ECE 1508S2: Applied Deep Learning

Chapter 7: Sequence-to-Sequence Models

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2025

Learning Sequence from Sequence

In many applications, our dataset contains data-points that of the form

$$(\mathbf{x}[t], \mathbf{v}[t])$$

This is still standard supervised learning where

- *input data is a sequence, and*
- *label is a sequence*

We have already seen several examples with RNNs

- *We want to train a machine translator*
  ↳ *Dataset contains sequences of German sentences with English translations*
- *We want to caption an image*
  ↳ *Dataset contains images with sequences of caption sentences*
- *We want to predict next words*
  ↳ *Dataset contains sequences of sentences with label being last word*

Sequence-to-Sequence Problem

+ *What is the key point in such learning problems?*
– They are often called a *sequence-to-sequence problem, since we intend to learn an output sequence from an input sequence*

Sequence-to-Sequence (Seq2Seq) Models

A Seq2Seq model is a model, e.g., a NN, that takes a sequence as an input and returns an output sequence

+ *Then isn't RNN a Seq2Seq model?*
– Sure! Strictly speaking *even MLPs and CNNs are Seq2Seq* models with *sequences of length 1!*

Despite this definition, when we talk about Seq2Seq models in practice, we mainly refer to architectures with encoder and decoder

First Seq2Seq Model

Let's start with a simple task:

we intend to train a model that generates coherent sentences

- *After training it should be able to generate meaningful sentences*
  - ↳ *If we give in an incomplete sentence, it can complete it*
  - ↳ *If we don't give it an input, it generates a random meaningful sentence*

Since, we only know RNNs up to now, we are going to use an RNN

- *As model we want to train an RNN*
  - ↳ *It could be an LSTM, a GRU, or even a basic RNN*
  - ↳ *It can be shallow or deep*
- *We are going to train this RNN via a given dataset*
  - ↳ *We compute the loss by some loss function, e.g., cross-entropy*

Seq2Seq Model: *Basic Language Model*

We intend to train a model that generates coherent sentences

this is a basic natural language model

You learn in detail about it in ECE 1786: Creative Applications of NLP

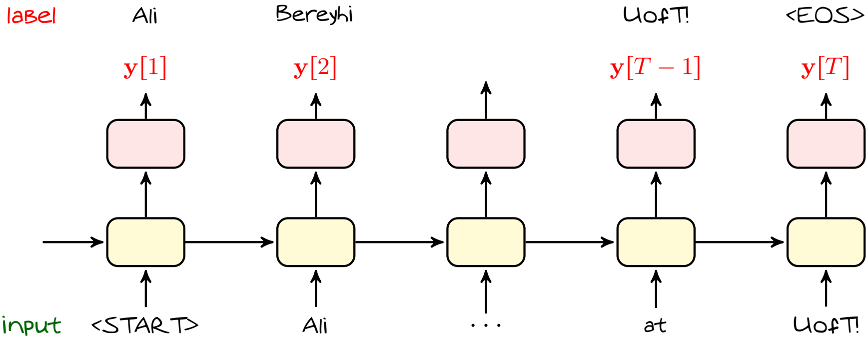
↳ *You may consider taking it in the next fall semester*

Let's write our dataset first: it contains sequences of coherent English sentences

Ali Bereyhi is the coolest professor at UofT!

- *Sentences are of different lengths, i.e., T is different for each sequence*
  - ↳ *Each word in a sentence is one input entry*
  - ↳ *We label each word with its next word in the sentence*

Basic Language Model: *Dataset*



- *To be able to predict first word and end of sentence we add two new words*
  - ↳ *We tag the beginning of sentence with* <START>
  - ↳ *We label the end of sentence with* <EOS>
- *We do not have the correspondence issue in this problem*

Basic Language Model: *Dataset*

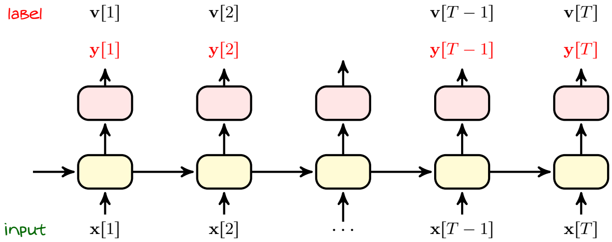
+ *How can we feed those words to our RNN?*
– We convert them to *vectors by some method*
    ↳ *You can learn those methods in ECE 1786*

The basic approach is to make a token for each word

- *We list all possible words and index them by 1 to D*
    ↳ *D could be very large: just imagine how many words we could say!*
    ↳ *The set of these words is what we call vocabulary*
- *We add <START> with index 0 and <EOS> with $D+1$*
- *We show each character by its one-hot vector which is called token, e.g.,*

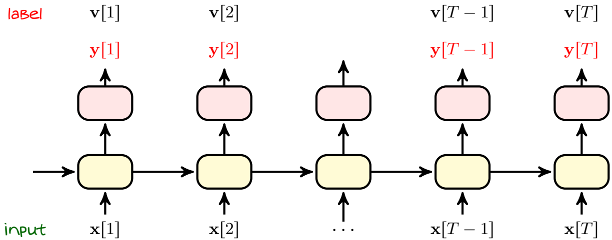
$$\text{<START>} \mapsto \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix} \in \{0,1\}^{D+2} \qquad \text{<EOS>} \mapsto \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} \in \{0,1\}^{D+2}$$

Basic Language Model: *Dataset*



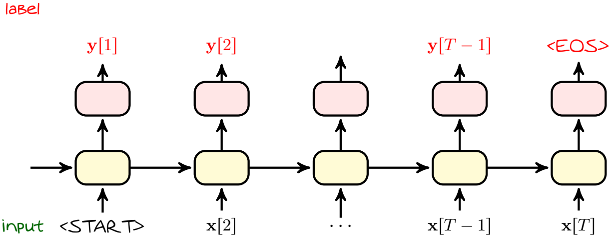
- *We can replace every word with its token*
  - ↳ *We now have a standard many-to-many setting*
  - ↳ *We could also use "embedding" to assign vectors to words*
    - ↳ *This will be taught in ECE 1786*

Basic Language Model: *Training*



- *We now train the RNN with our dataset*
  - ↳ *We break dataset into mini-batches*
  - ↳ *For each data-point we pass first forward and then backward through time*
  - ↳ *We compute aggregated loss for each point and average over mini-batch*
- *After a certain number of epochs, we have the trained RNN*

Basic Language Model: *Inference*



- *If we want to generate a random sentence, we can give <START>*
  - ↳ *It generates a word in each time step*
  - ↳ *Intuitively, these sentences are correlated to what RNN learned from dataset*
- *If we want to complete the sentence, we give the initial part*
  - ↳ *It keeps on generating till <EOS>*
  - ↳ *Intuitively, this is correlated to what RNN learned and the input part*

Sequence Generation: *Caption Generation*

Sentence completion worked simply with RNN: *mainly following the fact that entries of input and output sequences are of same nature*

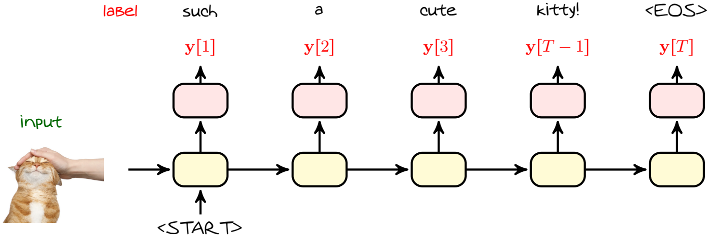
↳ *They are both tokens*

↳ *But in practice, we may have different types of sequences*

Let's consider another example: we want to train an NN that gets an image and writes a caption for it

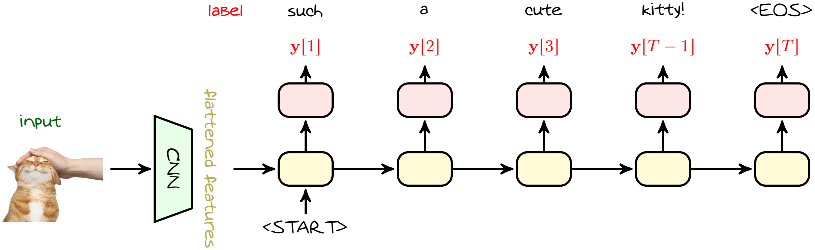
- *It gets as input a single image: a sequence of length one*
  - ↳ *For instance a 256 $\times$ 256 RGB image of a cat*
- *It returns a sentence: potentially a along sequence*
  - ↳ *For instance the sentence* such a cute kitty!

Caption Generation: *Model*



- *We can to generate a meaningful sentence with our basic language model*
  - ↳ *The sentence is probably not relevant to the image*
  - ↳ *We need to make the RNN speak about the cat image*
- *Maybe, we could set initial state of the RNN depending on the image*
  - ↳ *We need to extract features of image*
  - ↳ *Those features are going to explain what is inside the image*

Caption Generation: *Encoder-Decoder*



- *We can extract a rich vector of features from the image via a CNN*
  - ↳ *We use multiple convolutional layers to extract features*
  - ↳ *We flatten those features and give it as a initial state to the RNN*

This architecture is called an encoder-decoder model

↳ *A CNN is used to encode input to a good vector of features*

↳ *An RNN is used to decode extracted features to a desired label sequence*