Recurrent Neural Networks

COMP 4630 | Winter 2025

Charlotte Curtis

Overview

- Dealing with sequence data
- Feedforward vs recurrent networks
- References and suggested reading:
  - Scikit-learn book: Chapter 15
  - Deep Learning Book: Chapter 10

But first, transfer learning

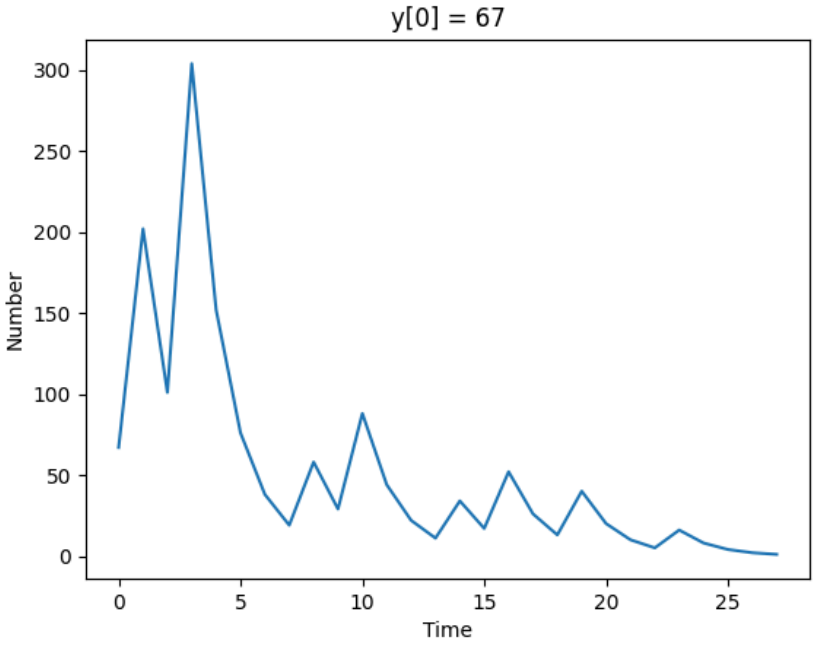
> *"If I have seen further, it is by standing on the shoulder of giants"*
> *-- Isaac Newton*

- **Transfer learning** copy pastes a trained network into a new task
- You can select which layers to keep, which to freeze, and which to re-train
- You can also drop new layers on top of the old ones
- Most of the time you want to freeze the early layers and add a new "head"
- ❓ Why are the early layers more general?

Sequence data

- So far we've been talking about images, tabular data, and other "static" data
- ❓ What are some examples of sequence data?

$$y[n] = \begin{cases} \frac{y[n-1]}{2} & \text{if } n \text{ is even} \\ 3y[n-1]+1 & \text{if } n \text{ is odd} \end{cases}$$

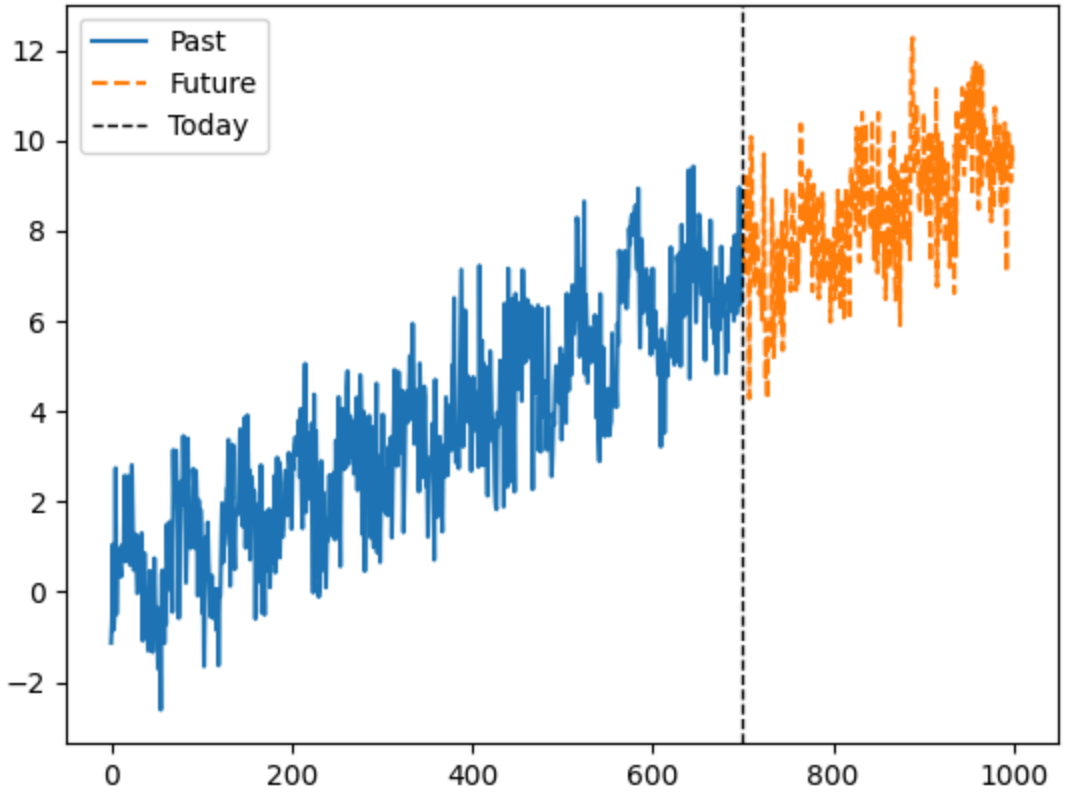


This is a famous sequence called the Collatz conjecture

Non-RNN Approaches

As usual, you don't always need a deep learning solution 🔨

- ❓ What is an example of a "naive" approach?
- ❓ What are some limitations of naive approaches?



Autoregressive Moving Average

- Models to predict time series with a weighted average of past value

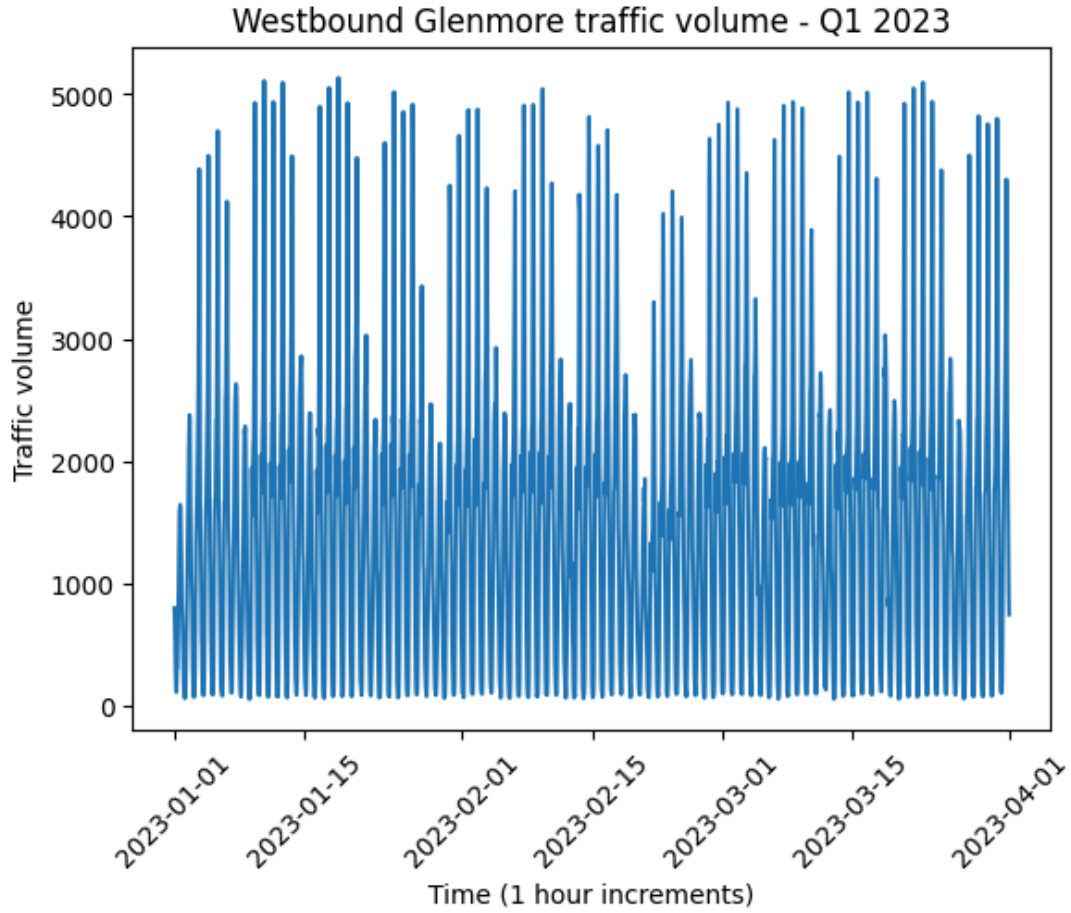
$$\hat{y} = \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

  where $\epsilon_t = y_t - \hat{y}_t$

- Key assumption: data is **stationary** (mean and variance don't change)
- ARIMA adds on "integration" or "differencing" to account for trends

Trends, Seasonality, and Assumptions

- ❓ Are there any obvious trends in the data?
- ❓ What about non-obvious trends?
- ❓ How might this dataset be treated differently from the previous one?



Feedforward vs recurrent networks

- Feedforward: data flows in one direction (then backpropagated)
- Recurrent: data can flow in loops

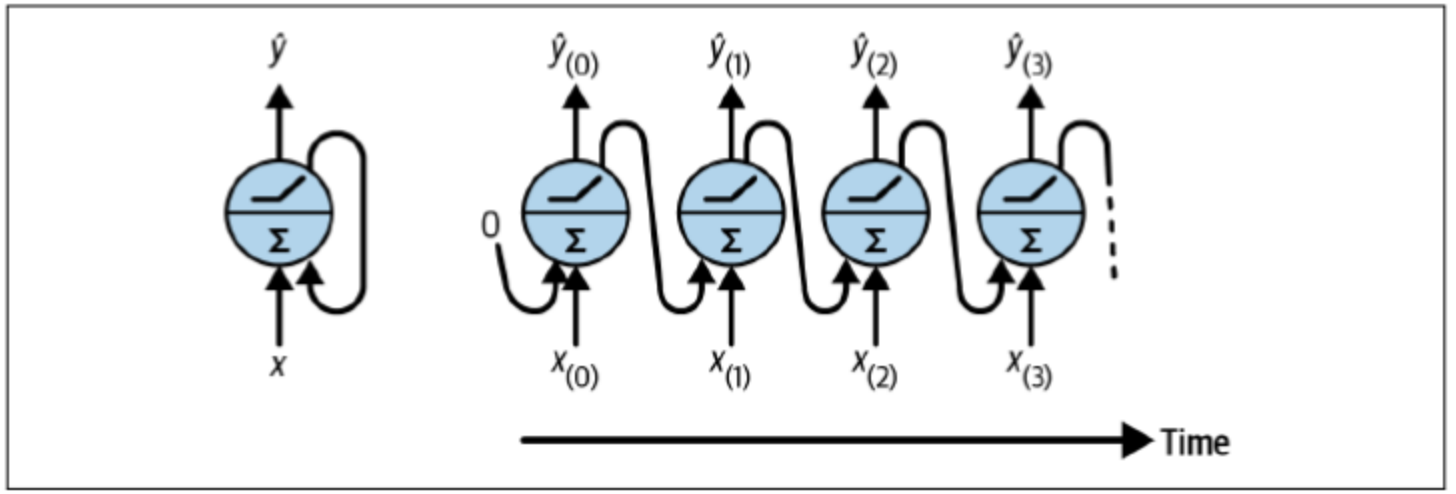


Figure 15-1. A recurrent neuron (left) unrolled through time (right)

Figure from Scikit-learn textbook

Recurrent layers

- The simplest recurrent layer has a single feedback connection

$$\hat{\mathbf{y}}_t = f(\mathbf{W}_x^T \mathbf{x}_t + \mathbf{W}_{\hat{y}}^T \hat{\mathbf{y}}_{t-1} + \mathbf{b})$$

  where f is the activation function and $\mathbf{W}_x$ and $\mathbf{W}_{\hat{y}}$ are weight matrices
- "Backpropagation through time" (BPTT) is exactly the same as regular backpropagation through the **unrolled** network
- ❓ What kind of issues might arise during training?
- ❓ What are some limitations of this approach?
- ❓ How can we deal with $\mathbf{y}_{t-1}$ for $t = 0$?

Preparing data for RNNs

- The data format depends on the task, e.g. do you want to predict:
  - The next value in a sequence (e.g. predictive text)
  - The next n values in a sequence (e.g. stock prices)
  - The next sequence in a set of sequences (e.g. language translation)
- Let's start with predicting the next value in a sequence

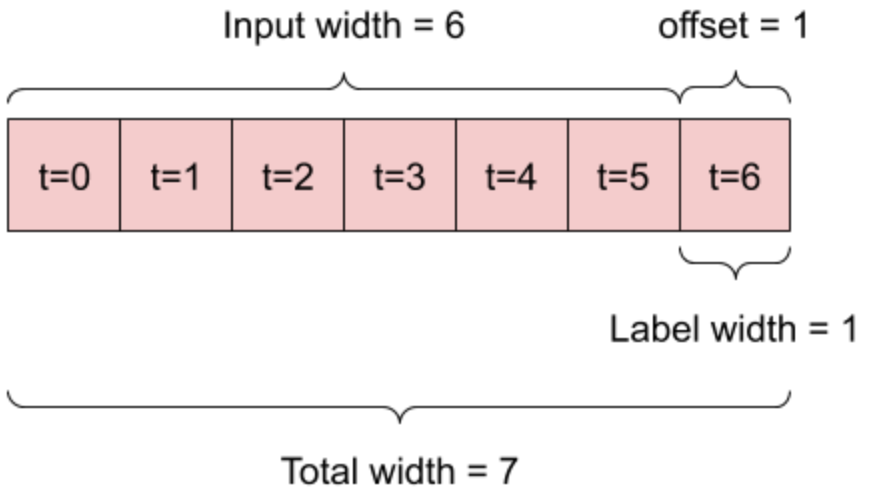


Image from TensorFlow Time Series Tutorial

Activation Functions for RNNs

- The default activation function in tensorflow is `tanh`
- In the simple example, I had to change to `relu` because my data was not normalized and `tanh` was saturating
- ❓ What is different about RNNs that might influence the choice of activation function?
- ❓ How might we normalize sequence data?

Beyond the "next value"

- Option 1: Use the single-prediction RNN repeatedly
- Option 2: Train the RNN to predict multiple values at once
  - Easy change model-wise, but data preparation is trickier
  - `n` inputs, `n` outputs
- Option 3: Use a "sequence to sequence" model
  - Even trickier data preparation, but `n` inputs are predicted at each time step instead of just at the end

Seq2seq input/target examples

n	Input	Target
1	`[0, 1, 2]`	`[1, 2, 3]`
2	`[0, 1, 2]`	`[[1, 2], [2, 3], [3, 4]]`
3	`[0, 1, 2]`	`[[1, 2, 3], [2, 3, 4], [3, 4, 5]]`

Problems with long sequences

- Gradient vanishing/exploding
  - Choose activation functions and initialization carefully
  - Consider "Layer normalization" (across features)
- "Forgetting" early data
  - Skip connections through time
  - "Leaky" RNNs
  - Long short-term memory (LSTM)
- Computational efficiency and memory constraints
  - Gated recurrent units (GRUs)

Skip connections and leaky RNNs

- Simple way of preserving earlier data:
- Vanilla RNN: $h^{(t)}$ depends on $h^{(t-1)}$ only
- Skip connection: $h^{(t)}$ depends on $h^{(t-1)}$, $h^{(t-2)}$, $h^{(t-n)}$, etc.
- Leaky RNN has a smooth "self-connection" to dampen the exponential:

$$h^{(t)} = \alpha h^{(t-1)} + (1 - \alpha) h^{(t)}$$

- Not common approaches anymore, as LSTM, GRU, and especially attention mechanisms are more popular

Long Short-Term Memory (LSTM)

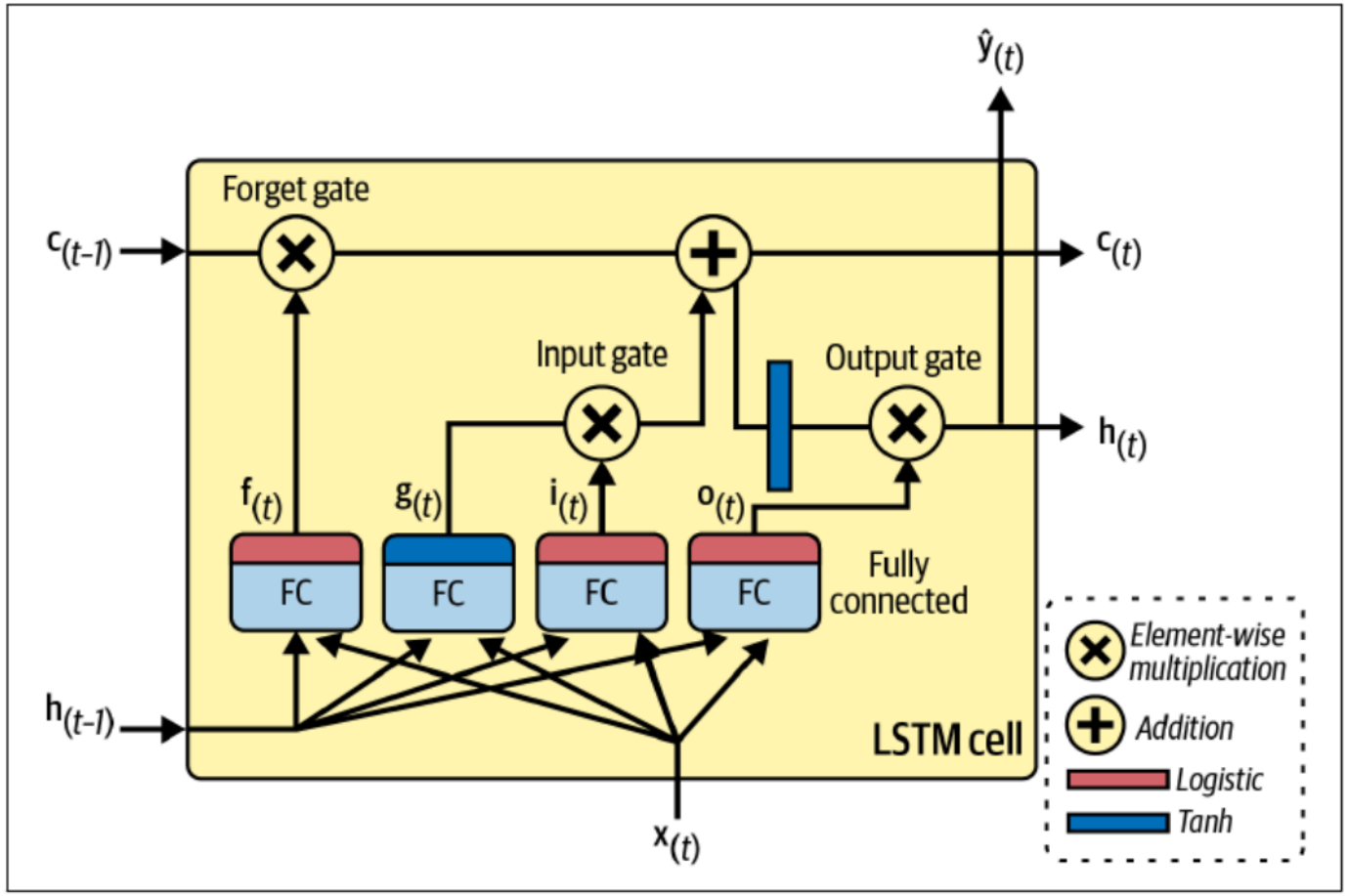


Figure 15-12. An LSTM cell

Figure from Scikit-learn textbook

Gated Recurrent Units (GRUs)

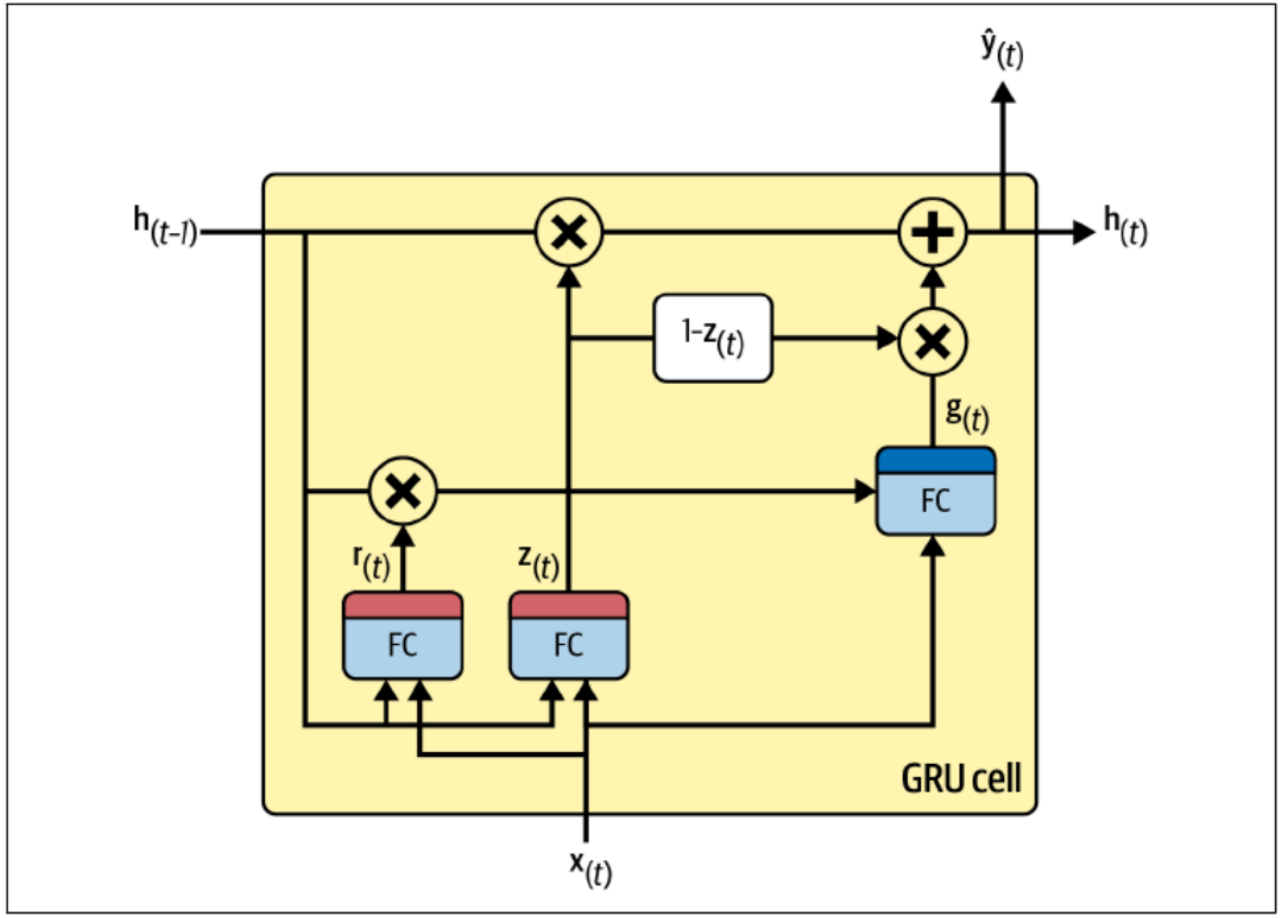


Figure 15-13. GRU cell

Figure from Scikit-learn textbook

Next up: Natural Language Processing

Preview: Natural Language Processing

- Natural Language Processing (NLP) is a field of study that focuses on the interaction between computers and human language.
- RNNs are widely used in NLP tasks such as language modeling, machine translation, sentiment analysis, and text generation.
- Language modeling involves predicting the next word in a sequence of words, which can be done using RNNs.
- Machine translation uses RNNs to translate text from one language to another.
- Sentiment analysis aims to determine the sentiment or emotion expressed in a piece of text, and RNNs can be used for this task.
- Text generation involves generating new text based on a given input, and RNNs are commonly used for this purpose.

Preview: Natural Language Processing

- What is Natural Language Processing (NLP)?
- Common NLP tasks:
  - Language modeling
  - Machine translation
  - Sentiment analysis
  - Text generation
- How RNNs are applied in NLP