="relu") for _ in range(n_layers)]

    # Output of block is a theta layer with linear activation
    self.theta_layer = tf.keras.layers.Dense(theta_size, activation="linear", name="theta")

  def call(self, inputs): # the call method is what runs when the layer is called
    x = inputs
    for layer in self.hidden: # pass inputs through each hidden layer
      x = layer(x)
    theta = self.theta_layer(x)

    # Output the backcast and forecast from theta
    backcast, forecast = theta[:, :self.input_size], theta[:, -self.horizon:]

    return backcast, forecast
```

Source: Figure 1 (proposed architecture) N-BEATS paper



N-BEATS basic block replication with TensorFlow layer subclassing (section 3.1 of N-BEATS paper)

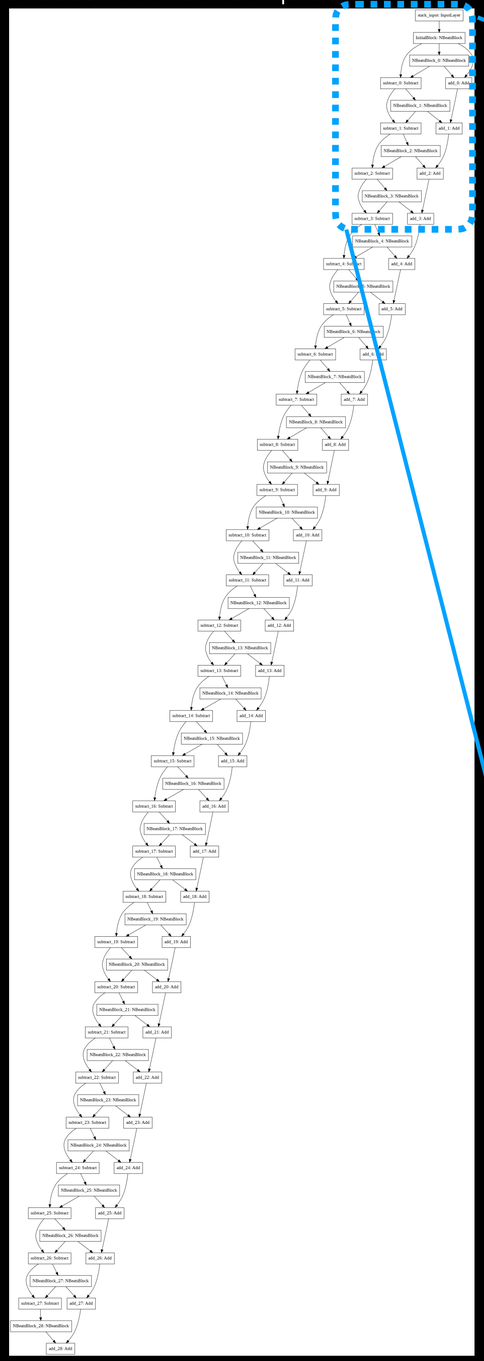
Source: Figure 1 (proposed architecture) N-BEATS paper

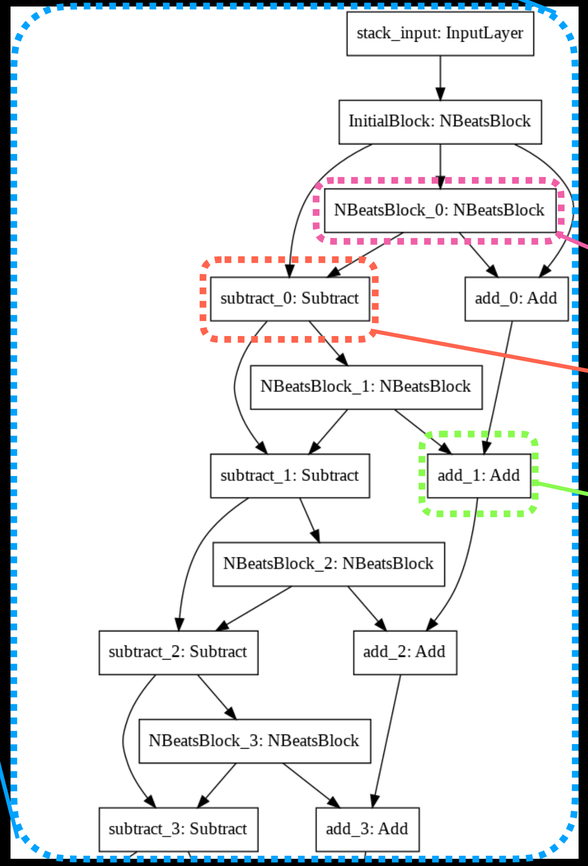


`tf.keras.layers.add(inputs)`

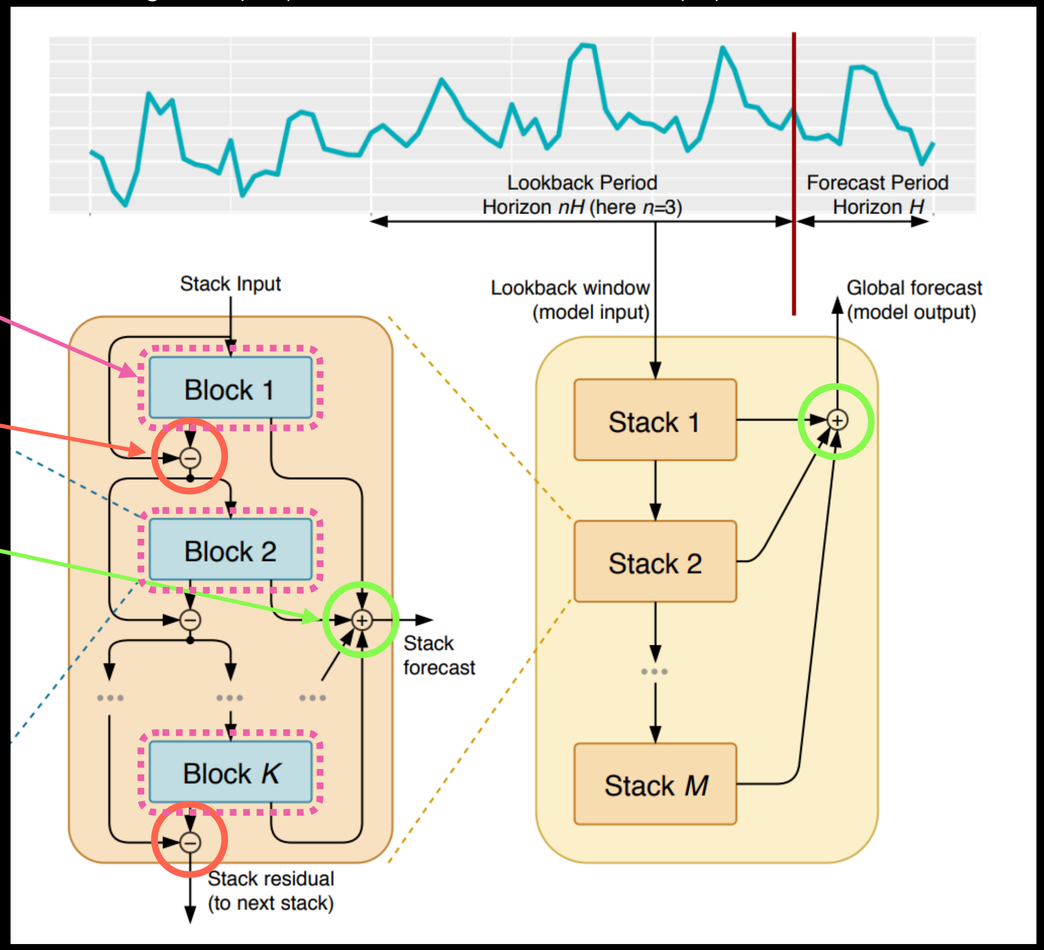
`tf.keras.layers.subtract(inputs)`

Model 7: N-BEATS replica model

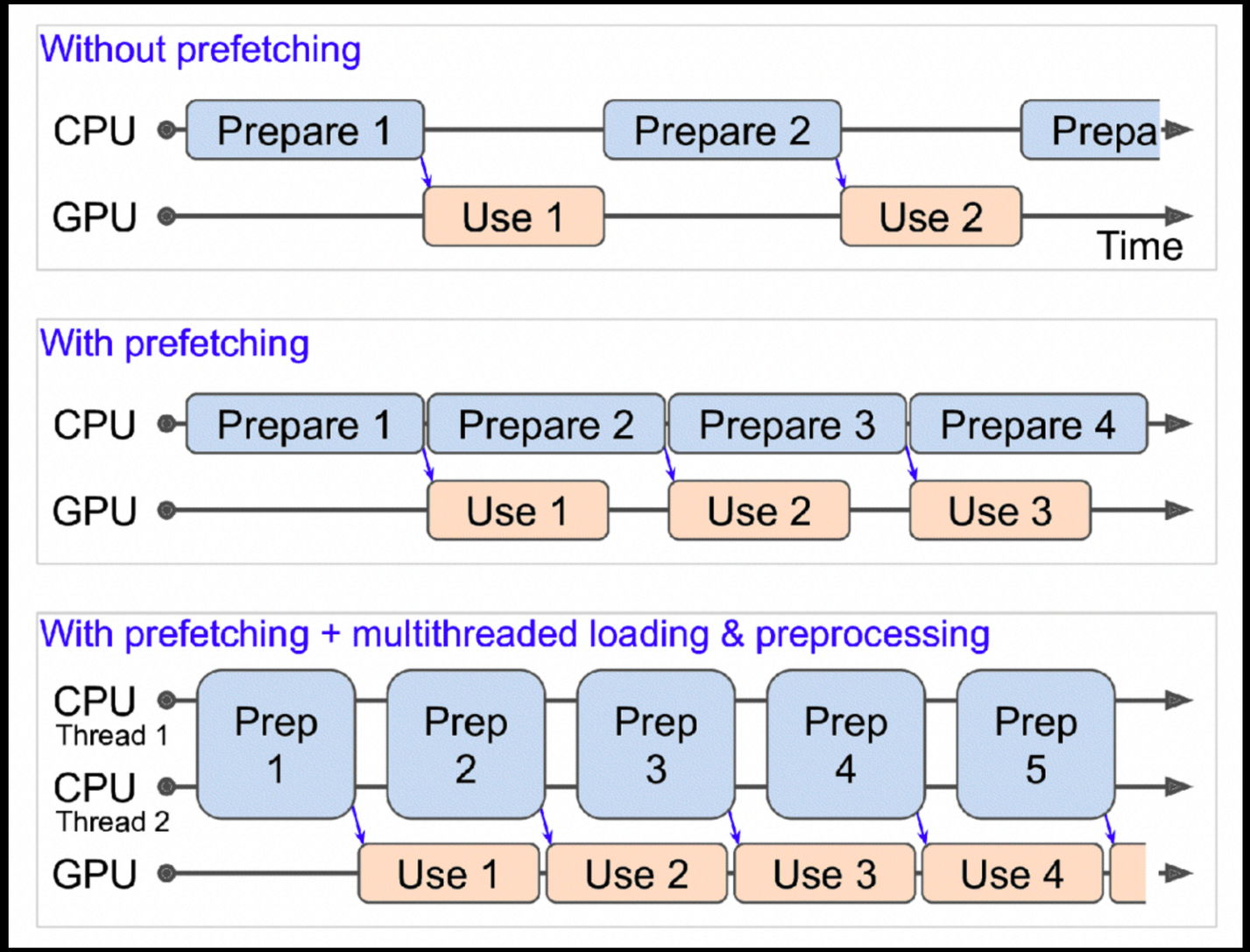




Source: Figure 1 (proposed architecture) N-BEATS paper



Prefetching



Source: Page 422 of Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book by Aurélien Géron

What is an ensemble model?

Single Model

Smart level 7

Ensemble

Smart level 10

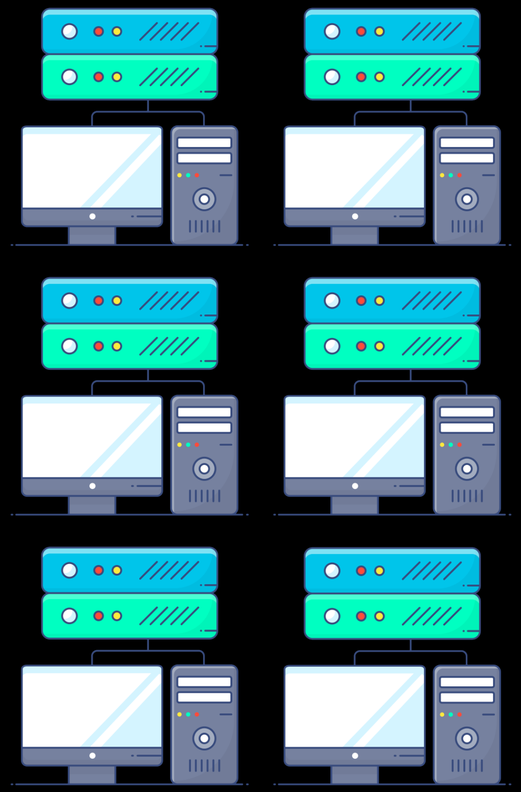
What is an ensemble model?

Smart level 7

Smart level 7

Example Forecasting Problems at Uber

Market demand



Compute requirements

Marketing campaigns

Source: Forecasting at Uber blog post

Prediction intervals/uncertainty estimates

95 percent prediction interval

Large periods of uncertainty around big events

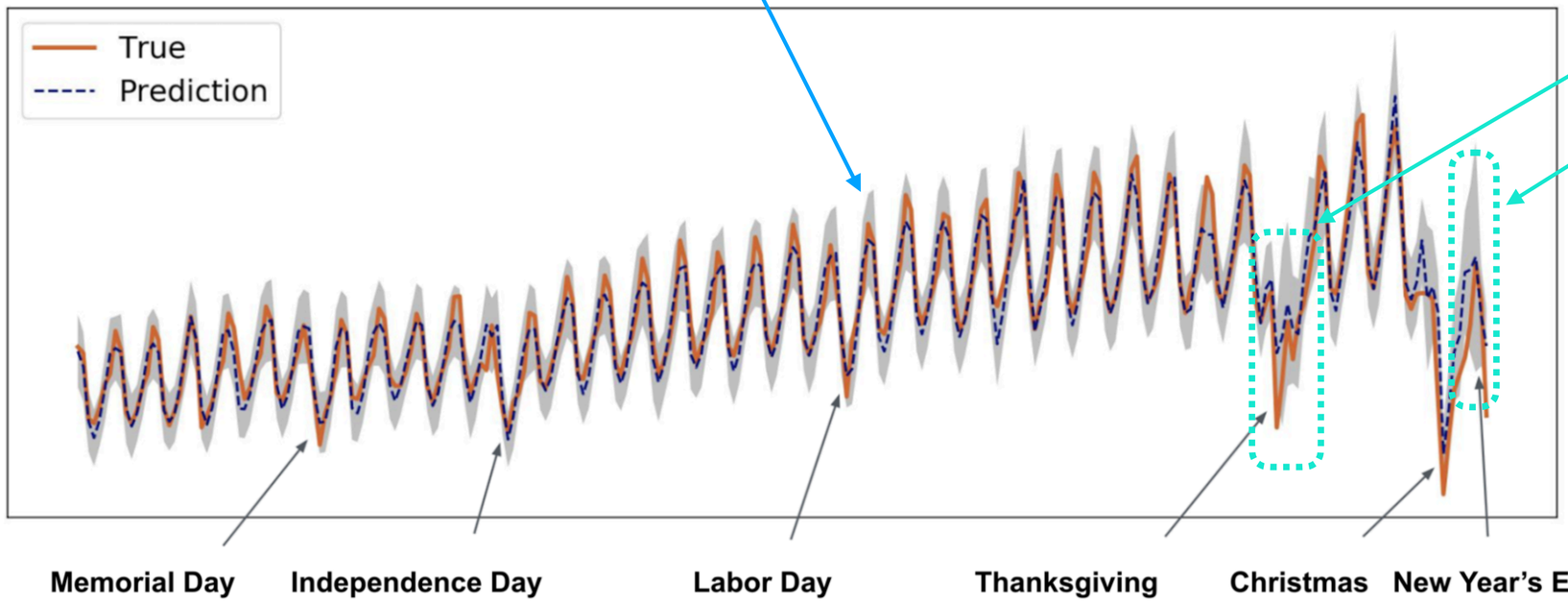
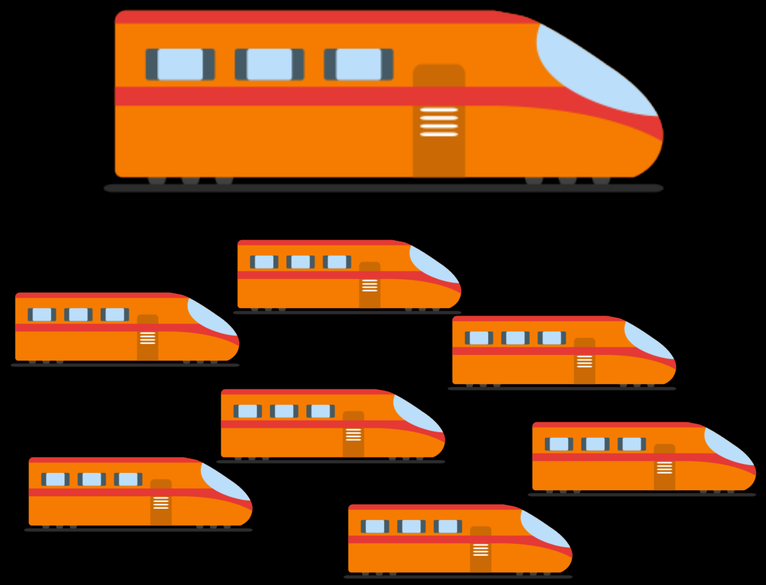


Figure 2: Daily completed trips in San Francisco during eight months of the testing set. True values are represented by the orange solid line, and predictions by the blue dashed line, where the 95 percent prediction band is shown as the grey area. (Note: exact values are anonymized.)

Source: Figure 2 from Engineering Uncertainty Estimation in Neural Networks for Time Series Prediction at Uber

Types of Uncertainty

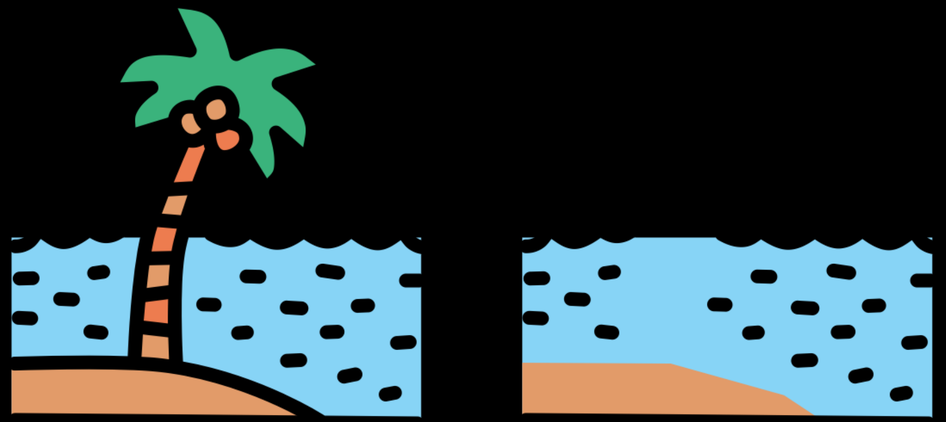
Aleatoric Uncertainty (data/subway uncertainty)



- **Cannot** be reduced with more data

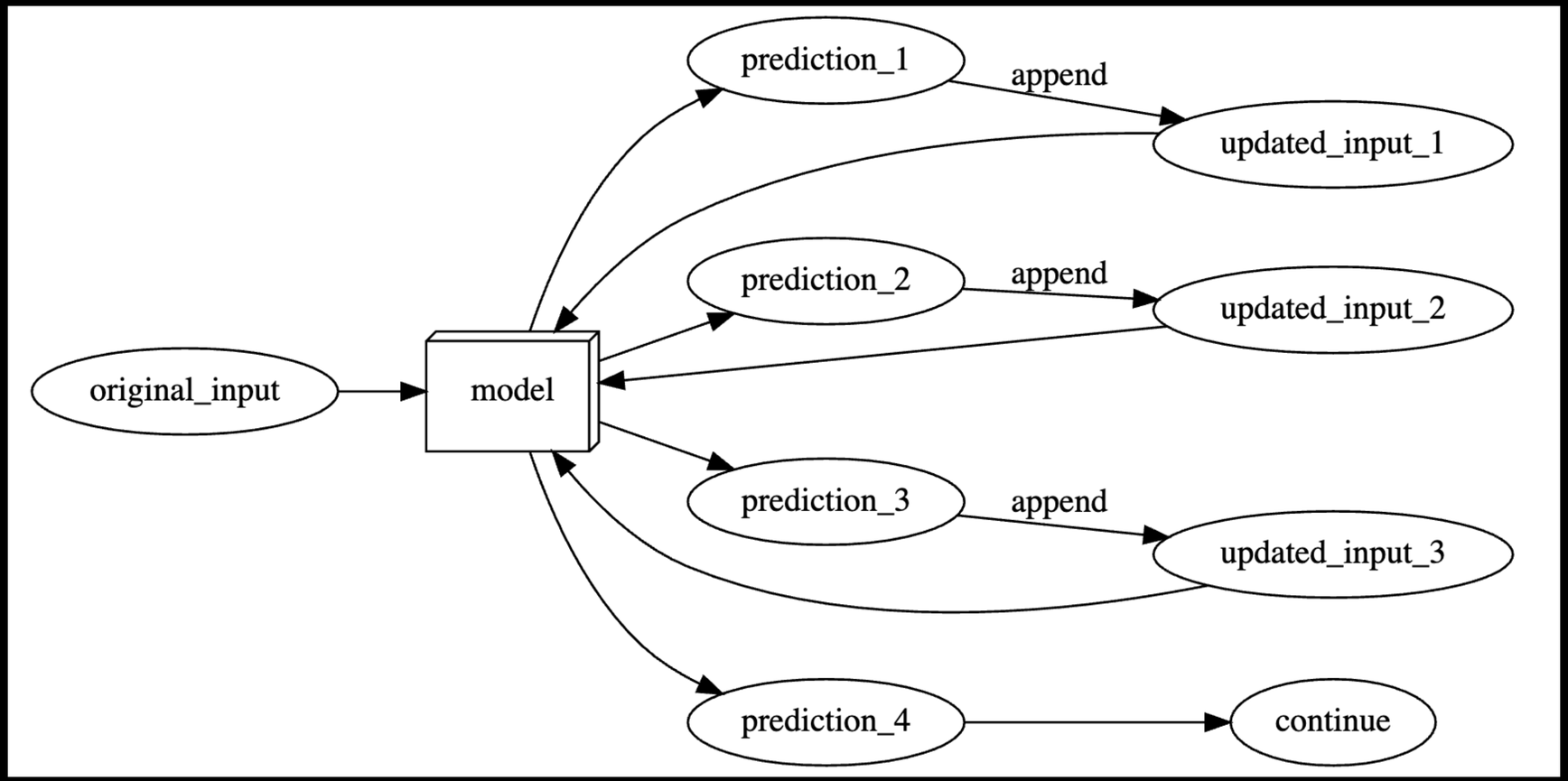
(when measuring prediction intervals, we're estimating a form of aleatoric uncertainty)

Epistemic Uncertainty (model/coconut uncertainty)



- **Can** be reduced with more data

Making time series forecasts



For time series forecasts, you have to retrain a model every time you make a prediction.

(as new data comes in, retrain a model and make a prediction)

The turkey problem

The turkey problem

(why forecasting is BS)

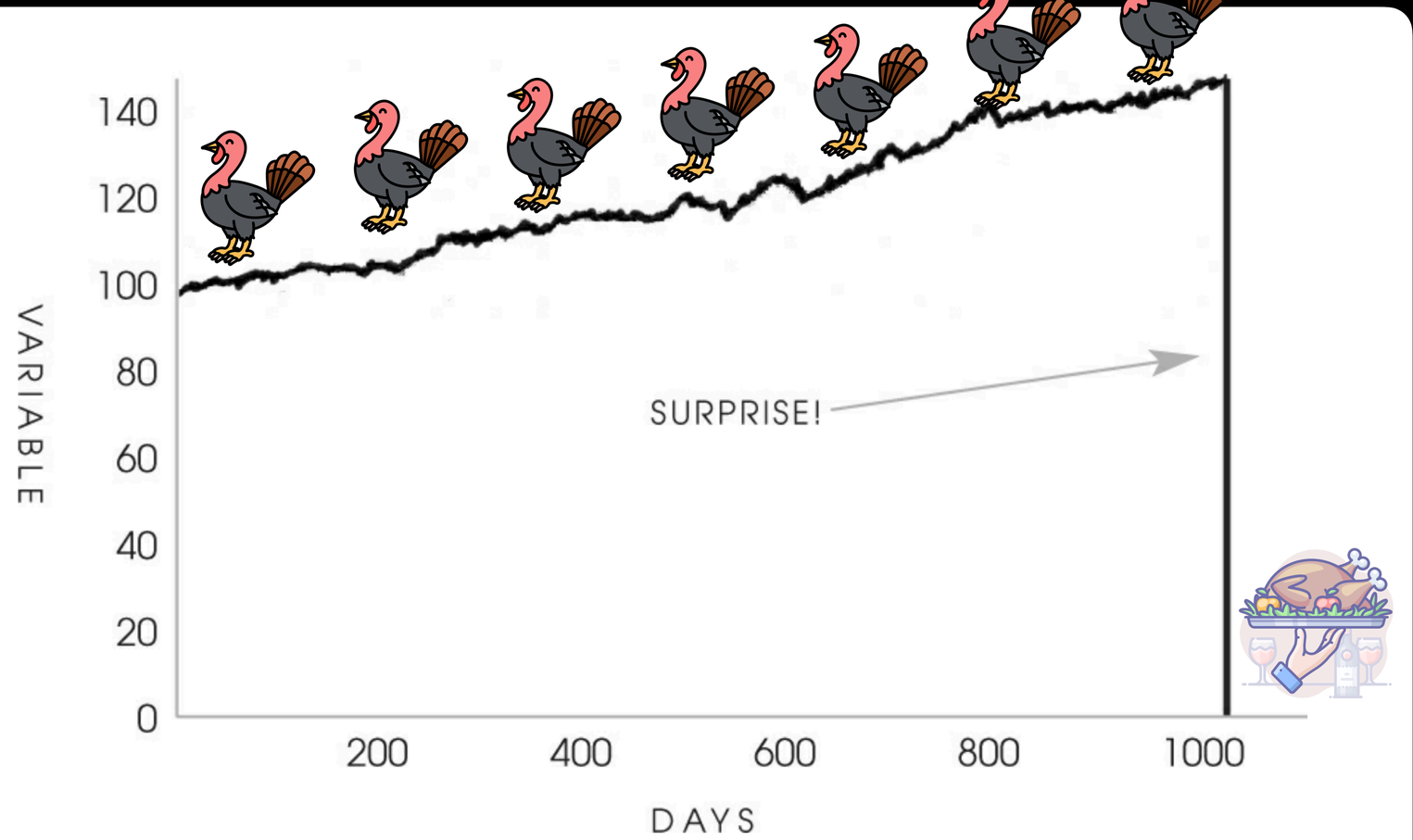
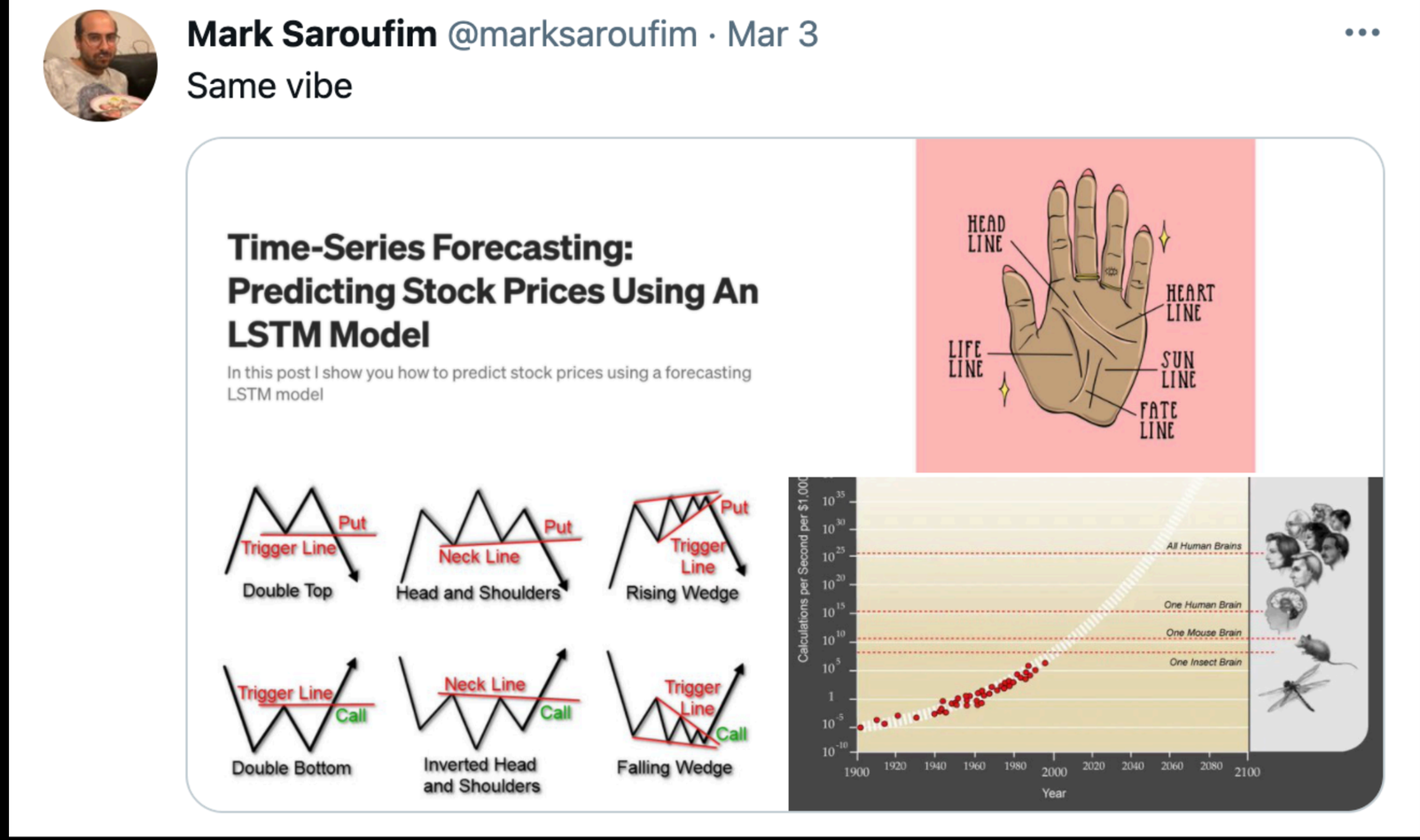


FIGURE 1: ONE THOUSAND AND ONE DAYS OF HISTORY

A turkey before and after Thanksgiving. The history of a process over a thousand days tells you nothing about what is to happen next. This naïve projection of the future from the past can be applied to anything.

Source: Page 41 of The Black Swan: The Impact of the Highly Improbable by Nassim Nicholas Taleb



Source: Mark Saroufim's Twitter

Improving a model

(from a model's perspective)

```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(4, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_subset, y_train_subset, epochs=5)
```

Smaller model


```python
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(100, activation="relu"),
  tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_full, y_train_full, epochs=100)
```

Larger model

Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate
- Fitting on more data
- Fitting for longer

(because you can alter each of these, they're hyperparameters)

Improving a model

(from a data perspective)

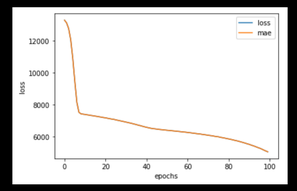
Method to improve a model (reduce overfitting)	What does it do?
More data	Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza).
Data augmentation (usually for images)	Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns.
Better data (**feature engineering**)	Not all data samples are created equally. Removing poor samples from or adding better samples/**engineering new features** in your dataset can improve your model's performance.
Use transfer learning	Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.

The machine learning explorer's motto

"Visualize, visualize, visualize"

Data

Model



Training

Predictions

It's a good idea to visualize these as often as possible.

The machine learning practitioner's motto

"Experiment, experiment, experiment"


(try lots of things and see what tastes good)