Deep learning for time series

Rodrigo Gonzalez, PhD

Hello!

I am Rodrigo Gonzalez, PhD

You can find me at
rodrigo.gonzalez@ingenieria.uncuyo.edu.ar

Summary

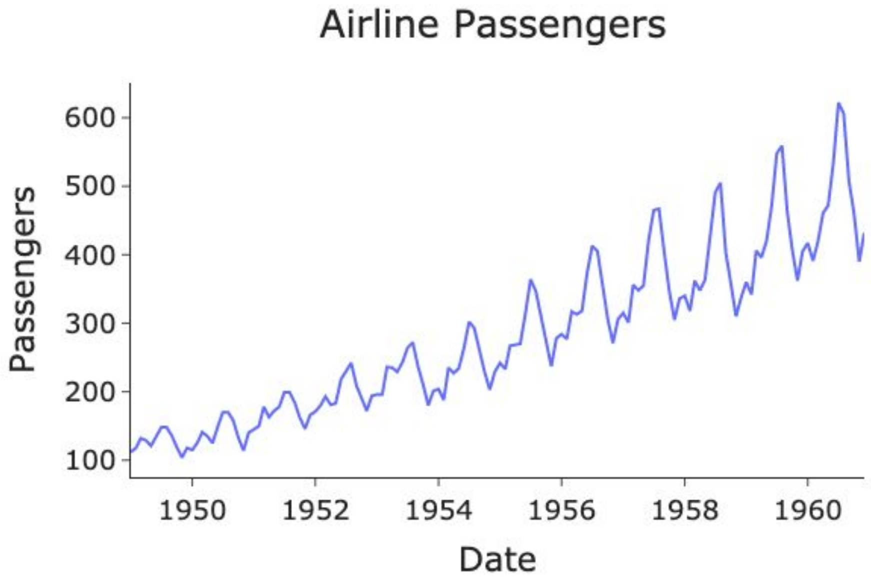
1. Examples of machine learning tasks that involve time-series data
2. Understanding recurrent neural networks (RNNs)
3. Applying RNNs to a temperature-forecasting example
4. Advanced RNN usage

1.

Time series

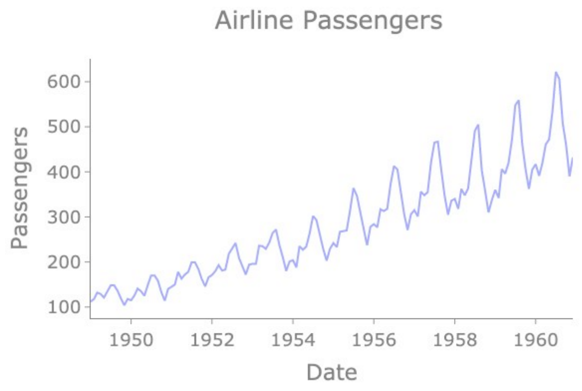
Time series

A time series can be any data obtained via measurements at r**egular intervals**, like the daily price of a stock, the hourly electricity consumption of a city, or the weekly sales of a store.



Time series properties

1. **Period**: time steps at which the series is observed;
2. **Frequency**: Frequency at which the series is observed;
3. **Trend**: long-term change in the mean of the data;
4. **Stationarity**: When time series properties remain constant over time;
5. **Regularity**: Whether the series is captured at regular intervals;
6. **Seasonality**: regular and predictable changes;
7. **Autocorrelation**: Correlation with past observations;



Time series tasks

1. By far, the most common time-series-related task is **forecasting**.
2. But there's actually a wide range of other things you can do with time series:
    a. **Classification**. Assign one or more categorical labels to a time series. For instance, given the time series of the activity of a visitor on a website,
    b. **Event detection**. Identify the occurrence of a specific expected event within a continuous data stream. Hotword detection: "OK, Google" or "Hey, Alexa."
    c. **Anomaly detection**. Detect anything unusual happening within a continuous datastream.

2.

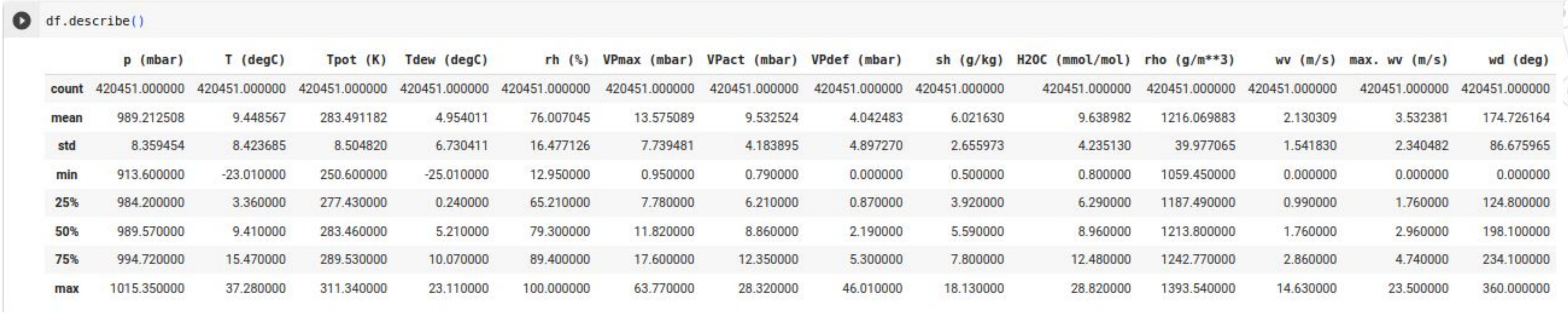
A temperature forecasting example

Forecasting challenge

- Predicting the temperature **24 hours** in the future.
- Dataset recorded for 8 years at the weather station at the Max Planck Institute for Biogeochemistry in **Jena**, Germany 14 different quantities (such as temperature, pressure, humidity, and wind direction) were recorded **every 10 minutes** over several years.

1.

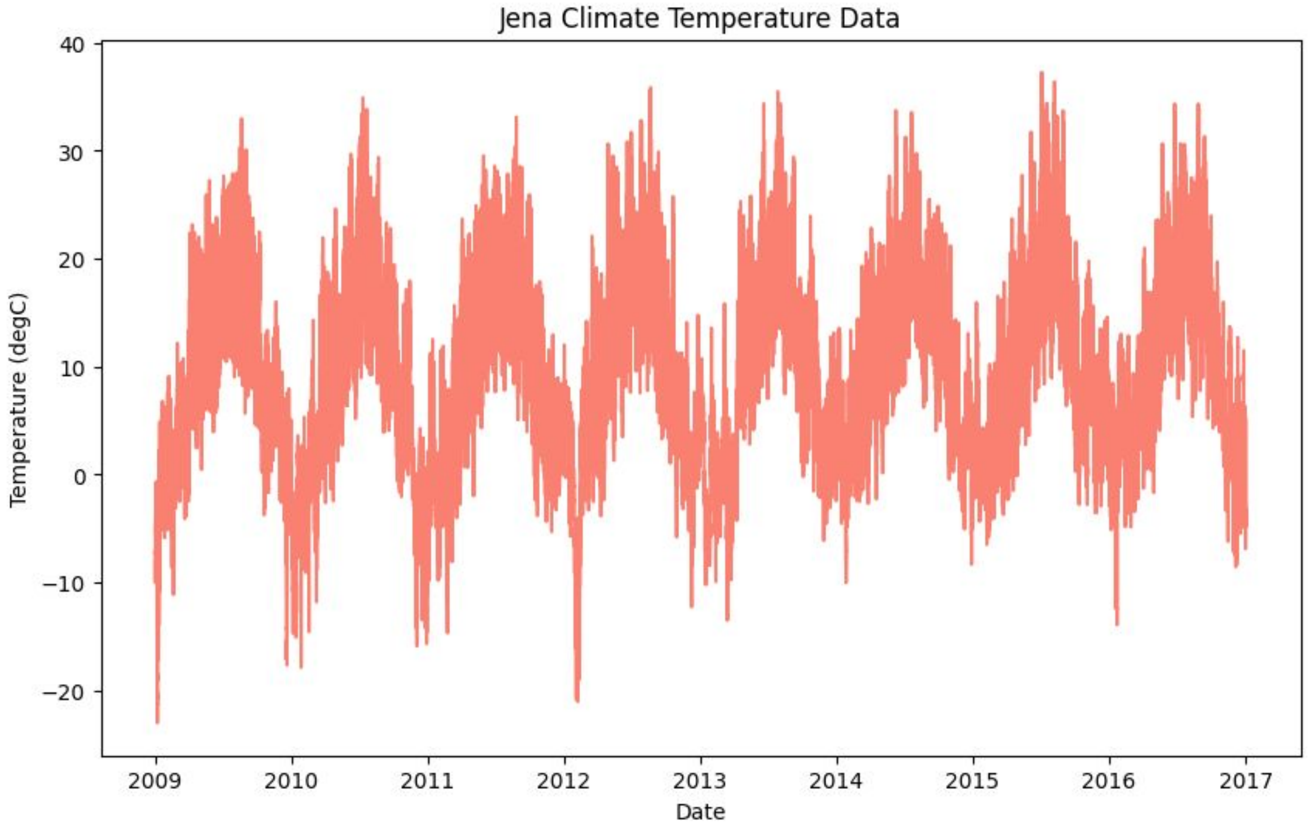
Jena Dataset



```
df.describe()
```

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
count	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000	420451.000000
mean	989.212508	9.448567	283.491182	4.954011	76.007045	13.575089	9.532524	4.042483	6.021630	9.638982	1216.069883	2.130309	3.532381	174.726164
std	8.359454	8.423685	8.504820	6.730411	16.477126	7.739481	4.183895	4.897270	2.655973	4.235130	39.977065	1.541830	2.340482	86.675965
min	913.600000	-23.010000	250.600000	-25.010000	12.950000	0.950000	0.790000	0.000000	0.500000	0.800000	1059.450000	0.000000	0.000000	0.000000
25%	984.200000	3.360000	277.430000	0.240000	65.210000	7.780000	6.210000	0.870000	3.920000	6.290000	1187.490000	0.990000	1.760000	124.800000
50%	989.570000	9.410000	283.460000	5.210000	79.300000	11.820000	8.860000	2.190000	5.590000	8.960000	1213.800000	1.760000	2.960000	198.100000
75%	994.720000	15.470000	289.530000	10.070000	89.400000	17.600000	12.350000	5.300000	7.800000	12.480000	1242.770000	2.860000	4.740000	234.100000
max	1015.350000	37.280000	311.340000	23.110000	100.000000	63.770000	28.320000	46.010000	18.130000	28.820000	1393.540000	14.630000	23.500000	360.000000

Temperature



Temperature, detail

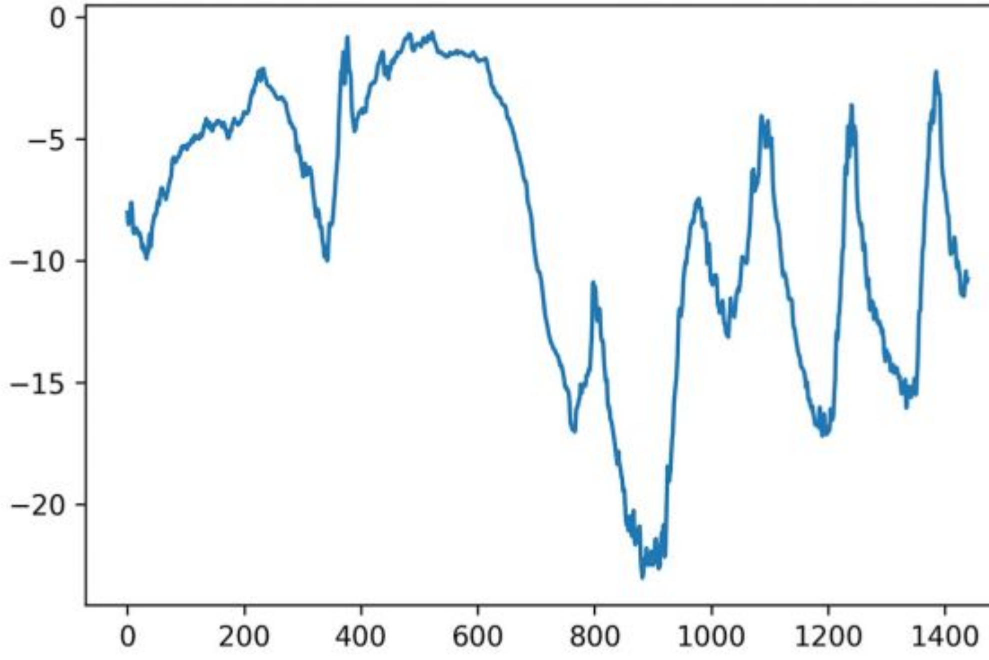


Figure 10.2 Temperature over the first 10 days of the dataset (°C)

3.

Recurrent Neural Networks

A simple RNN

A simple RNN

1. Common neural networks has no memory.
2. No state kept between inputs.
3. As you’re reading the present sentence, you’re processing it word by word while keeping memories of what came before.
4. A RNN keeps a ***state***, internal loop.

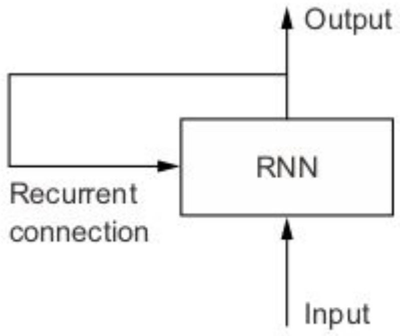
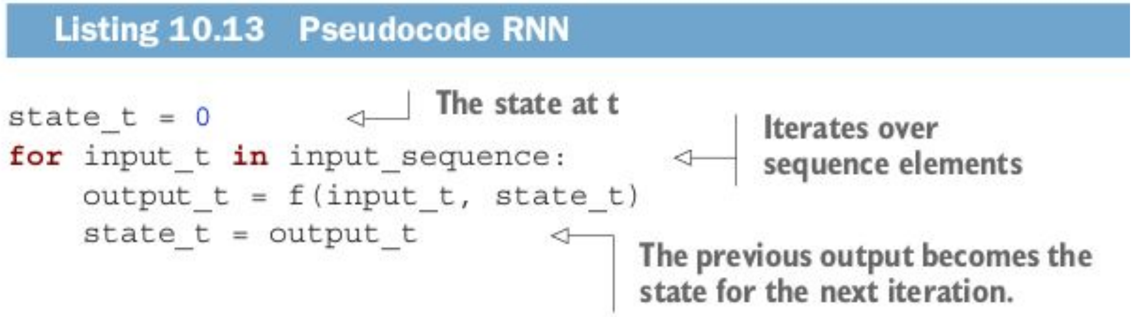


Figure 10.6 A recurrent network: a network with a loop

A simple RNN

1. This RNN takes as input a sequence of vectors which we'll encode as a rank 2 tensor of size (timesteps, input_features).
2. It loops over time steps, and at each time step, it considers its **current state** at *t* and the input a *t* (of shape (input_features)), and combines them to obtain the **output** at *t*.
3. We'll then set the state for the next step to be this previous output.
4. For the first time step, the previous output isn't defined; hence, there is no current state. It's initialized as an all-zero vector called the initial state of the network.



Listing 10.13 Pseudocode RNN

```
state_t = 0
for input_t in input_sequence:
    output_t = f(input_t, state_t)
    state_t = output_t
```

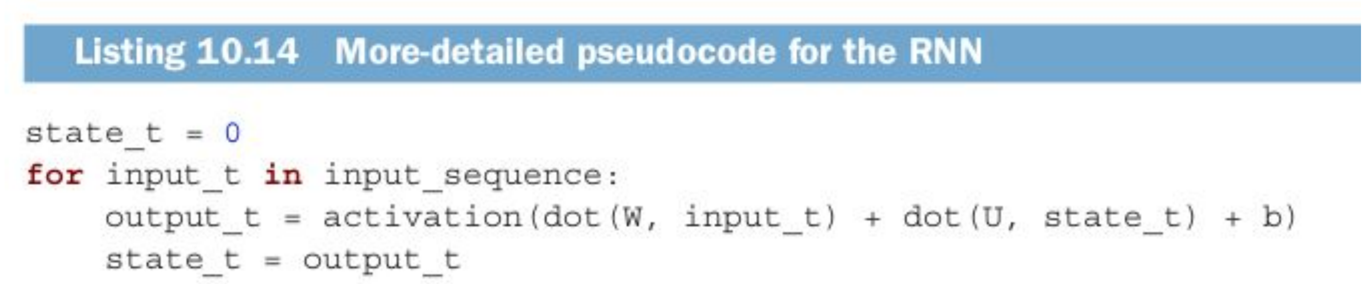
The state at t

Iterates over sequence elements

The previous output becomes the state for the next iteration.

A simple RNN

1. You can even flesh out the function f:
2. The transformation of the input and state into an output will be parameterized by two matrices, **W** and **U**, and a **bias** vector.
3. It’s similar to the transformation operated by a densely connected layer in a feed-forward network.



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
```

A simple RNN

1. In summary, an RNN is a for-loop that reuses quantities computed during the previous iteration of the loop, nothing more.

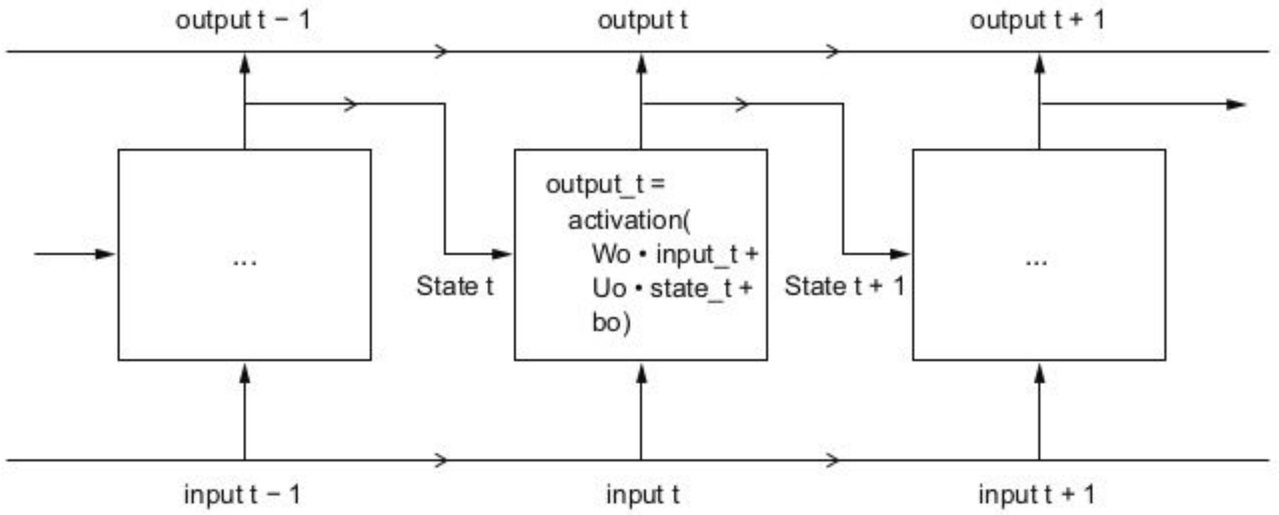
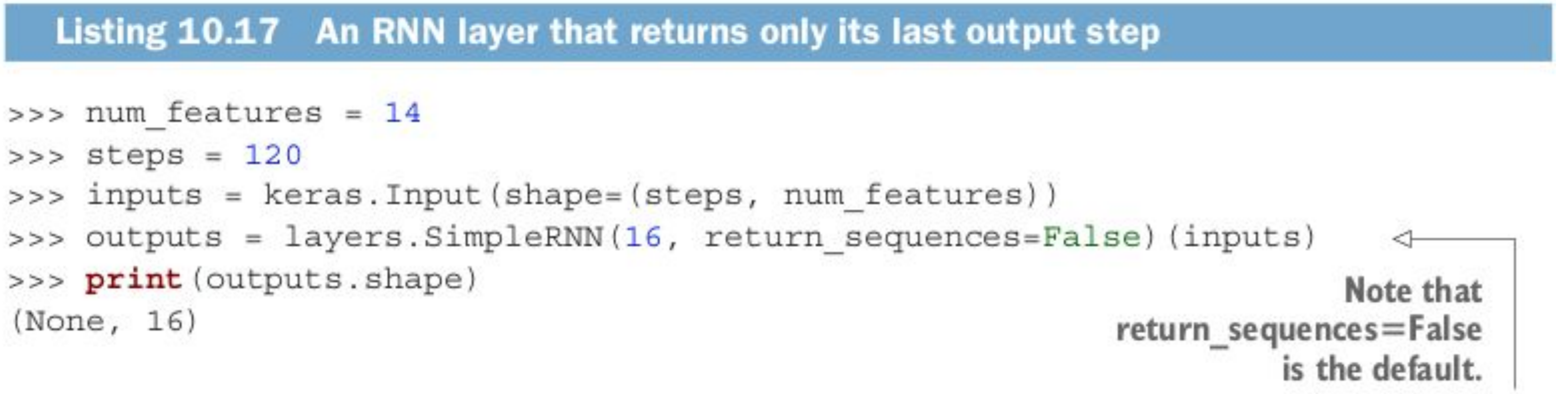


Figure 10.7 A simple RNN, unrolled over time

A simple RNN in Keras

1. In summary, an RNN is a for-loop that reuses quantities computed during the previous iteration of the loop, nothing more.



Listing 10.17 An RNN layer that returns only its last output step

```
>>> num_features = 14
>>> steps = 120
>>> inputs = keras.Input(shape=(steps, num_features))
>>> outputs = layers.SimpleRNN(16, return_sequences=False)(inputs)
>>> print(outputs.shape)
(None, 16)
```

Note that return_sequences=False is the default.

4.

Recurrent Neural Networks

LSTM architecture

LSTM

1. In practice, you’ll rarely work with SimpleRNN() layer.
2. It has a major issue: although it should theoretically be able to retain at time *t* information about inputs seen many time steps before, such long-term dependencies prove impossible to learn in practice. This is due to the **vanishing-gradient problem**,
3. As you keep adding layers to a network, the network eventually becomes untrainable.

LSTM

1. The underlying **long short-term memory** (LSTM) algorithm was developed by Hochreiter and Schmidhuber in 1997.
2. It adds a way to **carry** information across many time steps.
3. It saves information for later, thus preventing older signals from gradually vanishing during processing.
4. This should remind you of **residual connections**, it's pretty much the same idea.
5.

LSTM

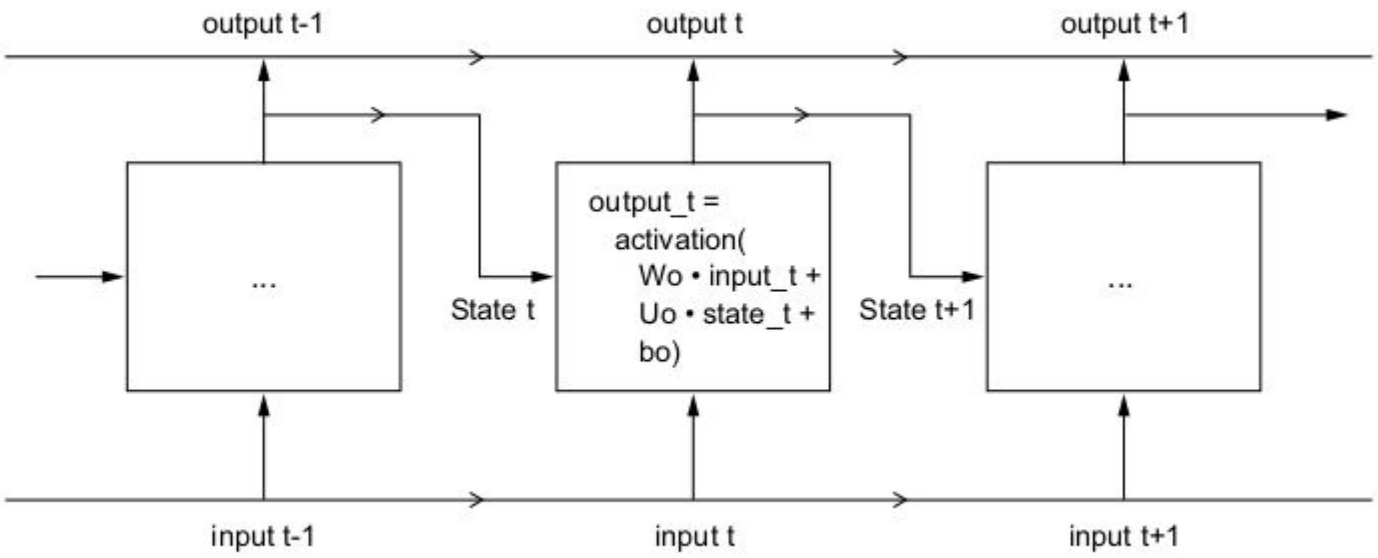


Figure 10.8 The starting point of an `LSTM` layer: a `SimpleRNN`

LSTM

Let’s add to this picture an additional data flow that carries information across time-steps.

Conceptually, the carry dataflow is a way to modulate the next output and the next state

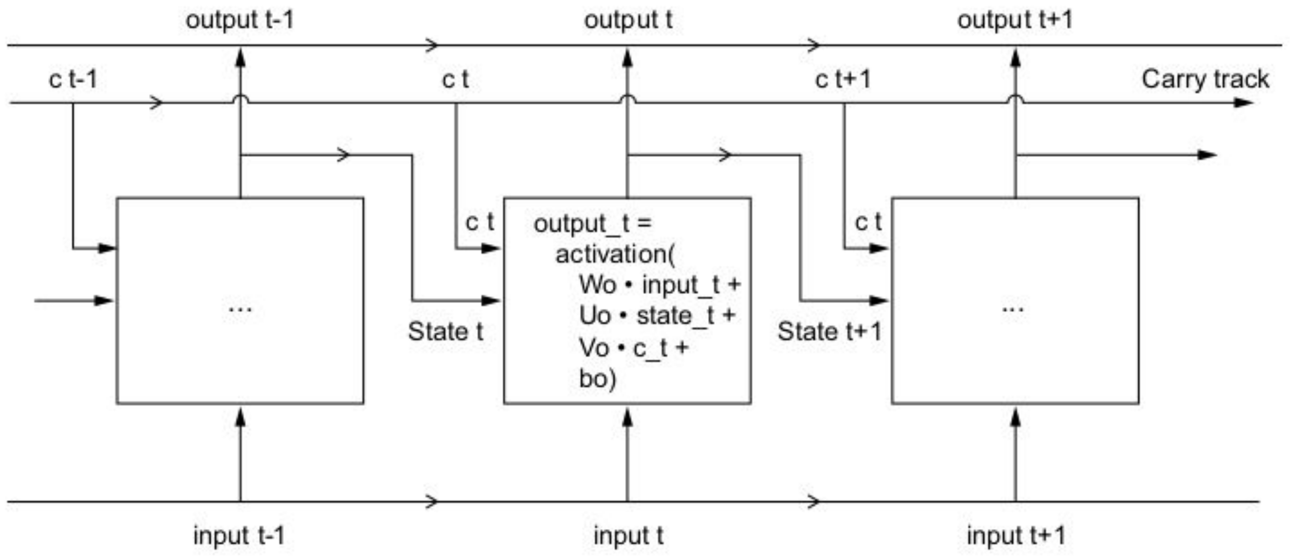
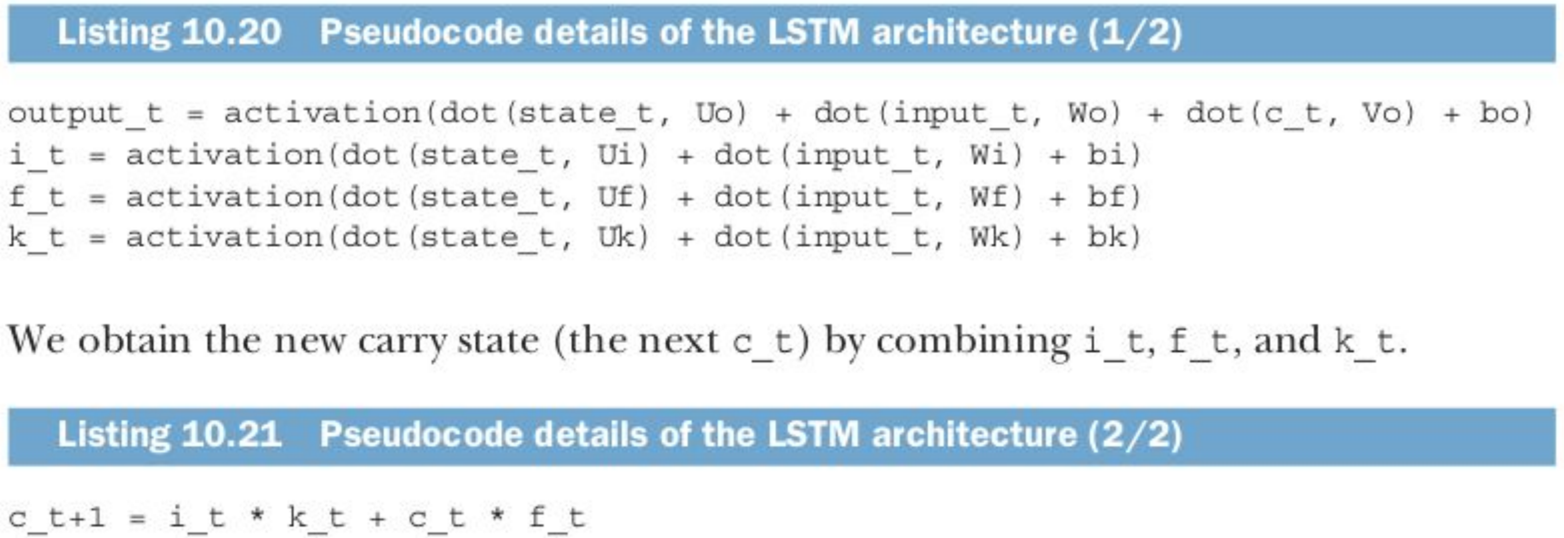


Figure 10.9 Going from a SimpleRNN to an LSTM: adding a carry track

LSTM

How the carry dataflow is computed.



Listing 10.20 Pseudocode details of the LSTM architecture (1/2)

```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(c_t, Vo) + bo)
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```

We obtain the new carry state (the next `c_t`) by combining `i_t`, `f_t`, and `k_t`.

Listing 10.21 Pseudocode details of the LSTM architecture (2/2)

```
c_t+1 = i_t * k_t + c_t * f_t
```

LSTM anatomy

1. The **carry** will be combined with the i**nput connection** and the **recurrent connection** (via a **dense transformation** (weight matrix and bias).
2. It will affect the state being sent to the next time step

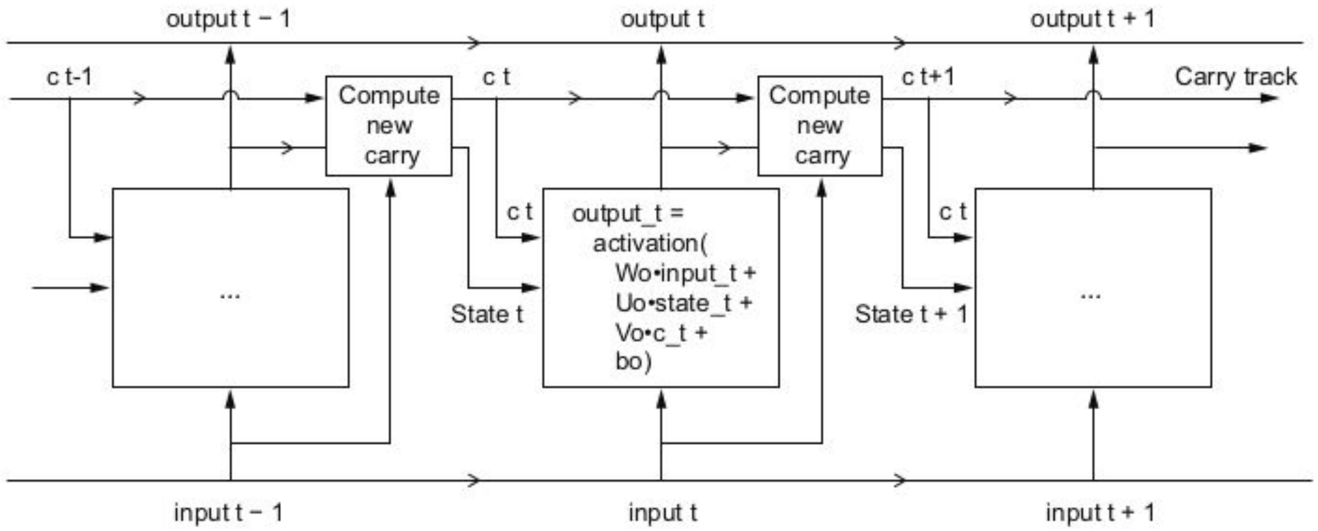


Figure 10.10 Anatomy of an LSTM

LSTM anatomy

Just keep in mind what the LSTM cell is meant to do: allow past information to be re-injected at a later time, thus fighting the vanishing-gradient problem.

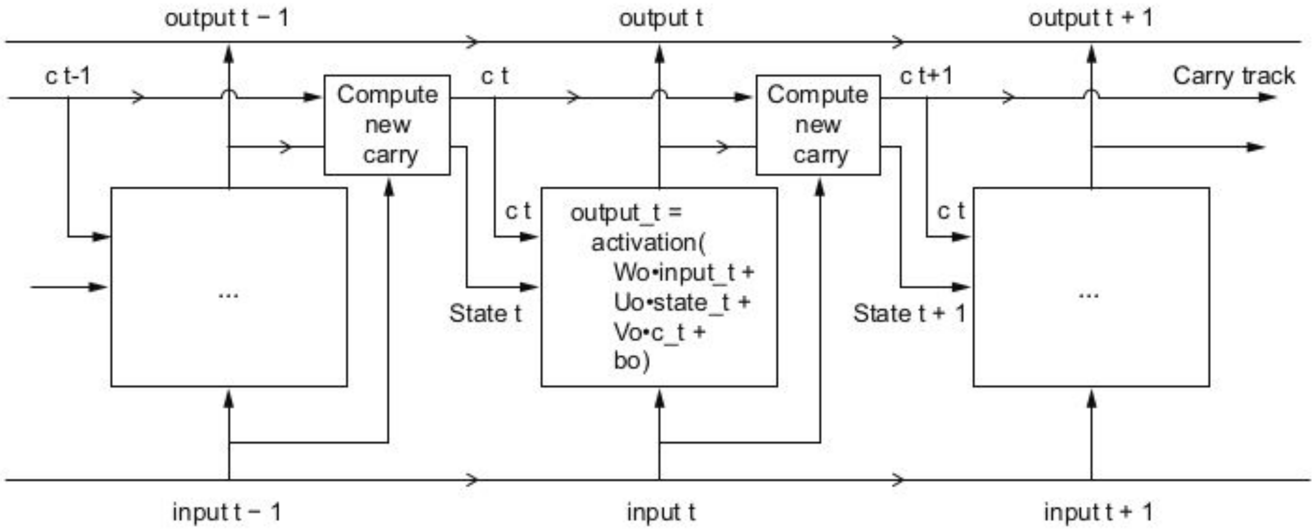


Figure 10.10 **Anatomy of an LSTM**

5.

Advanced use of recurrent neural networks

Recurrent Dropout

If you look at the training and validation curves (figure 10.5), it’s evident that the model is quickly overfitting.

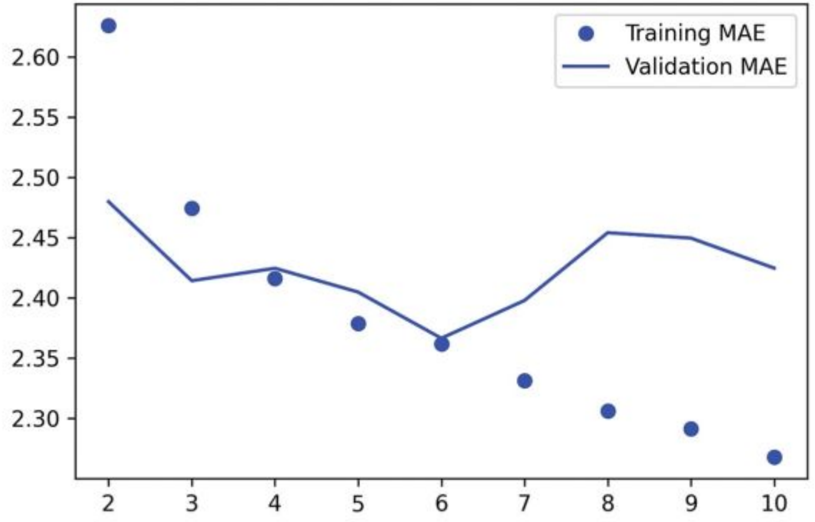


Figure 10.5 Training and validation MAE on the Jena temperature-forecasting task with an LSTM-based model (note that we omit epoch 1 on this graph, because the high training MAE (7.75) at epoch 1 would distort the scale)

Recurrent Dropout

1. **Dropout** randomly zeros out input units of a layer to break happenstance correlations in the training data that the layer is exposed to.
2. But how to correctly apply dropout in recurrent networks **isn't** a **trivial** question.
3. The same dropout mask (the same pattern of dropped units) should be applied at every time step, instead of using a dropout mask that varies randomly from time step to time step.
4. Every recurrent layer in Keras has two dropout-related arguments:
    a. **dropout**, a float specifying the dropout rate for input units of the layer,
    b. **recurrent_dropout**, specifying the dropout rate of the recurrent units.

6.

Temperature forecasting models summary

And the winner is…

1.	Common sense baseline approach:	test MAE of 2.62 degrees Celsius.
2.	Densely connected model:	test MAE of 2.71 degrees Celsius.
3.	A simple LSTM-based model:	test MAE of 2.52 degrees Celsius.
4.	A dropout-regularized LSTM:	test MAE of 2.45 degrees Celsius.

7.

Notebook

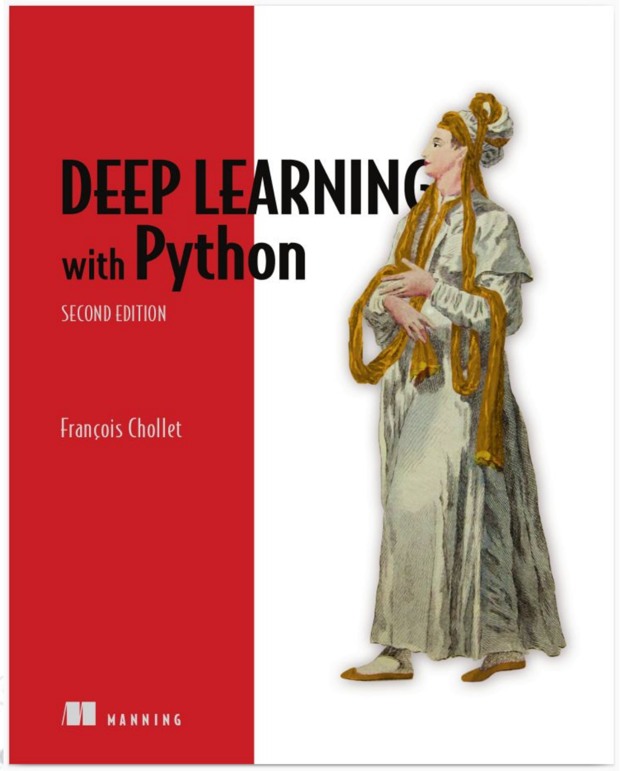
Google Colab notebook

A temperature-forecasting example

https://colab.research.google.com/drive/1gAHC_c3X_RJcKDEF5ItRstgJ2ryxmmd6?usp=shari
ng

8.
Book

Deep Learning with Python, 2nd Ed. by Francois Chollet



- Chapter 10

Thanks!

Any questions?