

# **LSTM in Pytorch with Temporal Data - Lab -**

Matteo Boschini, Lorenzo Bonicelli,  
Angelo Porrello, Emanuele Frascaroli,  
Aniello Panariello

# Agenda

- Jena Weather Dataset
- RNNs & LSTMs
- Build your LSTM-based model
- Train your LSTM-based model

Notebook link: <https://shorturl.at/xAI37>

# Jena Weather Dataset

For today's lab session, we will use an openly available weather data: the Jena Weather Analysis dataset.

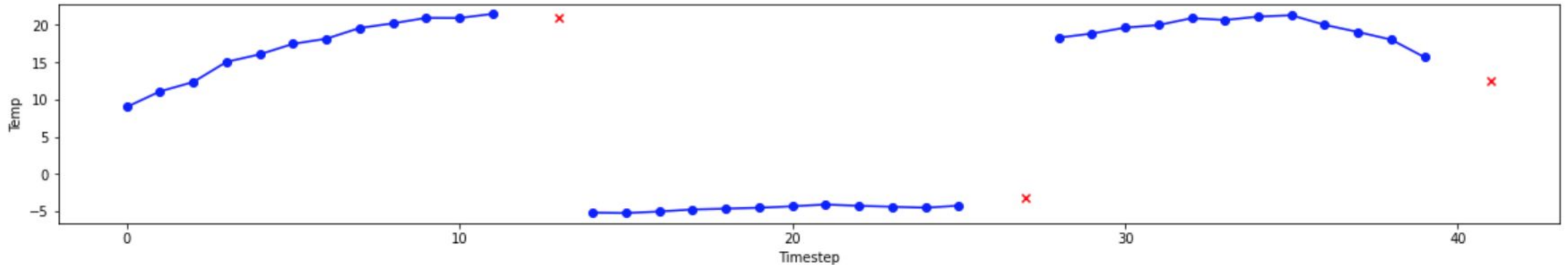
It includes 10-minute recordings of multiple weather parameters for the German city of Jena, including:

- Temperature
- Dew Point
- Relative Humidity
- Vapor Pressure
- Atmospheric Pressure
- Wind Speed
- ...

# Jena Weather Dataset

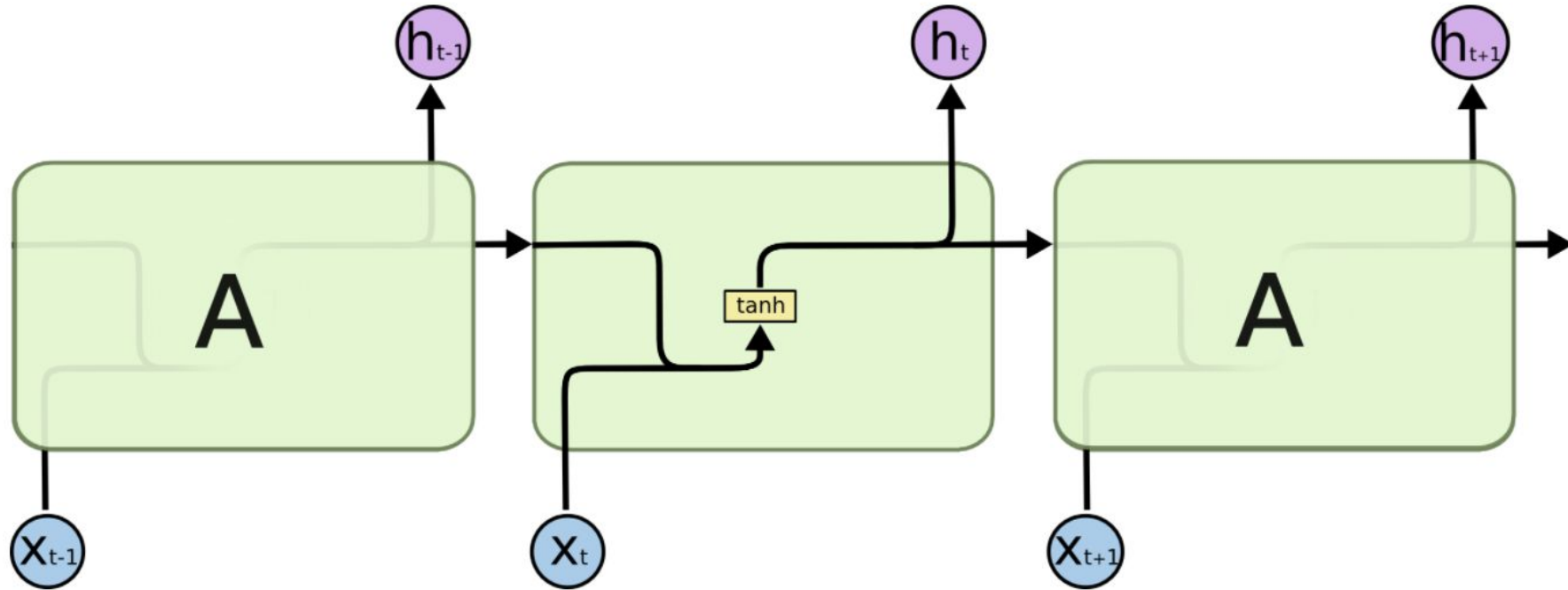
We will design a model that

- Receives a 12 hourly recordings of 19 weather variables
- Forecasts the temperature two hours into the future



# Recurrent Neural Networks

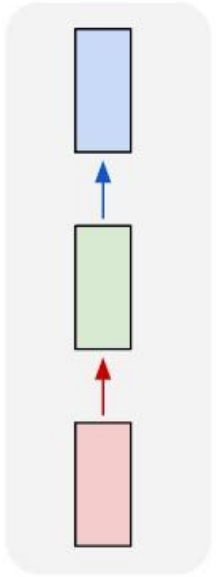
- Handle sequential data in an optimized manner
- Operation is similar to a state-machine
- Information on previous timesteps is encapsulated in the hidden state



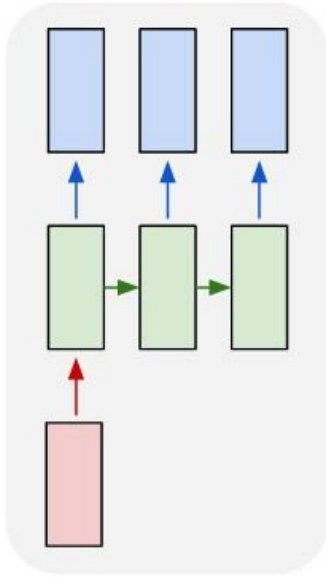
# Modes of operation

- As we deal with sequential data, there are many ways we can use RNNs

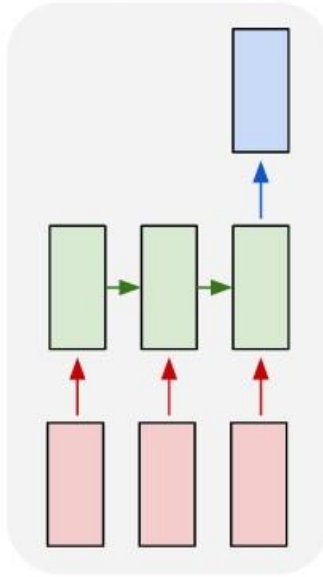
one to one



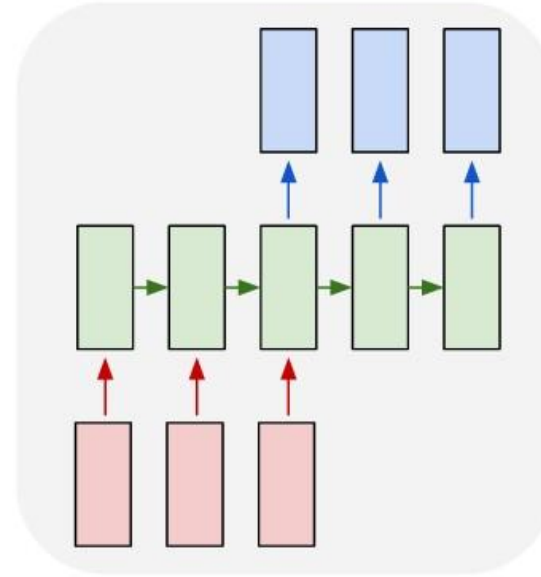
one to many



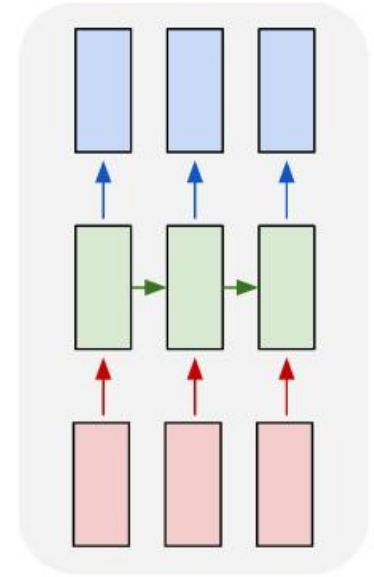
many to one



many to many



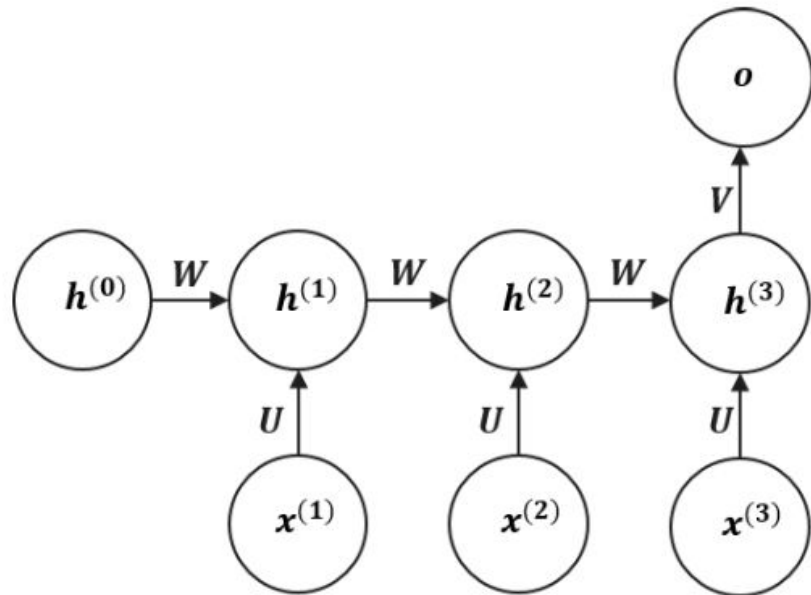
many to many



- What is our configuration for today's task?

# Backpropagation Through Time

- When applying backpropagation, we must *unroll* the computational graph
- Long-term dependencies are problematic (*vanishing gradients*)!

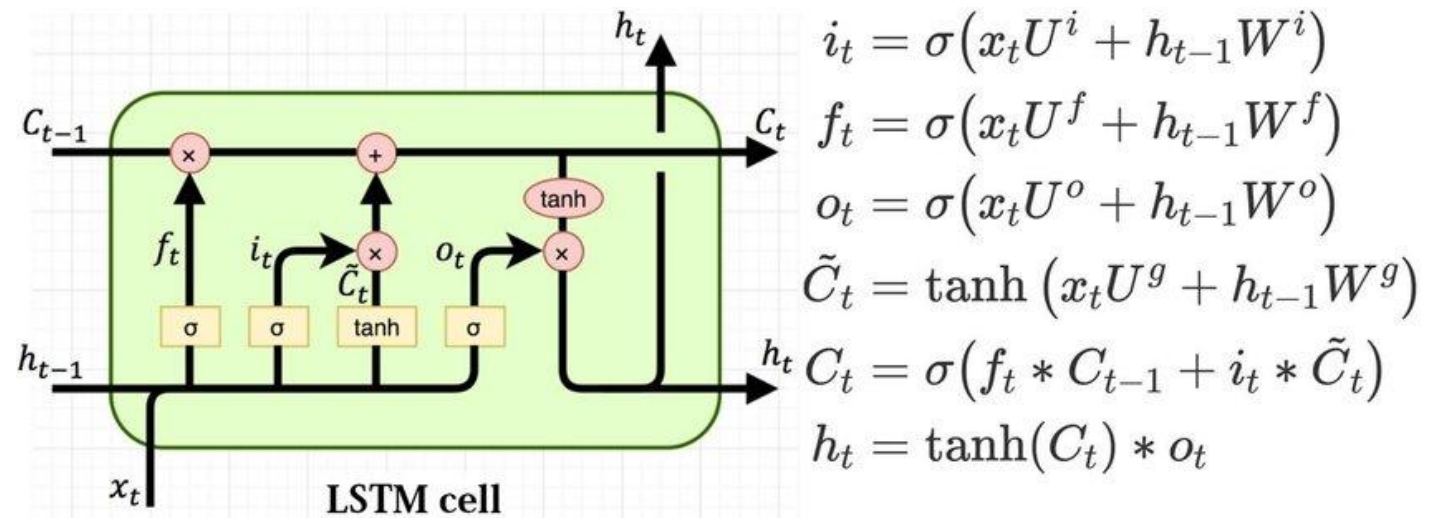
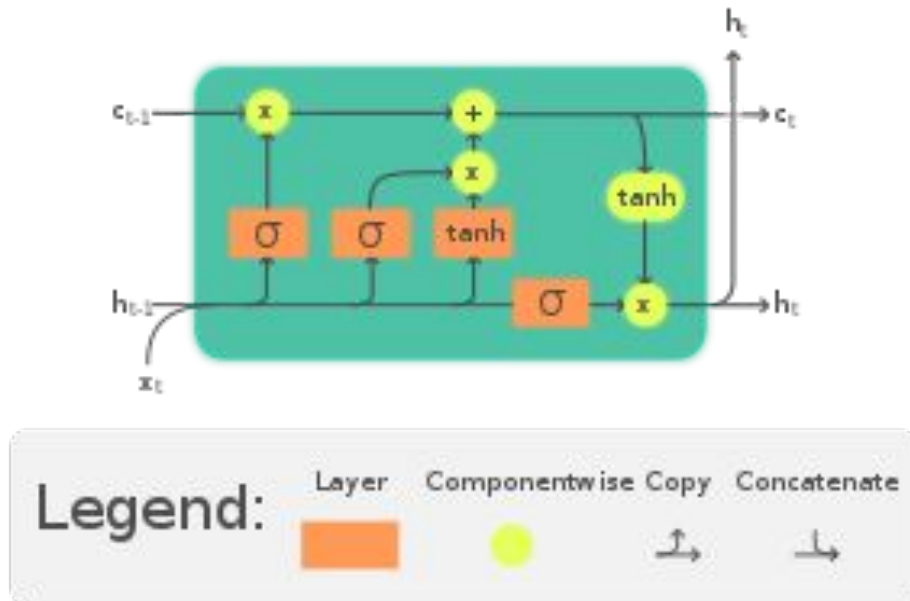


- $\frac{\partial L}{\partial \mathbf{V}} = \frac{\partial L}{\partial \mathbf{o}} \frac{\partial \mathbf{o}}{\partial \mathbf{V}}$
- $\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{o}} \frac{\partial \mathbf{o}}{\partial \mathbf{h}^{(3)}} \sum_{k=0}^3 \left( \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}} \right)$
- $\frac{\partial L}{\partial \mathbf{U}} = \frac{\partial L}{\partial \mathbf{o}} \frac{\partial \mathbf{o}}{\partial \mathbf{h}^{(3)}} \sum_{k=0}^3 \left( \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{U}} \right)$

# Long-Short Term Memory Networks

To compensate for this issue, we introduce LSTMs:

- They facilitate the flow of gradients through their hidden states
- Dual-track cell structure: cell state (*gradient superhighway*) + hidden state (output)
- Forget, input & output gates regulate the information flow from/to cell state
- Sigmoids vs tanhs!





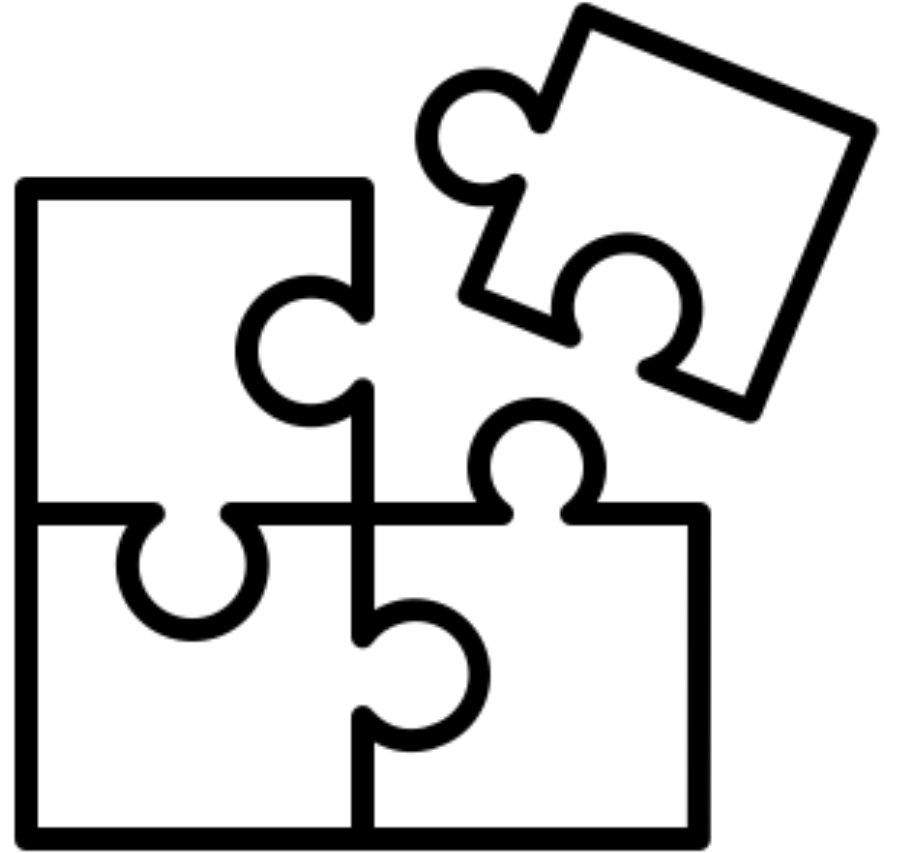
# Defining our model

To build our model, we will need to combine the following classes today:

- `torch.nn.Linear`
- `torch.nn.LSTM`
- `torch.nn.ReLU`

Each implements a piece of the network;  
how can they be combined?

*Follow the shapes!*

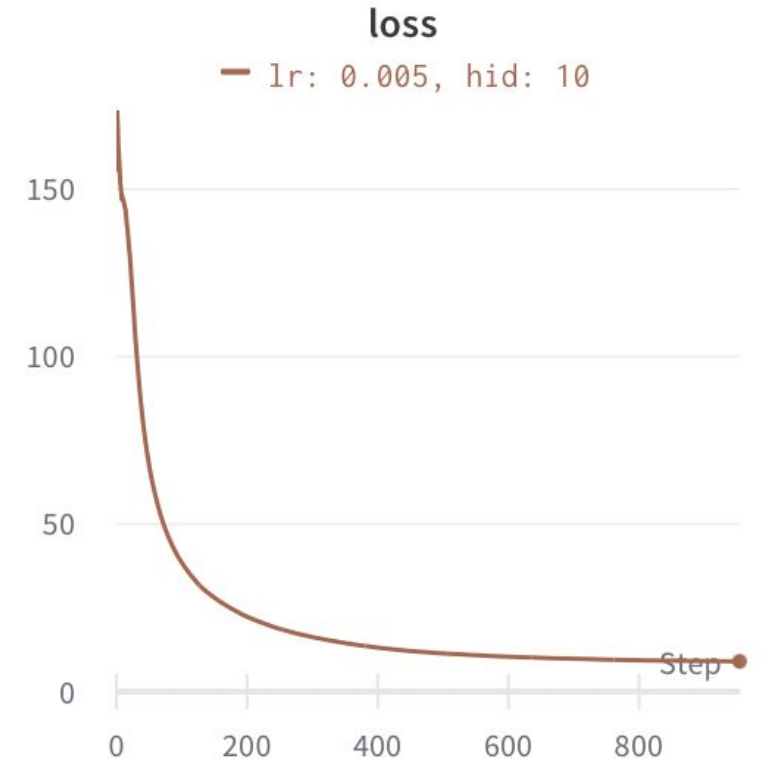


# Training our model

To train our model, we:

- Feed it randomly shuffled sequences of data
- Ask it to guess the corresponding target temperature
- Compute the error (loss)
- Back-propagate the loss through the network
- Adjust parameters accordingly
- goto 1!

*When do we stop?*



# Error Measures

The way we compute errors is a crucial part of the training procedure:

- Cross-Entropy Loss  $\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})}$
- Mean Square Error  $\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2$
- Mean Absolute Error (L1loss)  $\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = |x_n - y_n|$

What can we use for the task at hand? Why?

# Tweaking Hyperparameters

Our model seems to be working, but can we do better?

- Trying out different optimizers
- Testing out different hidden sizes
- Changing the learning rate
- Learning rate scheduling