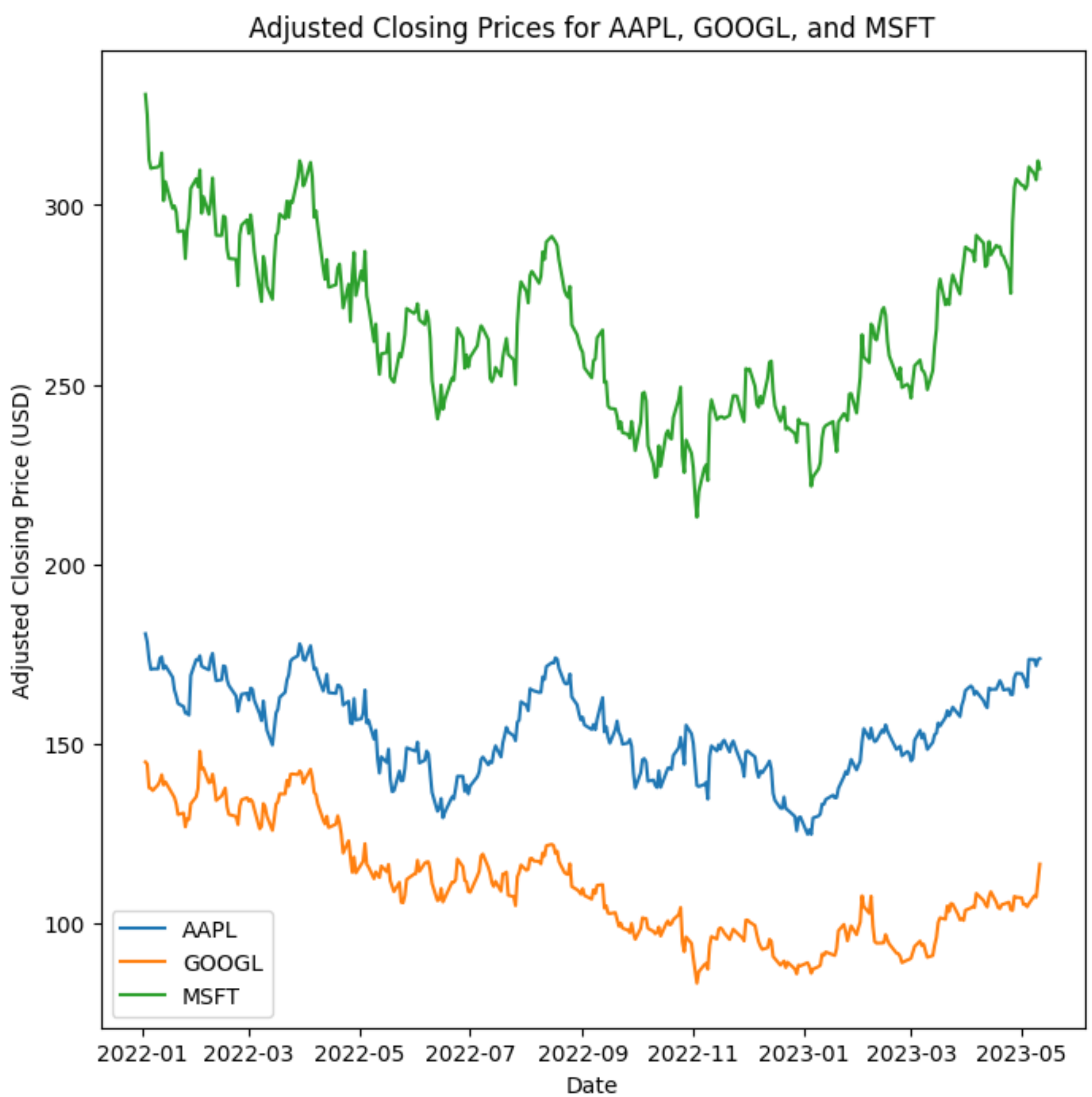
Recurrent Neural Networks and Natural Language Processing

Inar Timiryasov (NBI)

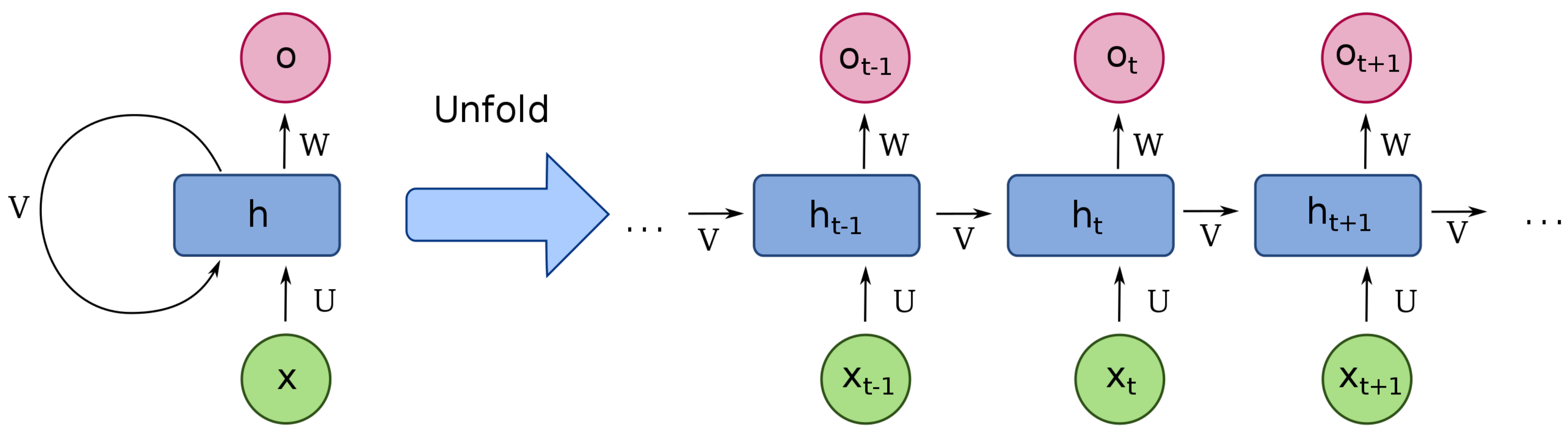
May 17, 2023

Time series data and limitations of the usual network architectures



- Time series data are ubiquitous
- A common task is to predict the next value
- Usual NN approaches are not well suited for these type of data:
    - Lack of temporal dynamics (treating input features independently)
    - Fixed input size
    - CNNs capture only local dependences by design

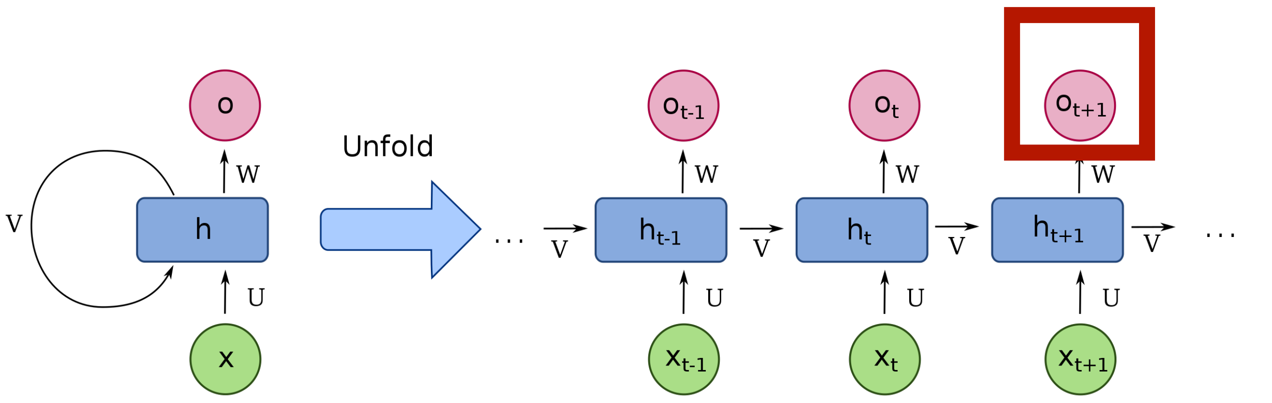
Recurrent Neural Network (RNN)

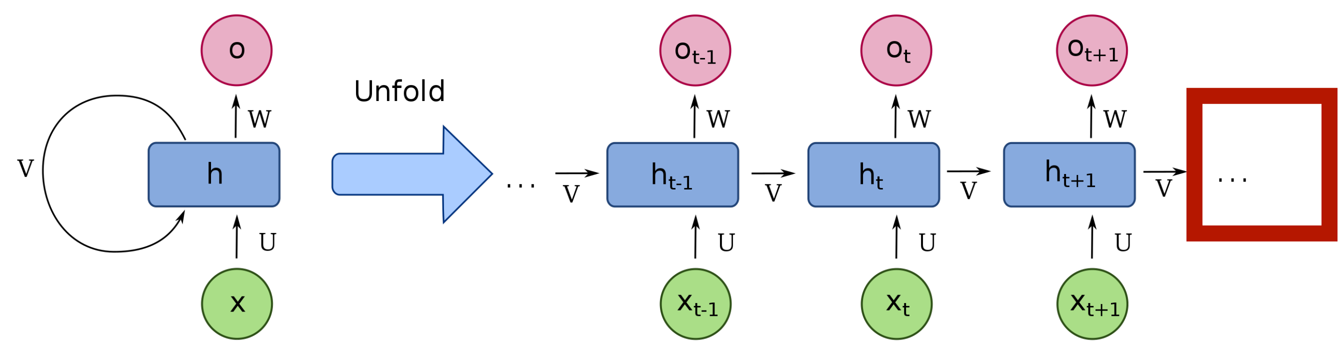


Notice that pytorch RNN only outputs $h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$

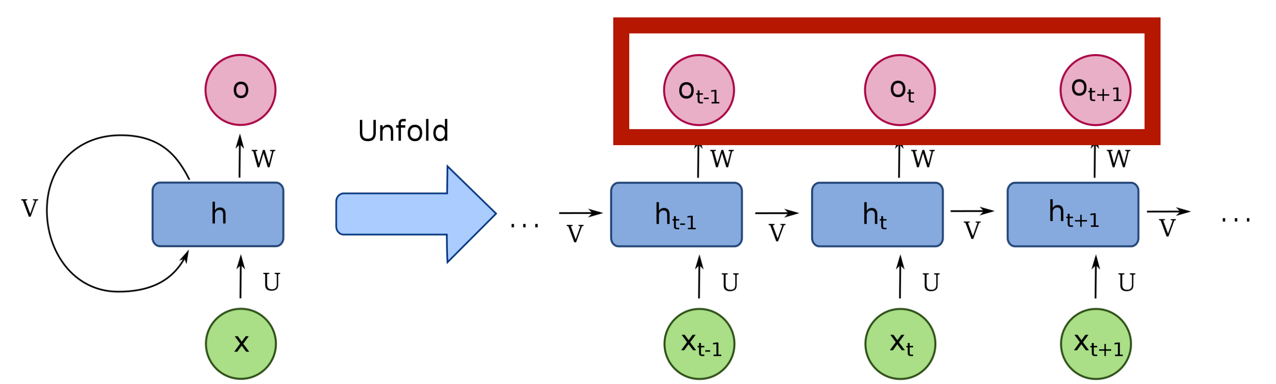
RNN use cases

- Predicting the next value





- Global properties of time series



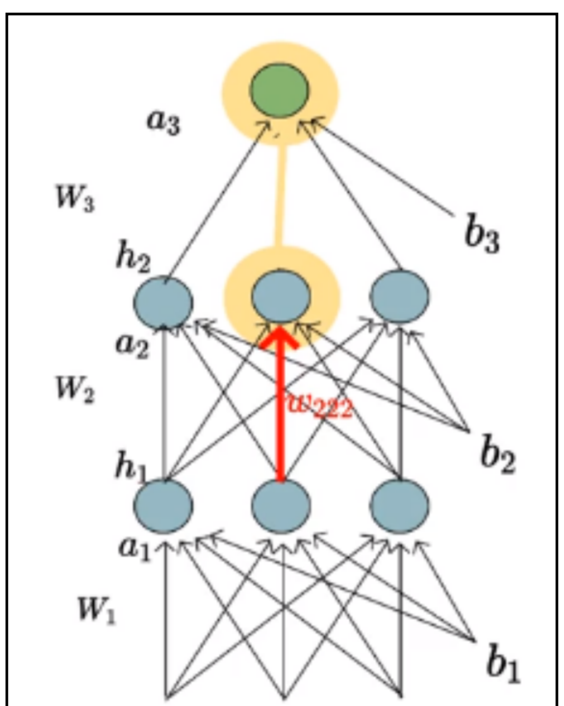
Training RNNs

Recall, that **back propagation** is the modification of NN weights to **minimise the error** of the network output compared to the target values.

The algorithm of **gradient descent** works as follows:

1. Present a training input pattern and use it to get an output.
2. Compare the predicted to the expected outputs and calculate the error.
3. Calculate the derivatives of the error with respect to the network weights.
4. Adjust the weights to minimise the error.

Repeating this should make the **weights converge towards optimal values**.



$$(w_{222})_{t+1} = (w_{222})_t - \eta * \left(\frac{\partial L}{\partial w_{222}}\right)$$

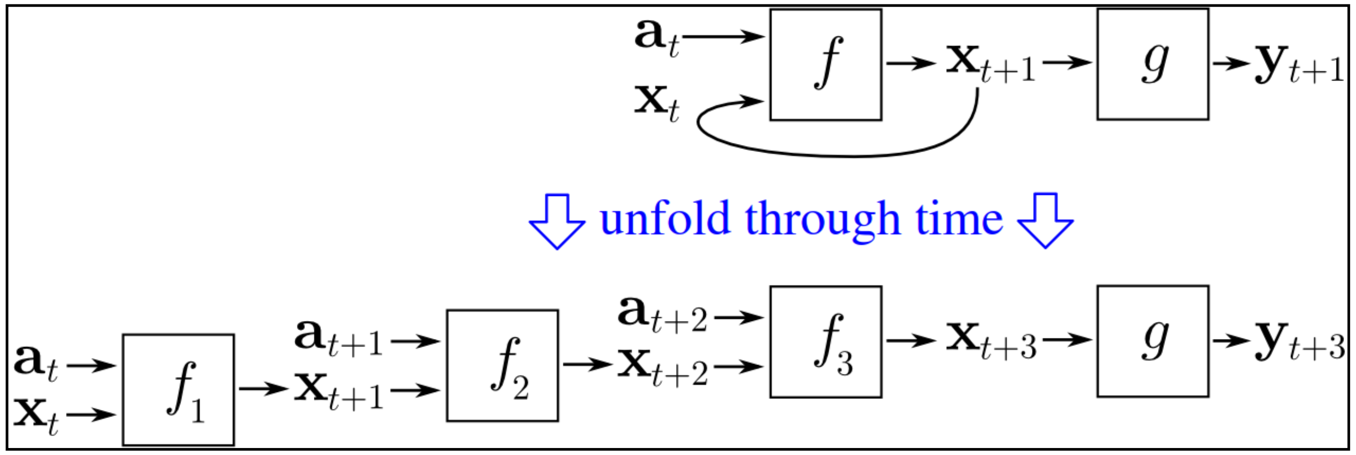
$$\begin{aligned}
\frac{\partial L}{\partial w_{222}} &= \left(\frac{\partial L}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right) \\
&= \left(\frac{\partial L}{\partial h_{22}}\right).\left(\frac{\partial h_{22}}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right) \\
&= \left(\frac{\partial L}{\partial a_{31}}\right).\left(\frac{\partial a_{31}}{\partial h_{22}}\right).\left(\frac{\partial h_{22}}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right) \\
&= \left(\frac{\partial L}{\partial \hat{y}}\right).\left(\frac{\partial \hat{y}}{\partial a_{31}}\right).\left(\frac{\partial a_{31}}{\partial h_{22}}\right).\left(\frac{\partial h_{22}}{\partial a_{22}}\right).\left(\frac{\partial a_{22}}{\partial w_{222}}\right)
\end{aligned}$$

Training RNNs

In **back propagation through time** (for e.g. LSTM), the weights are modified by "unrolling" all input timesteps. Each timestep has one input timestep, one copy of the network, and one output. Errors are then calculated and accumulated for each timestep. The network is rolled back up and the weights are updated.

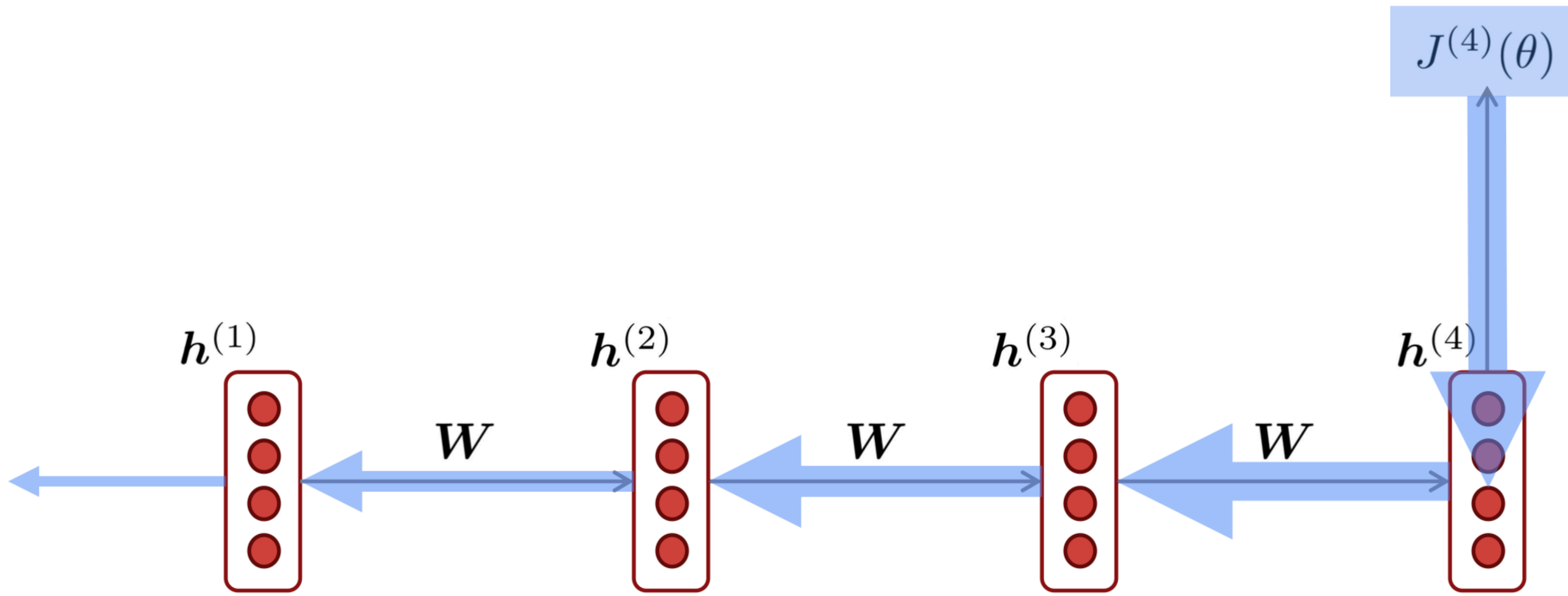
Spatially, each timestep of the unrolled recurrent neural network may be seen as an additional layer. In summary, the BPTT algorithm does as follows:

1. Present a sequence of timesteps of input and output pairs to the network.
2. Unroll the network then calculate and accumulate errors across each timestep.
3. Roll-up the network and update weights.
4. Repeat.

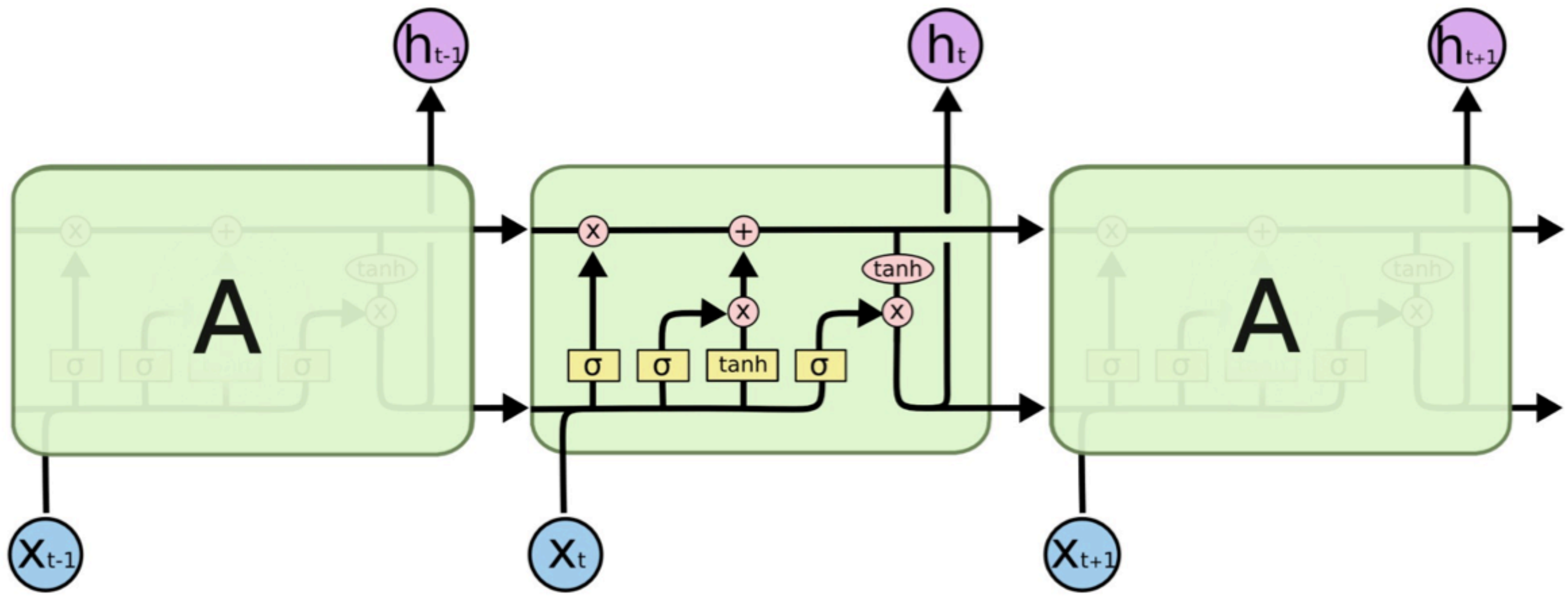


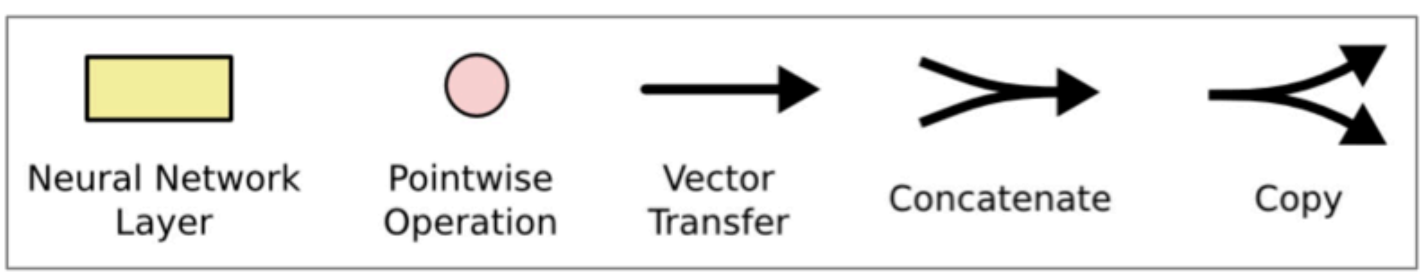
Problem of Long-Term Dependencies

- Vanishing gradients
  vanilla RNNs “forget” early inputs



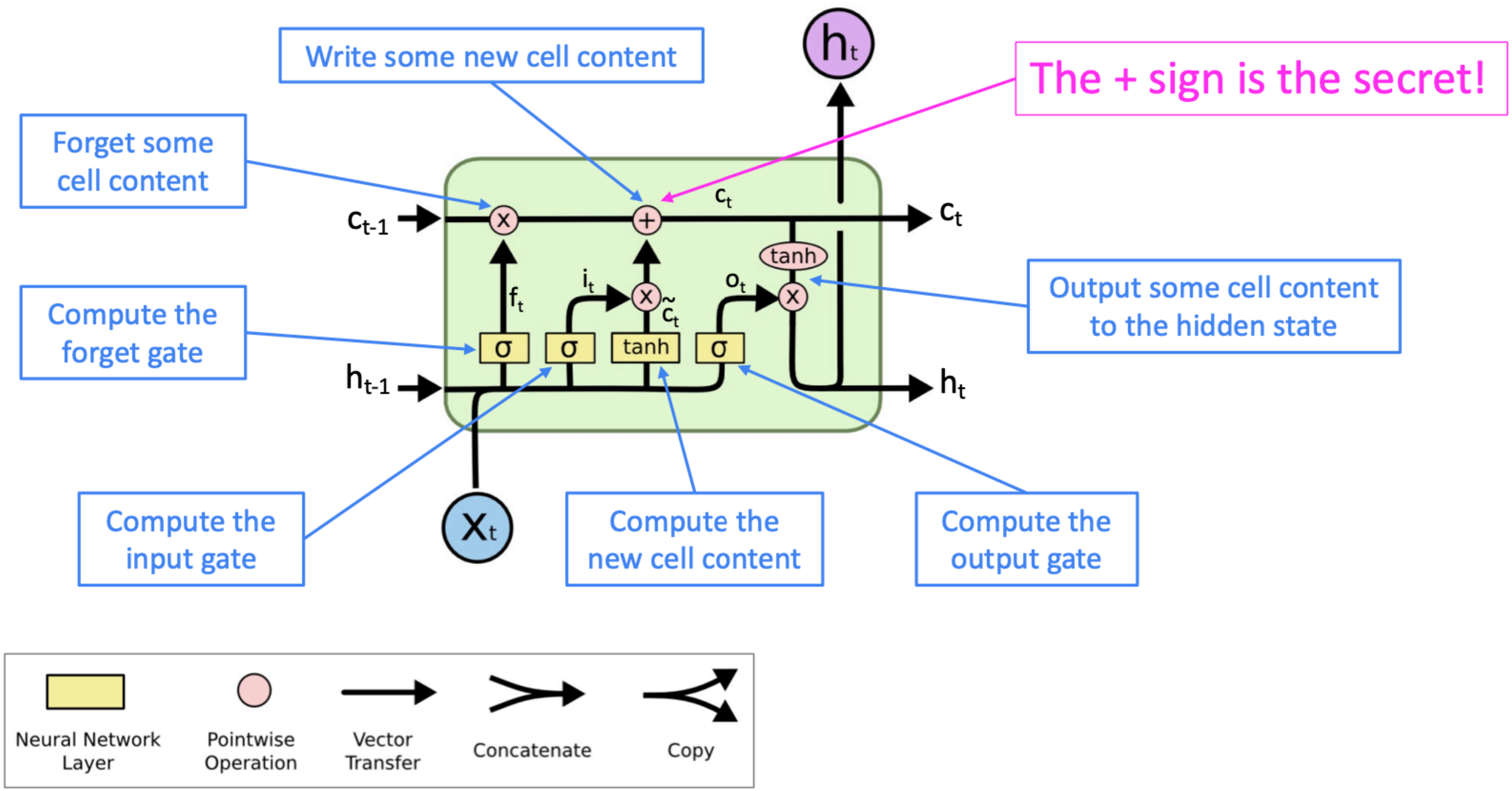
Long Short Term Memory





Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

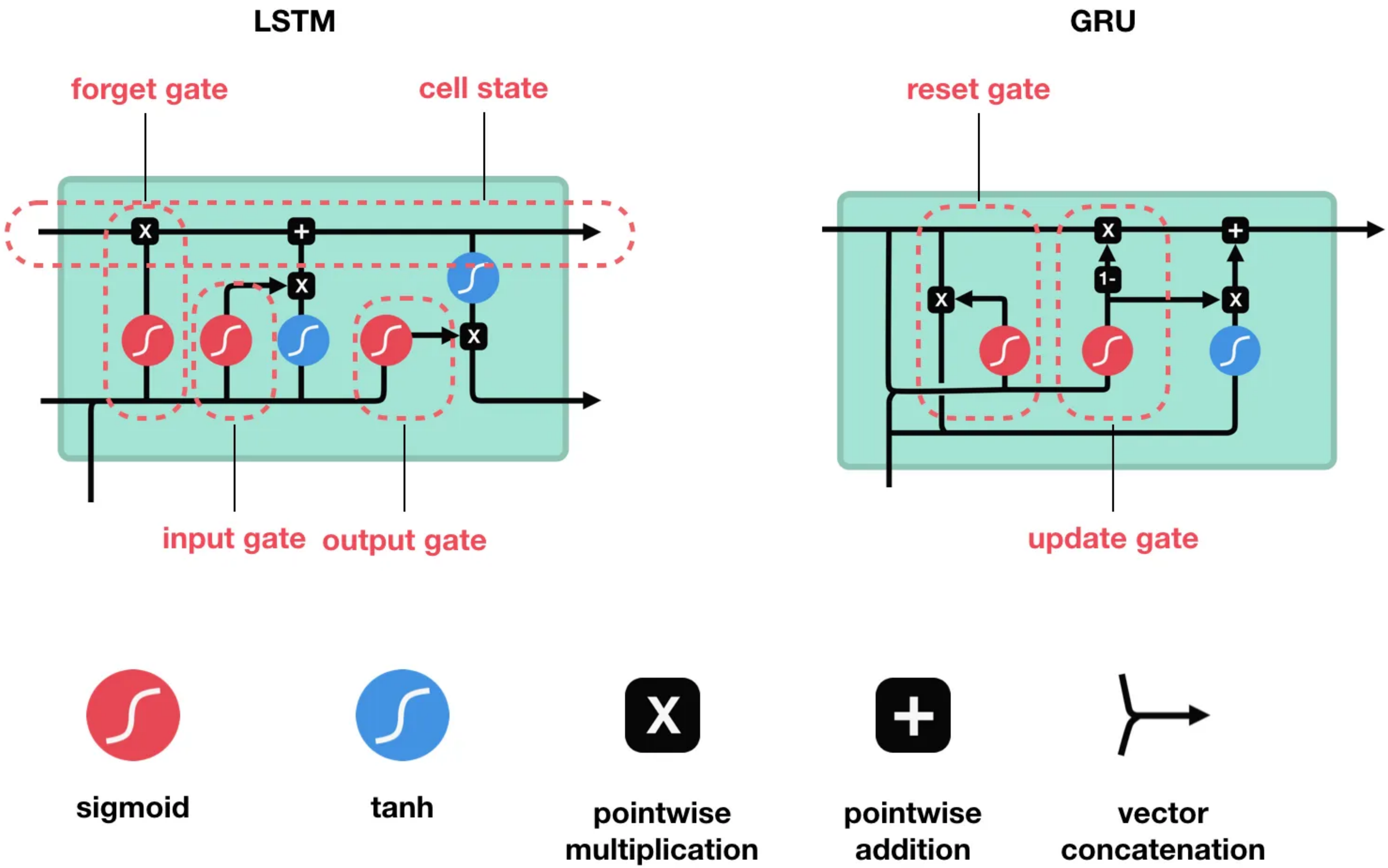
Long Short Term Memory



Recall ResNet

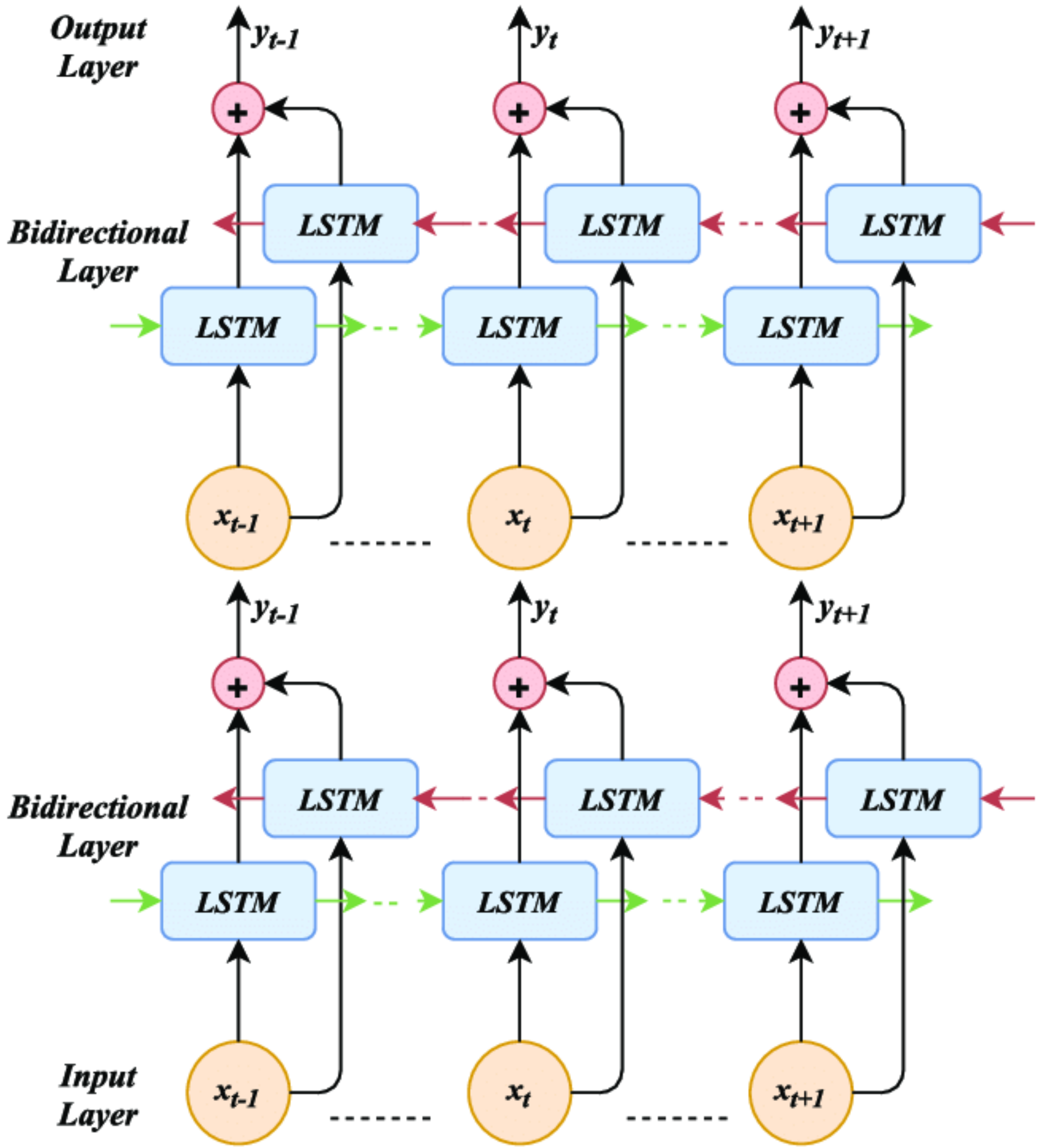
Source: https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture06-fancy-rnn.pdf

LSTM and GRU



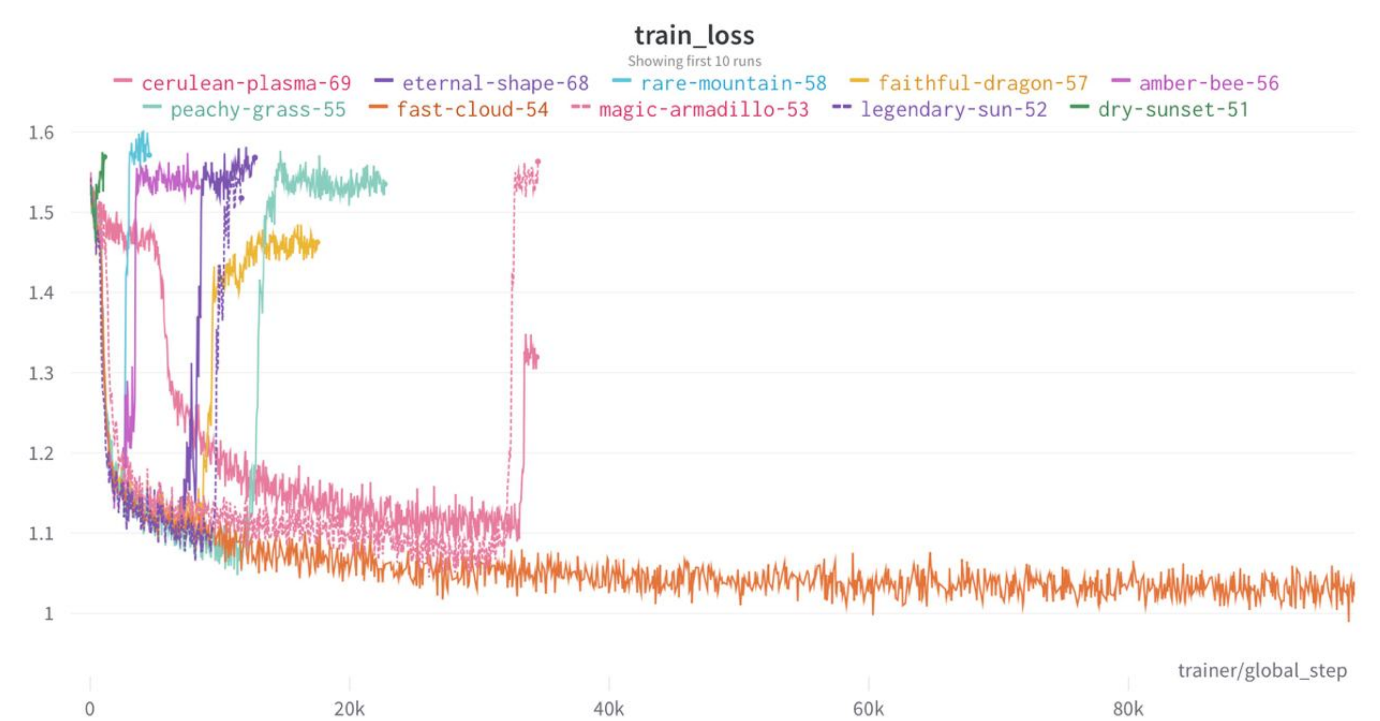
Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Multilayer and Bidirectional



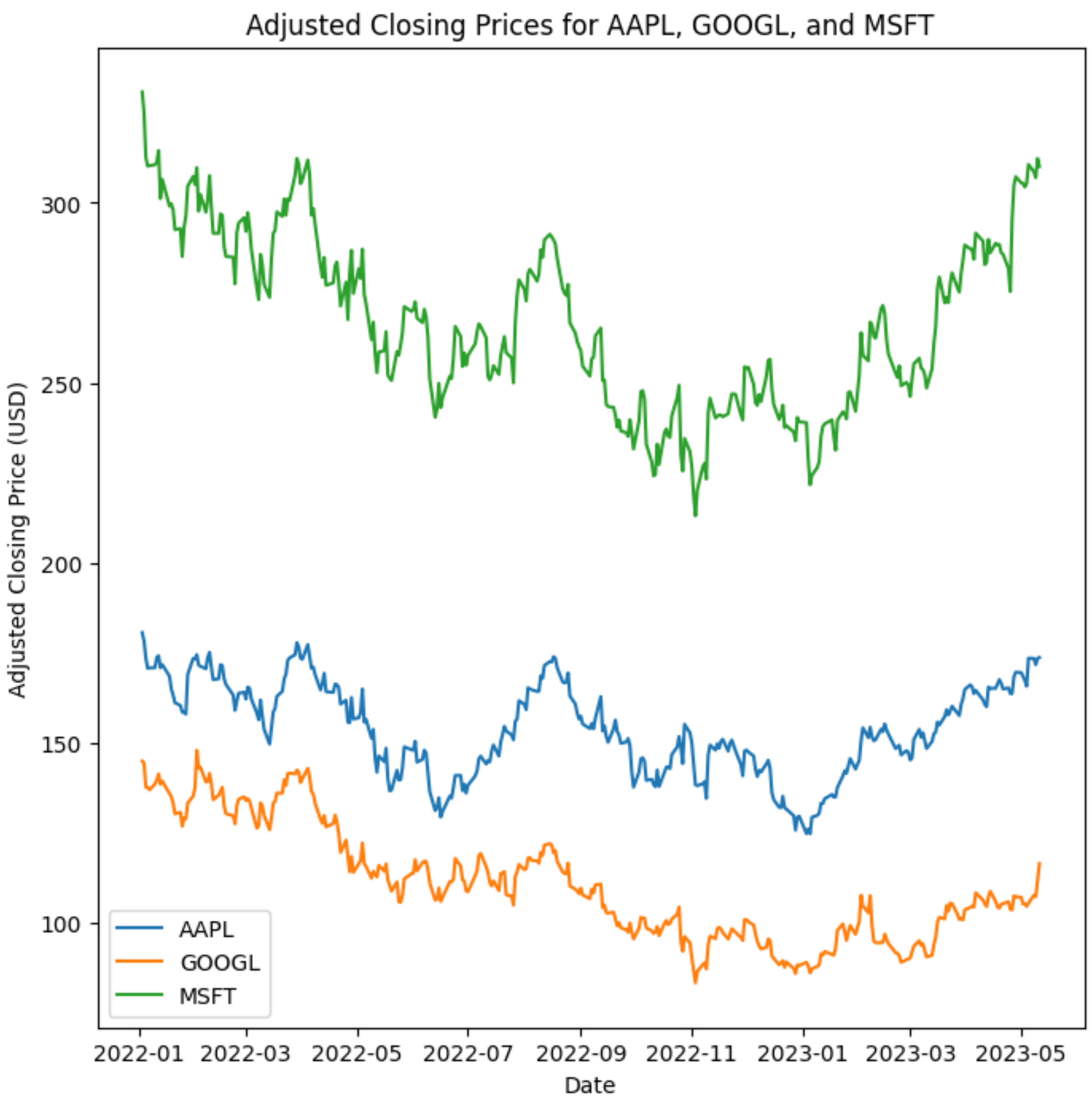
Practical tips

- Zero padding
- Packing/masking of padded inputs to increase performance (torch.nn.utils.rnn.pack_padded_sequence, tf.keras.layers.Masking)
- Fix random seed
- Gradient clipping
- Initial distribution matters (for LSTMs the one from tensorflow seems to be better)



Natural Language Processing

The code to draw the plot was generated by a language model. How do they work?



Here is the Python code to fetch the data and plot the closing prices:

python Copy code

```python
import yfinance as yf
import matplotlib.pyplot as plt

# Define the ticker symbols for the stocks we are interested in
tickers = ['AAPL', 'GOOGL', 'MSFT']

# Download the stock data
data = yf.download(tickers, start='2022-01-01', end='2023-05-12')

# Extract the adjusted closing prices
closing_prices = data['Adj Close']

# Plot the adjusted closing prices
plt.figure(figsize=(10, 6))
for ticker in tickers:
    plt.plot(closing_prices.index, closing_prices[ticker], label=ticker)

plt.xlabel('Date')
plt.ylabel('Adjusted Closing Price (USD)')
plt.title('Adjusted Closing Prices for AAPL, GOOGL, and MSFT')
plt.legend()
plt.show()
```

Natural Langage Processing

Representing words

- Vocabulary: enumerate all words
  (more specifically BPE — Byte Pair Encoding)

Tokens 220 **Characters** 747

```
Week 4 (Convolutional Neural Networks (CNNs), Recurrent Neural Networks (
RNNs), and Auto-Encoders (AE)):
May 15: 13:15-17:00: Convolutional Neural Networks (CNNs) and image
 analysis (Daniel Murnane).
    Exercise: Recognize images (MNIST dataset, sparse chips for
 radiation, and/or insoluables from Greenland ice cores) with a CNN.
May 17: 9:15-12:00: Recurrent Neural Networks (RNN), Long Short Term
 Memory (LSTM) and Natural Language Processing (NLP) (Inar Timiryasov).
    Exercise: Use an LSTM to predict flight traffic and do Natural
 Language Processing on IMDB movie reviews.
May 17: 13:15-17:00: (Variational) Auto-Encoder and anomaly detection (TP
).
    Exercise: Compress images using Auto-Encoder, and cluster latent
 space with UMAP.
```

TEXT TOKEN IDS

A helpful rule of thumb is that one token generally corresponds to ~4 characters of text for common English text. This translates to roughly ¾ of a word (so 100 tokens ~= 75 words).

Tokens 220 **Characters** 747

```
[20916, 604, 357, 3103, 85, 2122, 282, 47986, 27862, 357, 18474, 82, 828,
3311, 6657, 47986, 27862, 357, 49, 6144, 82, 828, 290, 11160, 12, 4834,
19815, 364, 357, 14242, 8, 2599, 220, 198, 6747, 1315, 25, 1511, 25,
1314, 12, 1558, 25, 405, 25, 34872, 2122, 282, 47986, 27862, 357, 18474,
82, 8, 290, 2939, 3781, 357, 19962, 337, 700, 1531, 737, 198, 220, 220,
220, 220, 32900, 25, 31517, 1096, 4263, 357, 39764, 8808, 27039, 11,
29877, 12014, 329, 11881, 11, 290, 14, 273, 35831, 84, 2977, 422, 30155,
4771, 21758, 8, 351, 257, 8100, 13, 198, 6747, 1596, 25, 860, 25, 1314,
12, 1065, 25, 405, 25, 3311, 6657, 47986, 27862, 357, 49, 6144, 828,
5882, 10073, 35118, 14059, 357, 43, 2257, 44, 8, 290, 12068, 15417,
28403, 357, 45, 19930, 8, 357, 818, 283, 5045, 9045, 292, 709, 737, 198,
220, 220, 220, 220, 32900, 25, 5765, 281, 406, 2257, 44, 284, 4331, 5474,
4979, 290, 466, 12068, 15417, 28403, 319, 8959, 11012, 3807, 8088, 13,
198, 6747, 1596, 25, 1511, 25, 1314, 12, 1558, 25, 405, 25, 357, 23907,
864, 8, 11160, 12, 27195, 12342, 290, 32172, 13326, 357, 7250, 737, 198,
220, 220, 220, 220, 32900, 25, 3082, 601, 4263, 1262, 11160, 12, 27195,
12342, 11, 290, 13946, 41270, 2272, 351, 471, 33767, 13]
```

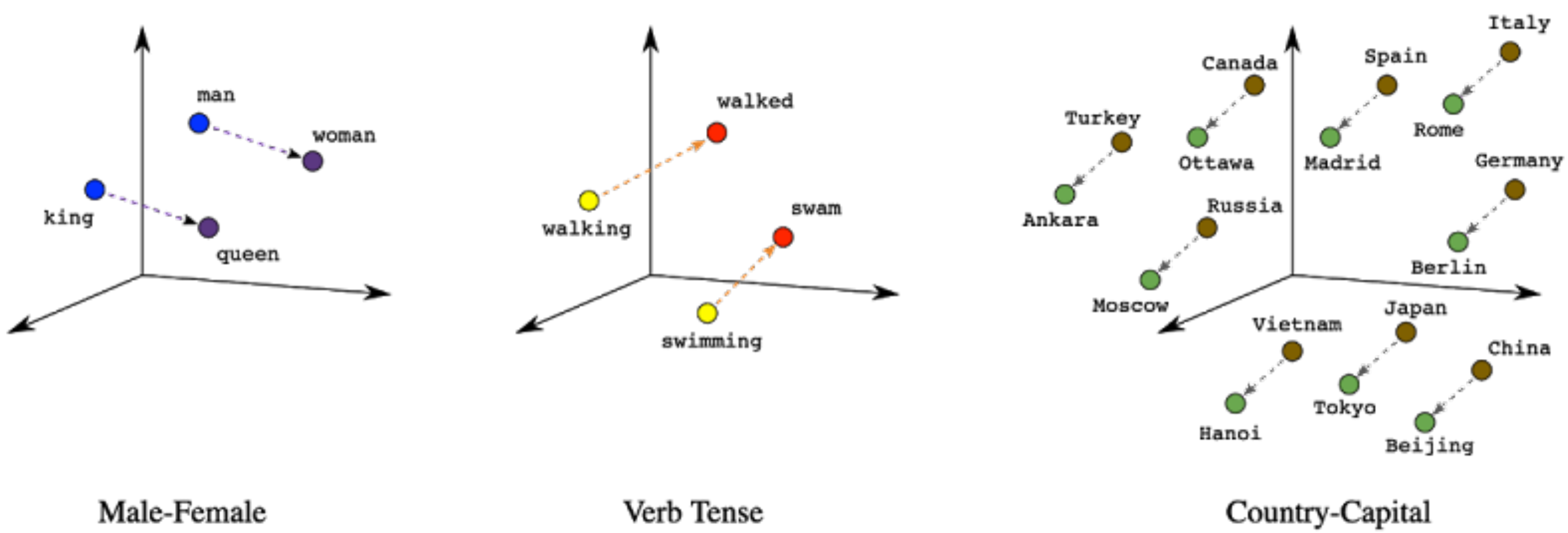
TEXT TOKEN IDS

https://platform.openai.com/tokenizer

Representing words

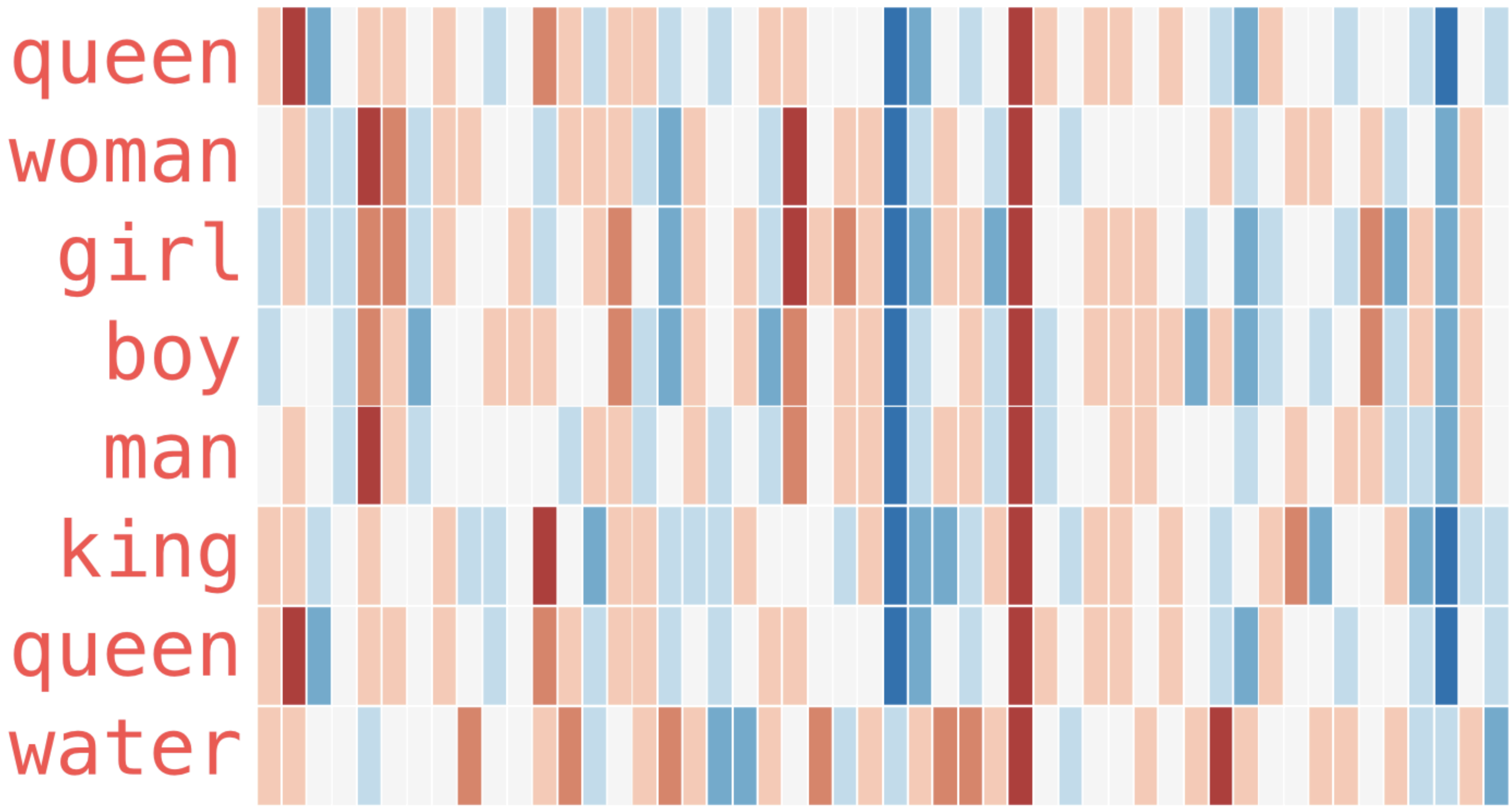
- Vocabulary: enumerate all words
(more specifically BPE — Byte Pair Encoding)

- Embeddings — every word is a vector in a multidimensional space



Source: https://cloud.google.com/blog/topics/developers-practitioners/meet-ais-multitool-vector-embeddings

Representing words: Embeddings



Operations over vectors: king - man + woman ~= queen

Source: https://jalammar.github.io/illustrated-word2vec/

Language modelling

- Given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$
  predict $P(x^{(t+1)} | x^{(1)}, x^{(2)}, \ldots, x^{(t)})$

- Applications: autocomplete, machine translation, speech recognition, sentiment analysis, information retrieval,…, text generation (chatGPT)

- Until ~2017 LSTMs dominated the field

- 2017: Transformers

Attention is all you need
A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc
The dominant sequence transduction models are based on complex recurrent orconvolutional neural networks in an encoder and decoder configuration. The best performing such models also connect the encoder and decoder through an attentionm echanisms. We propose a novel, simple network architecture based solely onan attention mechanism, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superiorin quality while being more …
☆ Save 99 Cite Cited by 74101 Related articles All 46 versions »

Transformers

- Power all well known language models, such as BERT, GPT, PALM, LLaMA,…

- Very parallelizable

- Fixed sequence lengths (4096 tokens for GPT-3.5)

- Complexity grows quadratically with the sequence lengths

- Resources:

  https://jalammar.github.io/illustrated-transformer/
  The ultimate experience:
  Let's build GPT: from scratch, in code, spelled out
  by Andrej Karpathy https://youtu.be/kCc8FmEb1nY
  https://github.com/karpathy/ng-video-lecture

Large Language Models

- GPT-3: 175B parameters

- Worst case — using float32:
  every parameter 4 bytes
  Weights only: $175 \times 10^9 \times 4$ bytes = 700 GB
  Activations ~ similar to model size +700 GB
  1400 / 80 = 17.5
  One would need 18 x NVIDIA A100 80GB for inference

- currently chatGPT is likely using a smaller model