Intro to Neural Nets

RNNs for Forecasting

Today's Agenda

Basic Setup for Forecasting

- Train vs. validation / test, and setting up data.

Sequence Data

- Temporal (1D) Convolution
- Recurrent Neural Networks (RNNs):
  - SimpleRNN
  - LSTM (Long Short-Term Memory)
  - GRU (Gated Recurrent Unit)

Other Stuff

- Bidirectional RNN
- Advanced Forecasting

Basic Structure for Forecasting Problems

Earlier Data Serves as Training, Later Data Serves as Validation / Holdout

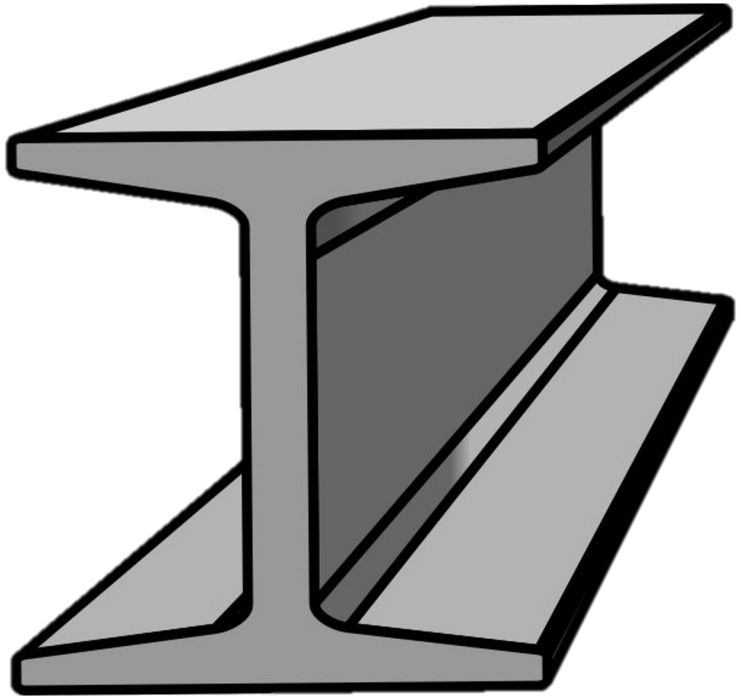
- Never train on future data and test on historical data.
- Temporal precedence often has a big effect on predictive performance.

Structuring Time Series Data for Prediction

- We need to construct sequences of fixed length followed by a prediction at some point in the future.

Handling Data

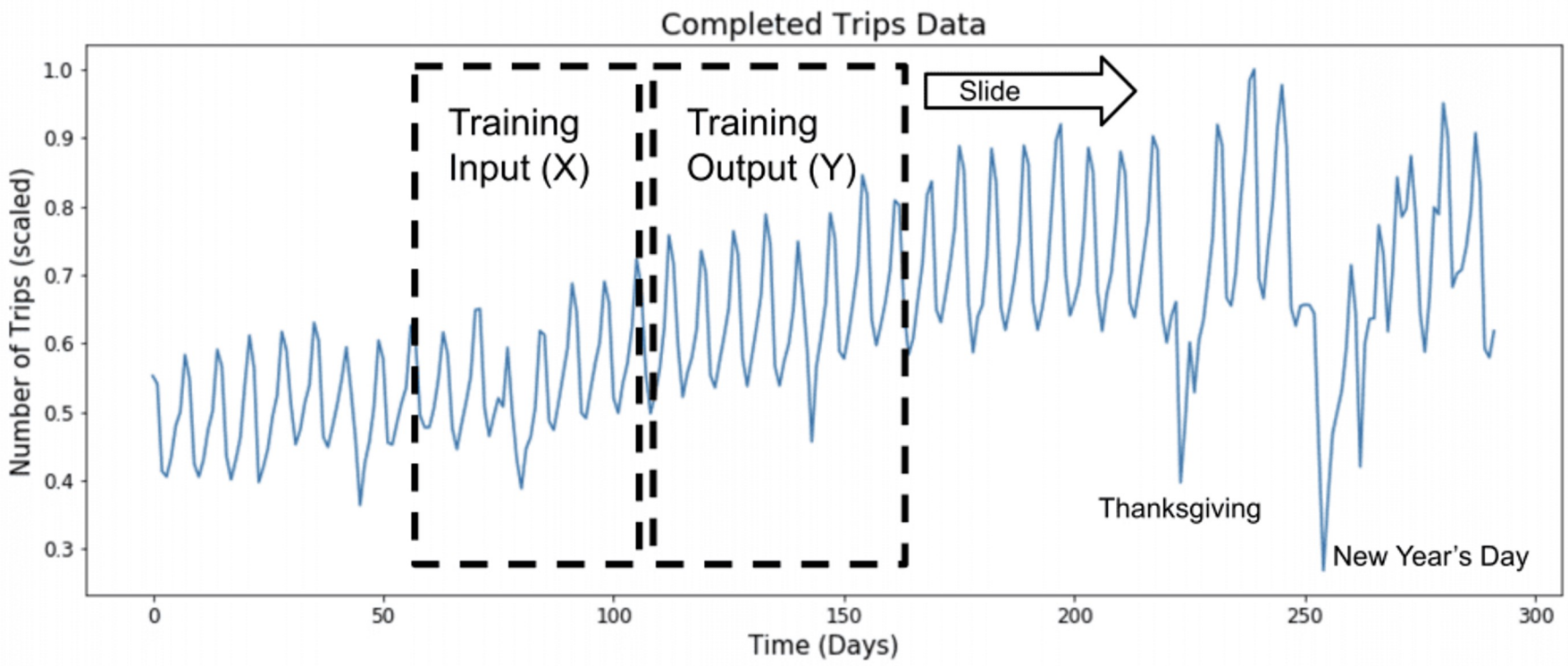
- TensorFlow dataset generator / iterator so we do not have to store many overlapping sequences of values in memory.
- Or we can make the sequences / labels from scratch.



Preparing Data

Constructing our Observations / Labels

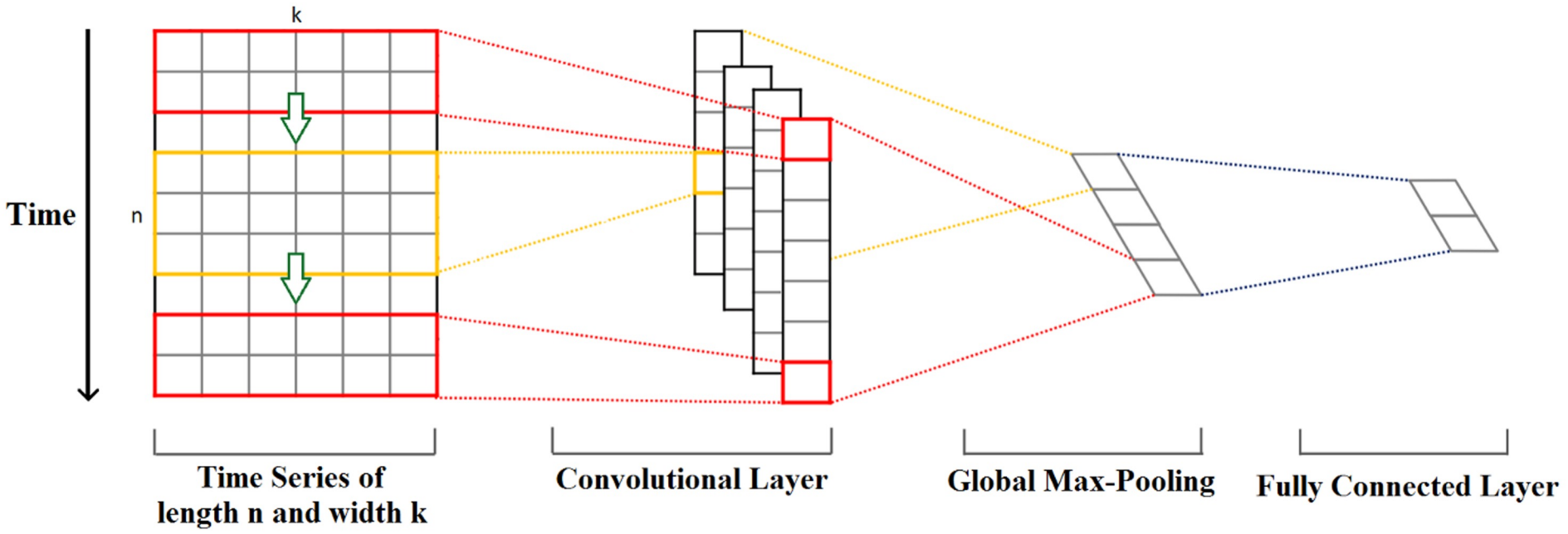
- This might be one observation (a sequence of prior values, i.e., x's, to one or more outcomes, i.e., y's)



Temporal 1D Convolution

1D Convolution Accomplishes Same Goal as 2D

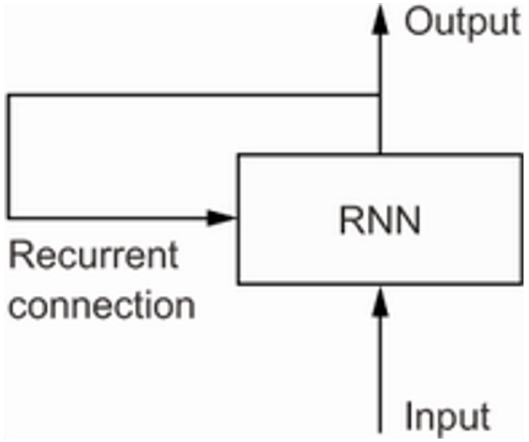
- It only considers arrangement of features in one dimension (temporal ordering).
- Compresses into shorter sequences, across the entire set of features (just as 2D Conv compresses matrices into smaller matrices, across the entire set of input channels (e.g., RGB).



RNN: Processing /w Memory

Incorporating Memory into a NN

- We include a feedback loop, where output feeds back into the same layer alongside the next input in the sequence (each gets its own separate set of weights, and they are able to interact with one another.



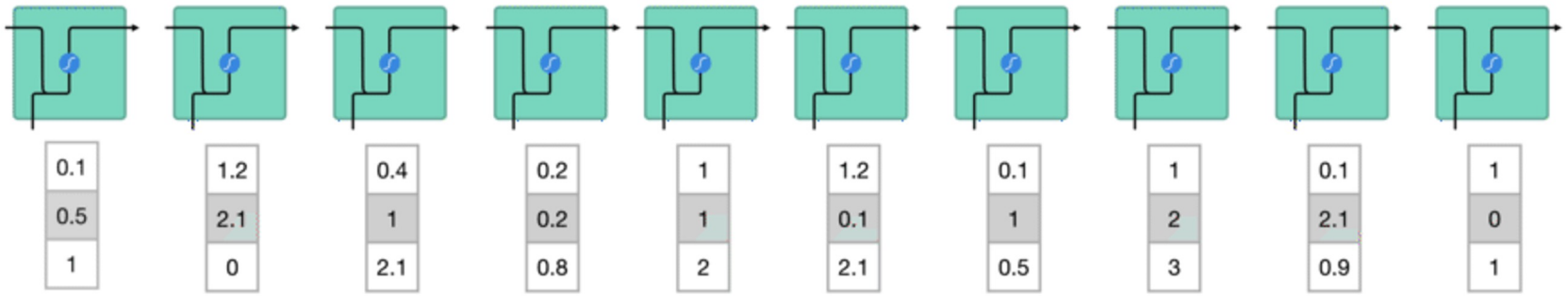
Listing 10.14 More-detailed pseudocode for the RNN

```
state_t = 0
for input_t in input_sequence:
    output_t = activation(dot(W, input_t) + dot(U, state_t) + b)
    state_t = output_t
```

Keras RNN Layers

SimpleRNN

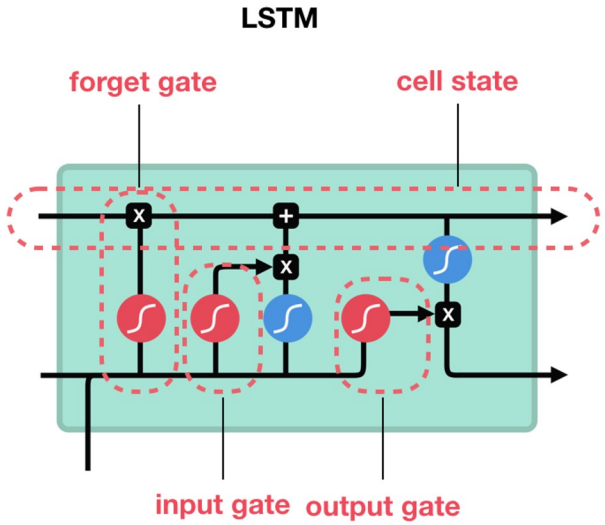
- We can, in practice, unroll a one-lag RNN into a Dense Network, as follows.
- It's just a large Dense network with many inputs and many outputs. The inputs are arranged to interact with each other on the basis of their temporal sequencing in the data.
- In practice we cannot use SimpleRNNs to achieve meaningful memory (vanishing gradients arise quickly).



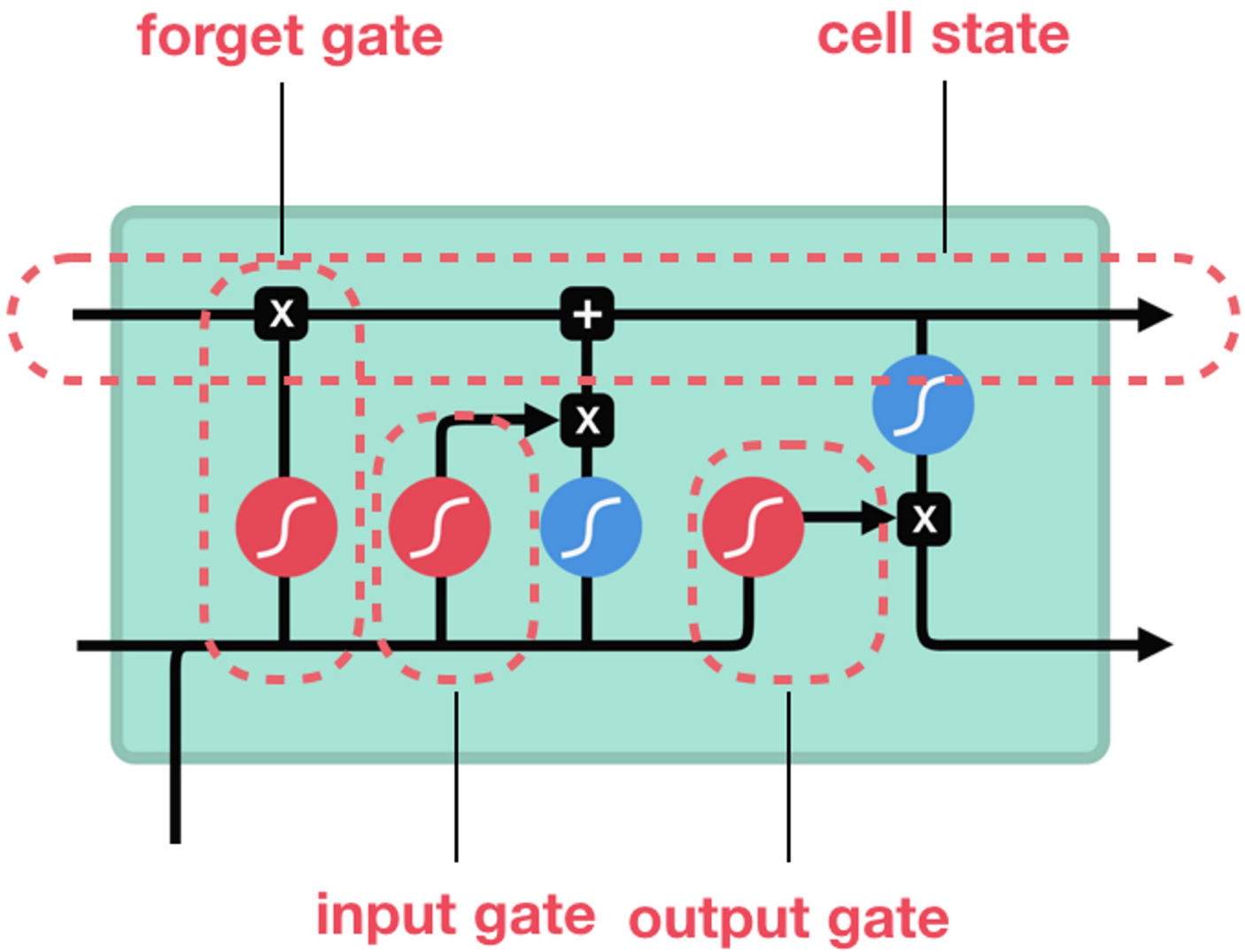
Keras RNN Layers

Long Short-Term Memory (LSTM)

- We add what's called a "carry track" – this is an additional connection that combines output at step t, inputs at step t, and the last carry track's output. The topology means your network can 'learn' to use these connections as passthroughs for old info, or it can learn to 'block' that information in favor of more recent information (whatever is useful for accurate prediction).
- Carry tracks are basically just another degree of freedom for learning how to use lagged information.
- These ideas inspired the design of LSTM units, but nothing guarantees that these gates serve these functions…



LSTM



Keras RNN Layers

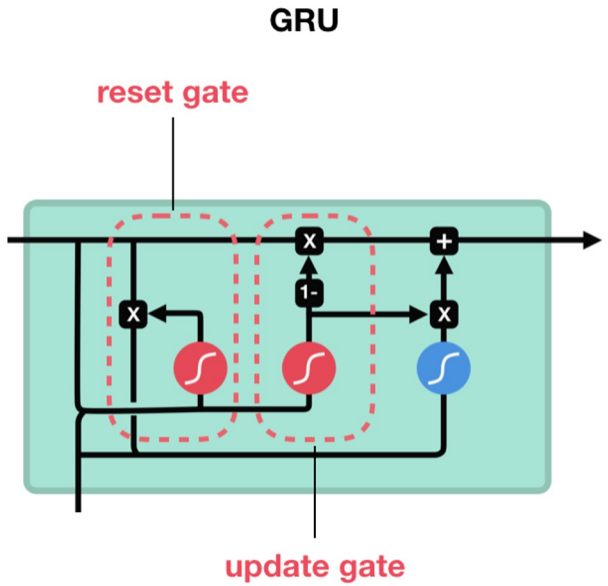
Long Short-Term Memory (LSTM)

```
>>> inputs = tf.random.normal([32, 10, 8])
>>> lstm = tf.keras.layers.LSTM(4)
>>> output = lstm(inputs)
>>> print(output.shape)
(32, 4)
>>> lstm = tf.keras.layers.LSTM(4, return_sequences=True, return_state=True)
>>> whole_seq_output, final_memory_state, final_carry_state = lstm(inputs)
>>> print(whole_seq_output.shape)
(32, 10, 4)
>>> print(final_memory_state.shape)
(32, 4)
>>> print(final_carry_state.shape)
(32, 4)
```

Keras RNN Layers

Gated Recurrent Unit (GRU)

- Less complex than an LSTM. Combines elements of the LSTM into simpler gated structure.
- Fits more quickly, with less data, but memory tends to be shorter.
- More common in NLP tasks, e.g., because “within sentence” or “within paragraph” memory is often sufficient.



Fighting Overfitting in RNNs

Recurrent Dropout

- We can apply dropout in a fixed fashion to all the recurrent steps within an RNN layer.
- The recurrent_dropout argument achieves this (ensures we are applying it homogenously at each time step).
- The dropout argument applies dropout to the inputs entering the RNN layer (like the Dropout you've seen previously).

Listing 10.22 Training and evaluating a dropout-regularized LSTM

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(32, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=50,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

Stacking RNN Layers

Stacked Layers (LSTM or GRUs)

- Same as other 'layer' types, but we need to pass the entire sequence of outputs (not just the last output). We do this with the 'return_sequences=True' argument to the layer.

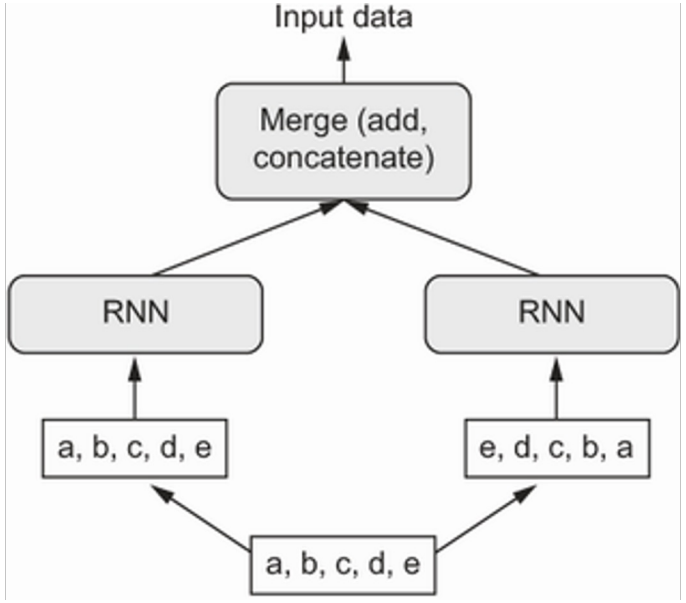
Listing 10.23 Training and evaluating a dropout-regularized, stacked GRU model

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

Bidirectional RNNs

Some Sequences Yield Information in Both Directions

- Consider that, in language, words that come later in a sentence can be predictive of what came before.
- Bidirectional RNNs implement a standard RNN, but they also incorporate a parallel layer implementation that takes the sequence ordered in reverse.



Advanced Scenarios

Forecasting Multiple Series in Parallel

- Sometimes we need to implement forecasts for a panel of units (e.g., revenue of different stores in a chain, or demand for transit at different transit stops).
- We can implement such multivariate timeseries forecasting in Keras a well. Choices need to be made about topology and how to handle the different series'.
- Sometimes (often) it can be better to train separate forecasting models for each series.

Multi-Step Forecasting

- We may want to forecast over a range of future values. These different horizons can be setup as different labels. The model will optimize jointly over the different labels.
- You can also train different models for different horizons.

Questions?