DEEP LEARNING:
Sequence Modeling

Gedeon M

Nyandwi JD

Introduction

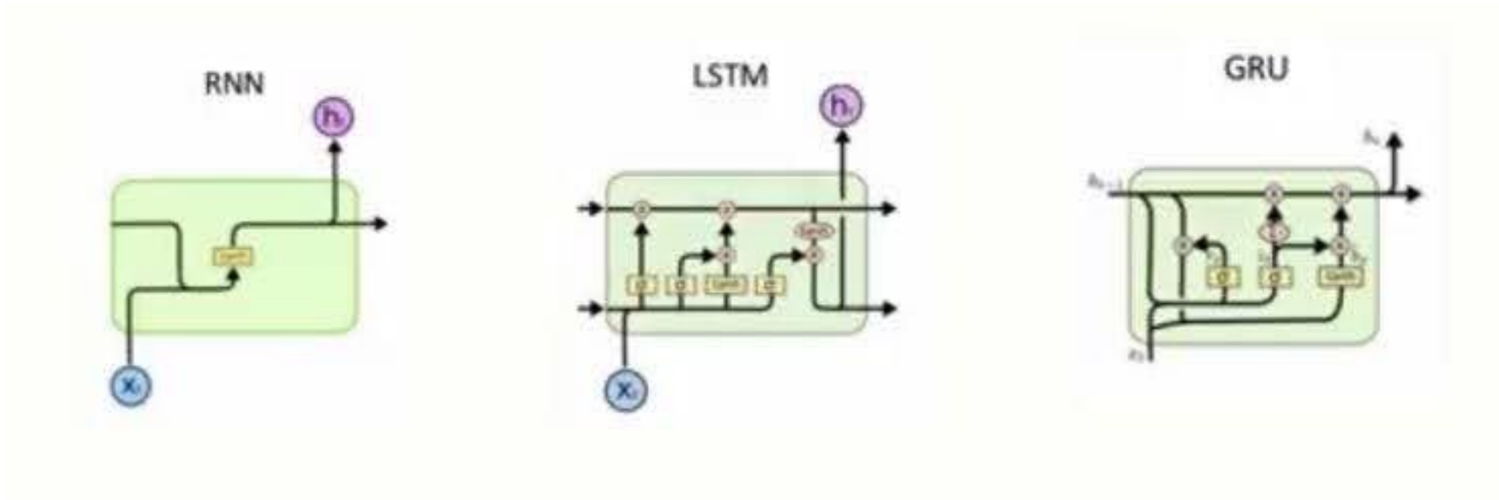
Sequence Models have been motivated by the analysis of sequential data such **text sentences**, **time-series** and other **discrete sequences data**

Applications of Sequence data

- Image Captioning
- Natural Language Processing
- Machine Translation
- Time Series
- Speech Recognition
- DNA Sequences Analysis

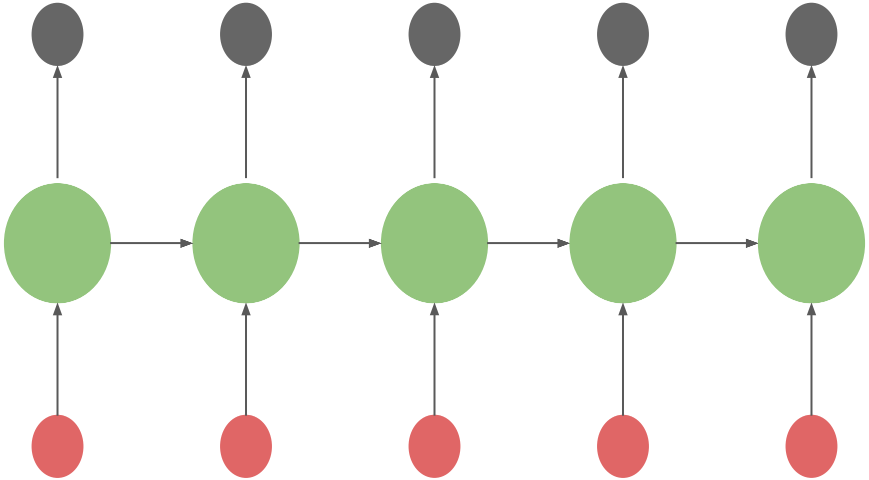
What is a sequence model

Sequence models are the machine learning models that input or output sequences of data.



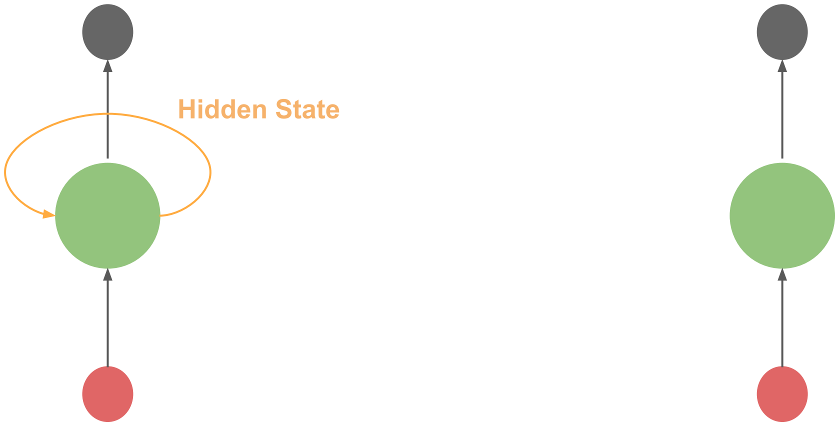
Recurrent Neural Networks

RNN are Neural Network good at modeling sequence data



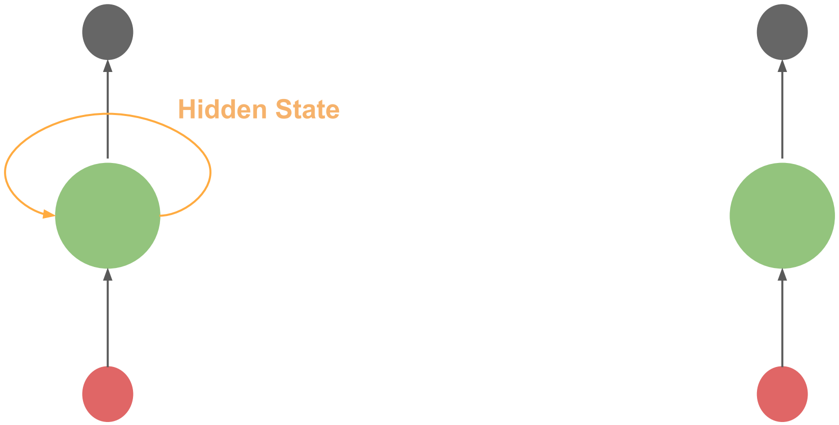
Recurrent Neural Networks

RNNs have a kind of sequential memory



Recurrent Neural Networks

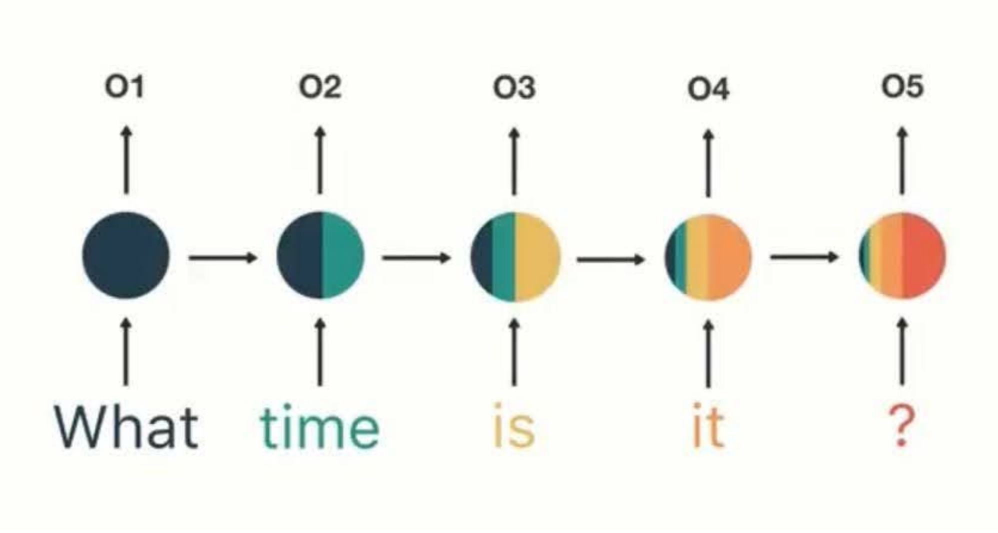
RNNs have a kind of sequential memory



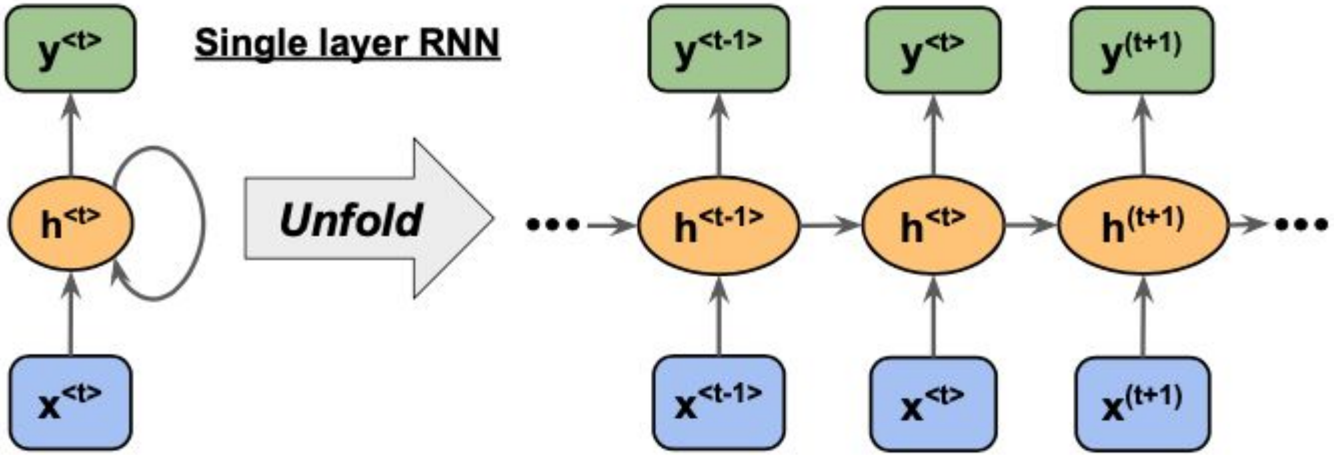
Sequential memory helps to realize the sequence pattern

Recurrent Neural Networks

RNNs have a kind of sequential memory

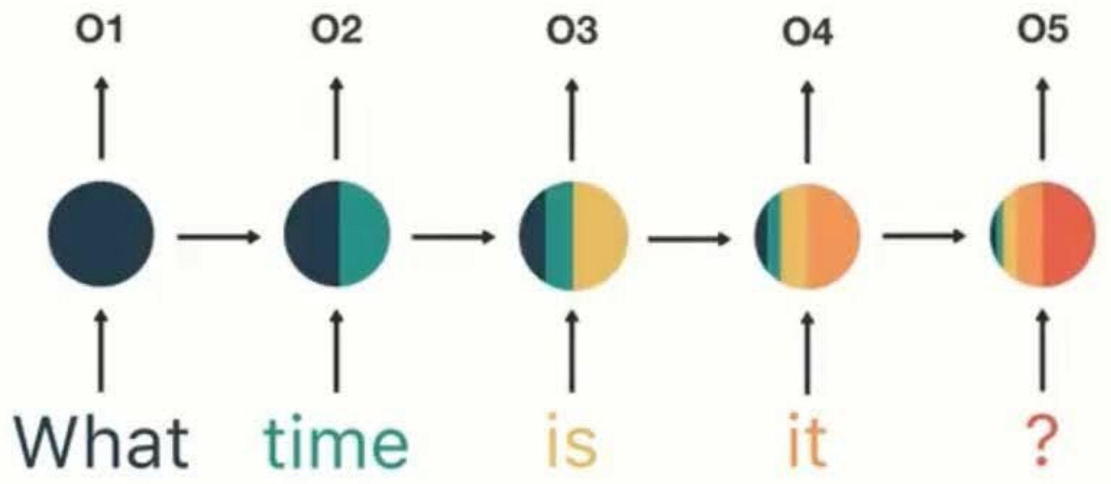


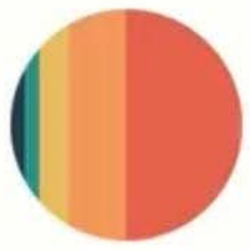
Recurrent Neural Networks



It is important to note that at every time step t the same function f is used and same set of weight parameters

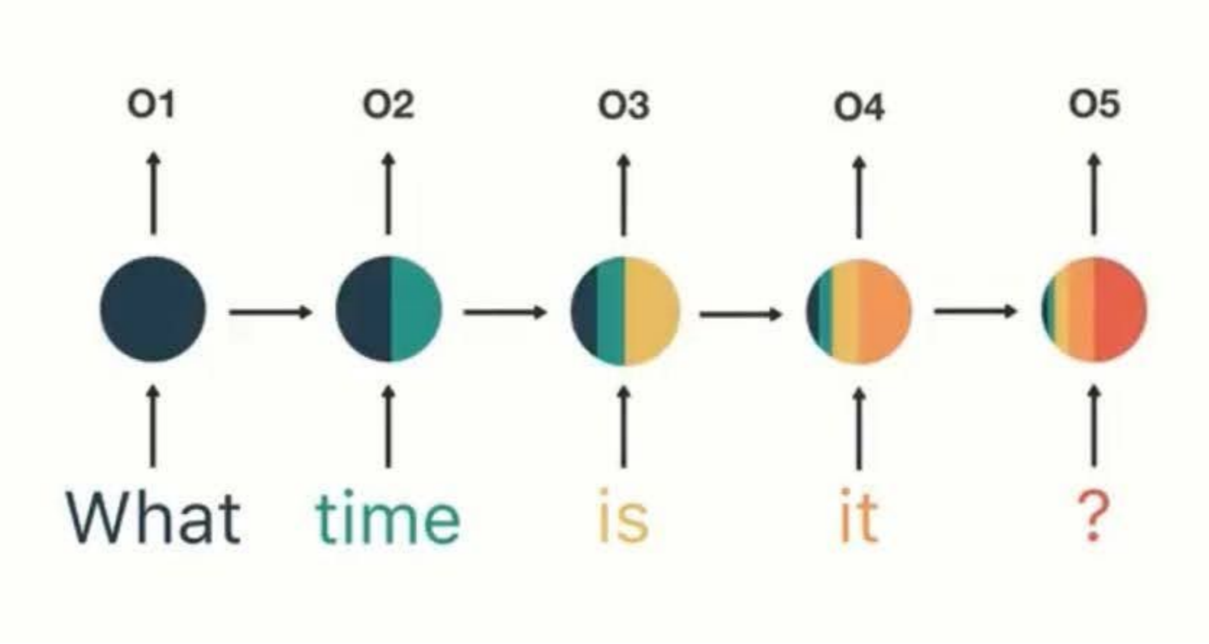
Recurrent Neural Networks: Problems

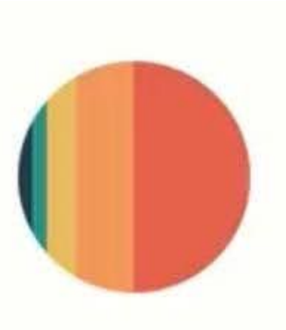




RNN passes through many different steps it has issue retaining previous information

Recurrent Neural Networks: Problems





RNN passes through many different steps it has issue retaining previous information: **This is what we call short term memory**

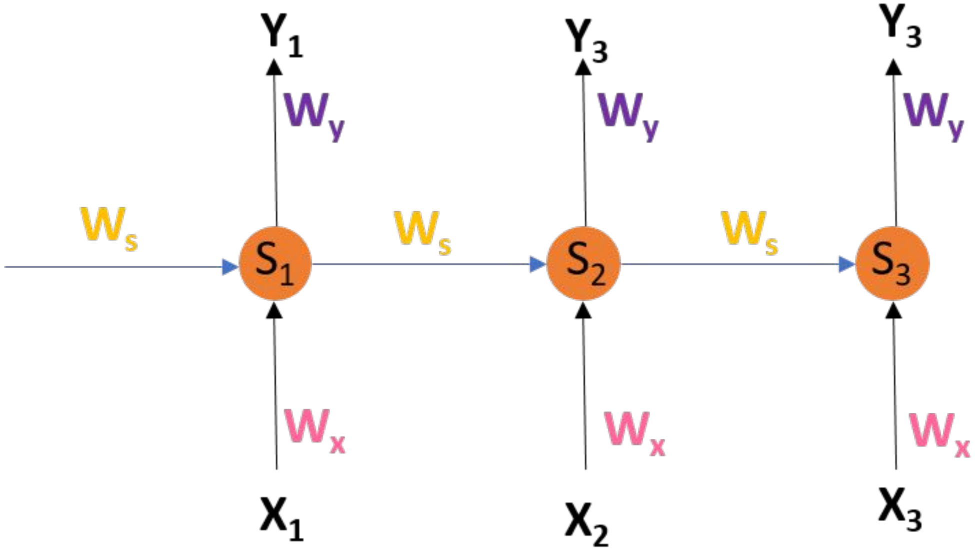
Recurrent Neural Networks: Pseudo code

```
rnn = RNN()
ff = FeedForwadNN()
hidde_state = [0.0, 0.0, 0.0, 0.0]

for word in input:
    output, hidde_state  = rnn(word, hidde_state )

prediction = ff(output)
```

Recurrent Neural Networks: Training



Backpropagation through time is a gradient-based technique for training recurrent neural networks

Backpropagation Through Time: Limitation

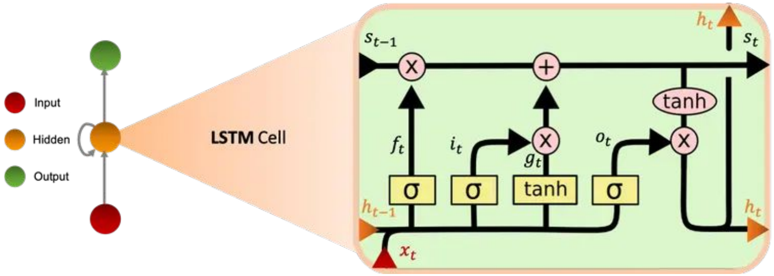
BPTT can be used up to a limited number of time steps like 8 or 10. If we back propagate further, the gradient becomes too small.

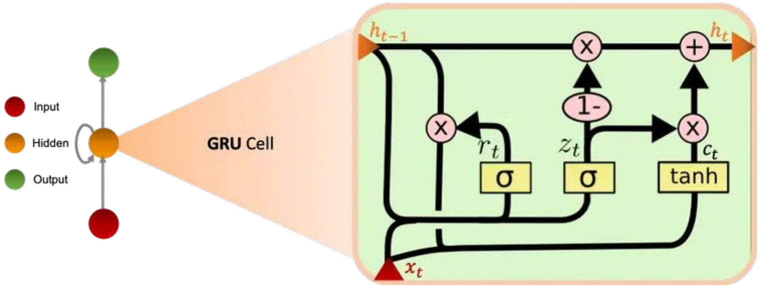
Backpropagation Through Time: Limitation

BPTT can be used up to a limited number of time steps like 8 or 10. If we back propagate further, the gradient becomes too small.

This is called **Vanishing Gradient**

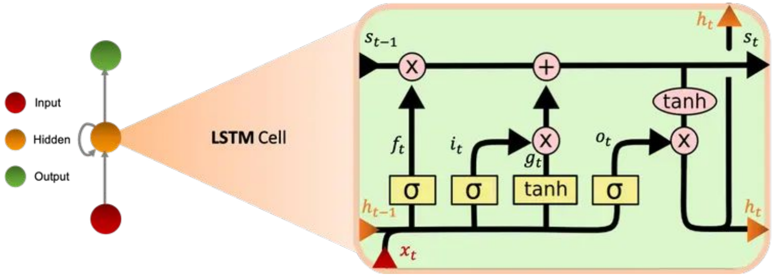
Going Further: LSTM and GRU

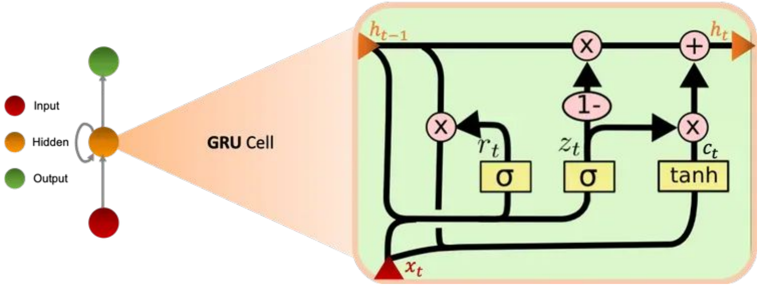




LSTM and GRU use a similar concept as RNN but they are capable of learning long-term dependencies using gates.

Going Further: LSTM and GRU





LSTM and GRU use a similar concept as RNN but they are capable of learning long-term dependencies using gates.

Gates are tensor operations that can learn what information to add or remove from the hidden state, hence **no short term memory problems**

Recap

1 RNN, LSTM and GRU are widely used for sequence modeling

2 RNN: Train faster, and use less computational resources, because there are less tensor operations

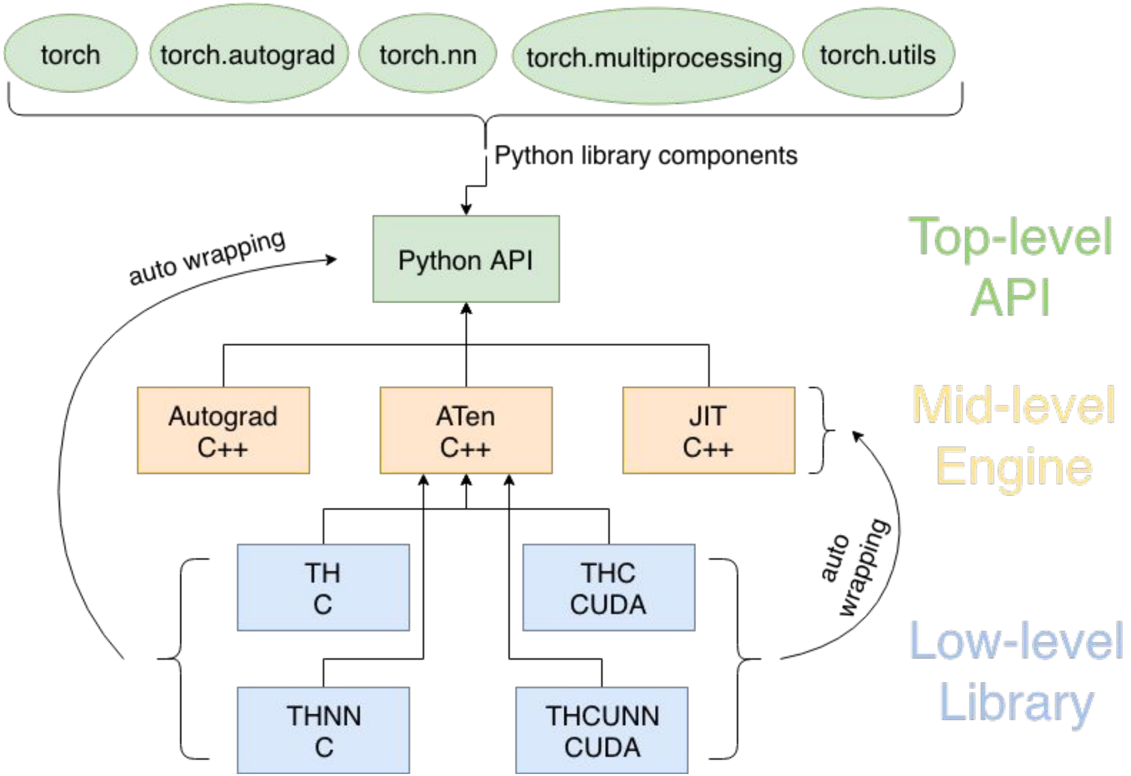
3 LSTM and GRU use a similar concept as RNN but they are capable of learning long-term dependencies using gates.

4 RNN suffer from short term memory and vanishing gradient

colab

https://bit.ly/3ARpWGF

PyTorch


torch
torch.autograd
torch.nn
torch.multiprocessing
torch.utils
Python library components
Python API
auto wrapping
Autograd C++
ATen C++
JIT C++
TH C
THC CUDA
THNN C
THCUNN CUDA
auto wrapping
Top-level API
Mid-level Engine
Low-level Library

Bonus:

Beginners steps for creating a NN model in PyTorch

1. Think about the task at hand, and the data to use
2. Step 1 will help you to know if your task is a classification, regression, detection, segmentation, translation,etc. The amount of the data would also help to decide the architecture to use.
3. Decide architecture either CNN, RNN, LSTM, GRU, NN, Transformers, etc.
4. Where will you read your data from? Either online, from a package or on a local storage
5. Read your data and create dataloaders (torch.utls.dataloaders)
6. Create model class (torch.nn)
7. Define criterion, loss function
8. Define Optimizers, learning algorithm
9. Train you model
10. Test your model

The End!