Time Series

BILICI, M. Şafak
safakk.bilici.2112@gmail.com

UYSAL, Enes S.
enessadi@gmail.com

Contents

1 What Is Time Series?

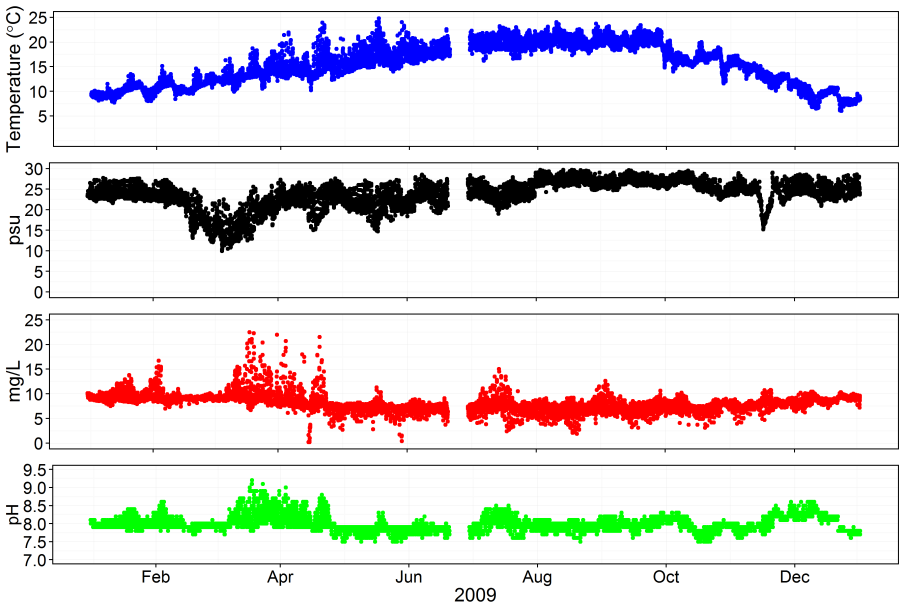

Figure 1: A cool time series figure.

Time series is a series of data points, that are collected at simultaneous intervals or randomly collected. Most of research areas are based on time series: statistics, signal processing, pattern recognition, econometrics, mathematical finance, weather forecasting, earthquake prediction, electroencephalography, control engineering, astronomy, communications engineering, and largely in any domain of applied science and engineering which involves temporal measurements (from Wikipedia).

2 Modeling Time Series

Time series modeling can be seen as a hierarchical function. The data point at time t is dependent on previous k data point. This sequential dependency causes hierarchical modeling. If we denote this f, the objective is can be formulated as

$$\hat{y}_{i,t+1} = f(y_{i,t-k}, y_{i,t-k+1}, ...y_{i,t}, x_{i,t-k}, x_{i,t-k+1}, ...x_{i,t}) \tag{1}$$

We call $y_{i,t-k}, y_{i,t-k+1}, ...y_{i,t}$ as model forecast and $x_{i,t-k}, x_{i,t-k+1}, ...x_{i,t}$ as observations/features [5].

2.1 White Noise

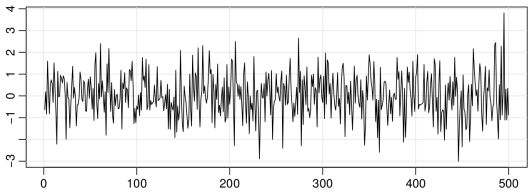

Figure 2: Gaussian White Noise Process.

The time series generated from uncorrelated variables is used as a model for noise, which is called white noise. White noise has mean 0 and finite variance σ^2 [5]. A particularly useful white noise series is Gaussian white noise, formally denoted as

$$w_t \sim \mathcal{N}(0, \sigma^2) \tag{2}$$

Figure 2 show a nice and cool Gauissan white noise [5].

2.2 Moving Average

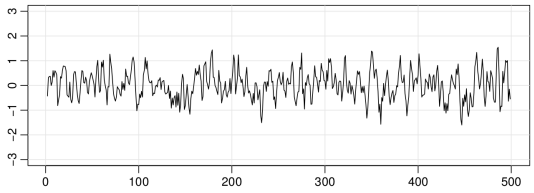

Figure 3: Gaussian White Noise Process.

Sometimes, we want to smooth a time series. For example, we might replace white noise with a smoothed version. Moving average can be used for those purposes. For example, replace a white noise random sample at time t: w_t, replacing its value with neighbors in the past and future:

$$\bar{w}_t = \frac{1}{3}(w_{t-1} + w_t + w_{t+1}) \tag{3}$$

This provides more calm oscillations in white noise. The three-point moving average of Figure 2 is Figure 4 [5].

2.3 Autoregression

Autoregression is a time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next time step [5]. Consider second order equation based on white noise example

$$x_t = x_{t-1} - 0.9 \times x_{t-2} + w_t, \quad t \in [1, 500] \tag{4}$$

This represents a regression of the current value x_t of a time series as a function of the past two values of the series. This is called autoregression [5].

2.4 Random Walk With Drift

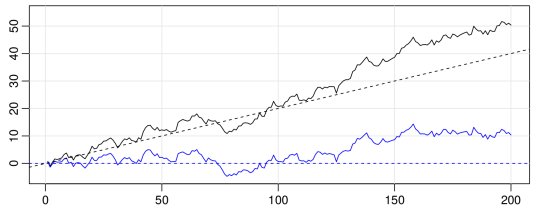

Figure 4: with random white noise $\sigma = 1$; upper jagged line is drift with $\lambda = 0.2$, lower jagged line is drift with $\lambda = 0$.

Random walk with drift model is defined as

$$x_t = \delta + x_{t-1} + w_t, \quad t \in [1, 500], \quad x_0 = 0 \tag{5}$$

The constant δ is called drift. If $\delta = 0$, we call it as random walk [5]. The Equation 5 can be rewritten as a cumulative sum:

$$x_t = \delta \times t + \sum_{j=1}^{t} w_j \tag{6}$$

2.5 Stationary And Non-Stationary

A stationary (time) series is one whose statistical properties such as the mean, variance and autocorrelation are all constant over time. Hence, a non-stationary series is one whose statistical properties change over time. In particular, moments and joint moments are constant. This can be described intuitively in two ways: 1) statistical properties do not change over time 2) sliding windows of the same size have the same distribution.
For example, white noise is stationary, on the other hand, random walk is non-stationary.

3 Deep Learning For Time Series

3.1 Convolutions

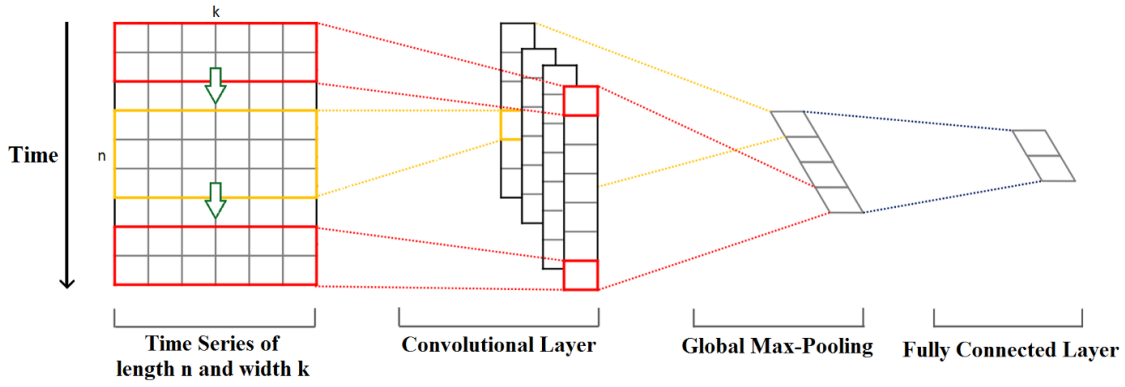

Figure 5: 1D CNN for Time Series.

For an intermediate feature at hidden layer l, each causal convolutional filter takes the form below:

$$h_t^{l+1} = \sigma\Big((W \circledast h)(l,t)\Big) \tag{7}$$

$$(W \circledast h)(l,t) = \sum_{\tau=0}^{k} W(l,r) \cdot h'_{t-\tau} \tag{8}$$

where h'_t is an intermediate state at layer l at time t, $\circledast$ is the convolution operator [3].

3.2 Long Short-Term Memory

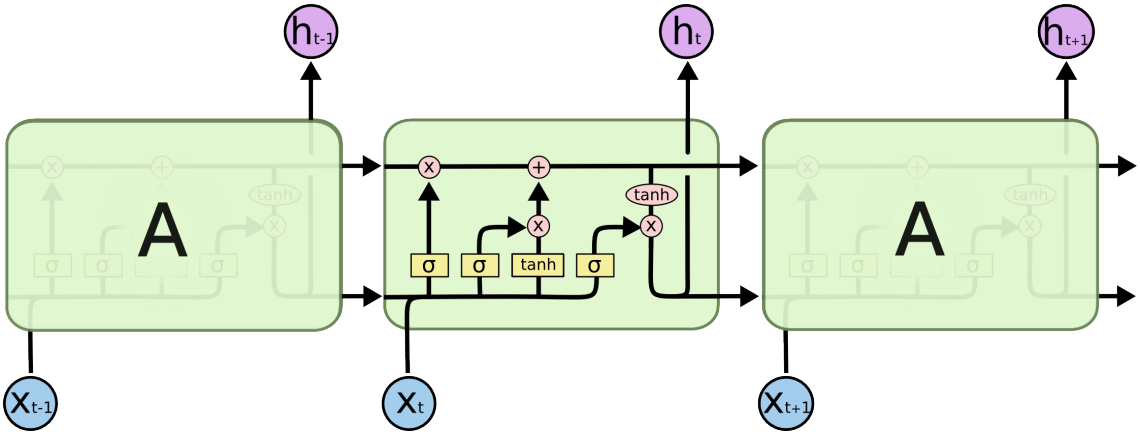

Figure 6: Long short-term memory [4].

Before everything, special thanks to Christopher Olah, due to providing beautiful figures, examples and explanations about LSTMs in detail. Most of the figures and explanations below are directly taken from his blog. You can visit his blog and see other informative blog posts.

LSTMs (Long Short-Term Memory) [2] are fancy version of RNNs (Recurrent Neural Network). Instead of a unit that simply applies an elementwise nonlinearity to the affine transformation of inputs and recurrent units, LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN. Each cell has the same inputs and outputs as an ordinary recurrent network, but has more parameters and a system of gating units that controls the flow of information [1].

3.2.1 Cell State

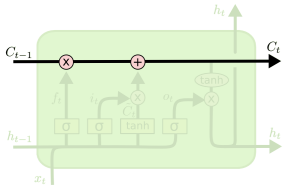

Figure 7: Cell state [4].

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation [4].

3.2.2 Forget Gate

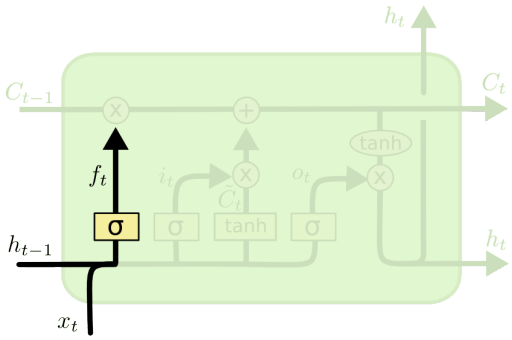

Figure 8: Forget gate [4].

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the forget gate. Let's consider a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject [4].
The formal definition of forget gate can be done:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \tag{9}$$

where $\mathbf{x}^{(t)}$ is current input vector and $\mathbf{h}^{(t)}$ is the current hidden layer vector. $\mathbf{b}^f$ is biases, $\mathbf{U}^f$, $\mathbf{W}^f$ are trainable input and recurrent weights for forget gate.

3.2.3 External Input Gate

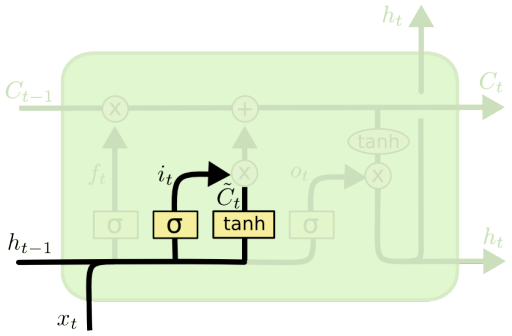

Figure 9: External input gate [4].

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the external input gate decides which values we'll update.

Next, a *tanh* layer creates a vector of new candidate values, $\tilde{\mathbf{C}}^{(t)}$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting [4]. The formal definition of external input gate can be done:

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right) \tag{10}$$

$$\tilde{C}_i^{(t)} = \tanh\left(b_i^C + \sum_j U_{i,j}^C x_j^{(t)} + \sum_j W_{i,j}^C h_j^{(t-1)}\right) \tag{11}$$

3.2.4 Updating Cell State

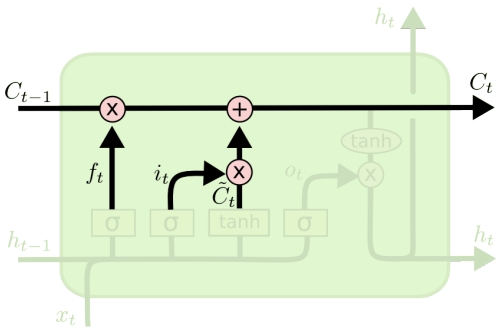

Figure 10: Cell state updating [4].

It's now time to update the old cell state: $\mathbf{C}^{(t-1)} \rightarrow \mathbf{C}^{(t)}$. We multiply the old state by $\mathbf{f}^{(t)}$ forgetting the things we decided to forget earlier. Then we add

$$\mathbf{i}^{(t)} \odot \tilde{\mathbf{C}}^{(t)} \tag{12}$$

$$C_i^{(t)} = f_i^{(t)} C_i^{(t-1)} + g_i^{(t)} \tilde{C}_i^{(t)} \tag{13}$$

This is the new candidate values, scaled by how much we decided to update each state value. In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps [4].

3.2.5 Output Gate

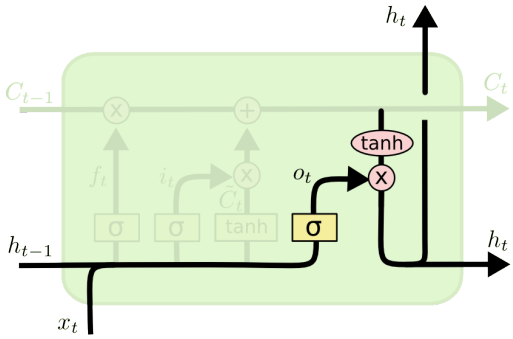

Figure 11: Output gate [4].

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to [4].

$$\mathbf{C}^{(t)} \odot \mathbf{o}^{(t)} \tag{14}$$

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next [4]. The formal definition of output gate can be done:

$$o_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t)}\right) \tag{15}$$

$$h_i^t = \tanh(C_i^{(t)}) o_i^{(t)} \tag{16}$$

3.3 PyTorch's nn.LSTM in Nutshell

3.3.1 One Layer Unidirectional LSTM

```python
import torch
import torch.nn as nn

batch_size = 16
seq_len = 512
input_size = 1

input = torch.randn(batch_size, seq_len, input_size)
print(input.shape)
# torch.Size([16, 512, 1])

lstm = nn.LSTM(
    input_size = input_size,
    hidden_size = 128,
    num_layers = 1,
    bidirectional = False,
    batch_first = True)

output, (hidden, cell) = lstm.forward(input)

print(output.shape)
# torch.Size([16, 512, 128])
print(hidden.shape)
# torch.Size([1, 16, 128])
print(cell.shape)
# torch.Size([1, 16, 128])
```

Figure 12: One Layer Unidirectional LSTM

3.3.2 Multilayer Unidirectional LSTM

```python
import torch
import torch.nn as nn

batch_size = 16
seq_len = 512
input_size = 64

input = torch.randn(batch_size, seq_len, input_size)
print(input.shape)
# torch.Size([16, 512, 64])

lstm = nn.LSTM(
    input_size = input_size,
    hidden_size = 128,
    num_layers = 4,
    bidirectional = False,
    batch_first = True)

output, (hidden, cell) = lstm.forward(input)

print(output.shape)
# torch.Size([16, 512, 128])
print(hidden.shape)
# torch.Size([4, 16, 128])
print(cell.shape)
# torch.Size([4, 16, 128])
```

Figure 13: Multilayer Unidirectional LSTM

3.3.3 One Layer Bidirectional LSTM

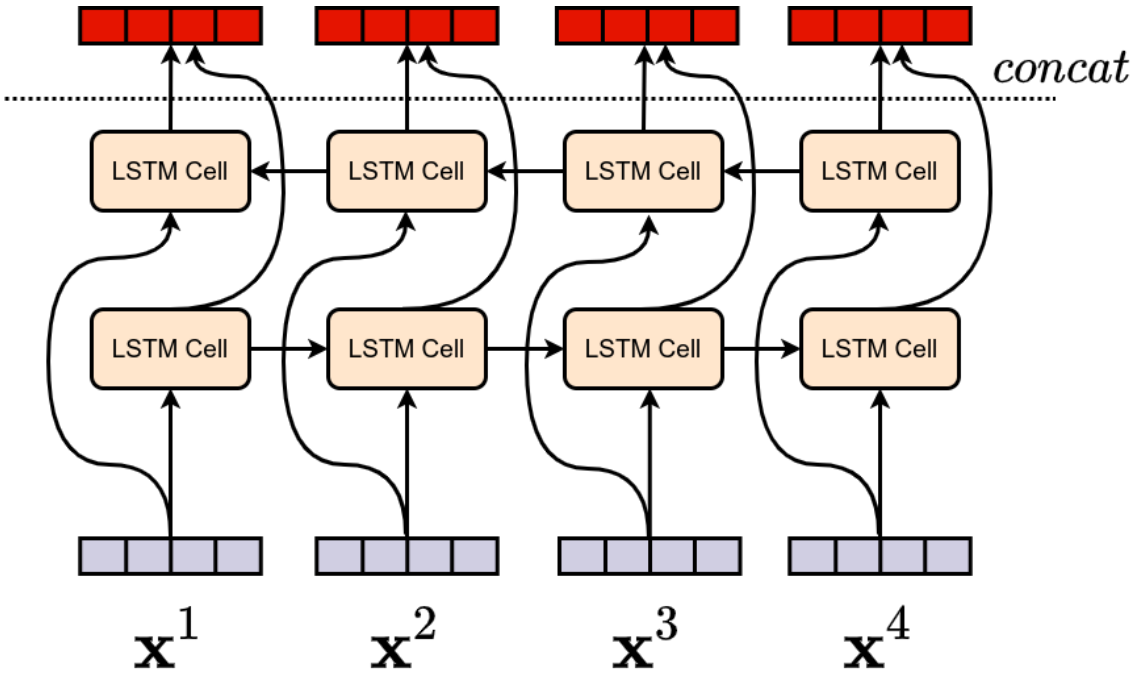

Figure 14: Visual guide for One Layer Bidirectional LSTM.

```python
import torch
import torch.nn as nn

batch_size = 16
seq_len = 512
input_size = 64

input = torch.randn(batch_size, seq_len, input_size)
print(input.shape)
# torch.Size([16, 512, 64])

lstm = nn.LSTM(
    input_size = input_size,
    hidden_size = 128,
    num_layers = 1,
    bidirectional = True,
    batch_first = True)

output, (hidden, cell) = lstm.forward(input)

print(output.shape)
# torch.Size([16, 512, 256])
print(hidden.shape)
# torch.Size([2, 16, 128])
print(cell.shape)
# torch.Size([2, 16, 128])
```

Figure 15: One Layer Bidirectional LSTM

3.3.4 Multilayer Bidirectional LSTM

```python
import torch
import torch.nn as nn

batch_size = 16
seq_len = 512
input_size = 64

input = torch.randn(batch_size, seq_len, input_size)
print(input.shape)
# torch.Size([16, 512, 64])

lstm = nn.LSTM(
    input_size = input_size,
    hidden_size = 128,
    num_layers = 4,
    bidirectional = True,
    batch_first = True)

output, (hidden, cell) = lstm.forward(input)

print(output.shape)
# torch.Size([16, 512, 256])
print(hidden.shape)
# torch.Size([8, 16, 128])
print(cell.shape)
# torch.Size([8, 16, 128])
```

Figure 16: Multilayer Bidirectional LSTM

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[2] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[3] Bryan Lim and Stefan Zohren. "Time-series forecasting with deep learning: a survey". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379.2194 (Feb. 2021), p. 20200209. ISSN: 1471-2962. DOI: 10.1098/rsta.2020.0209. URL: http://dx.doi.org/10.1098/rsta.2020.0209.

[4] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[5] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications (Springer Texts in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2005. ISBN: 0387989501.