Applied Deep Learning

Chapter 7: Sequence-to-Sequence Models

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

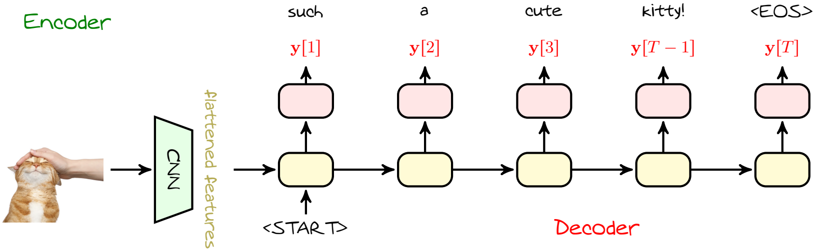
Encoder-Decoder Architecture

Encoder-Decoder

Encoder-decoder architecture comprises of two separate NNs

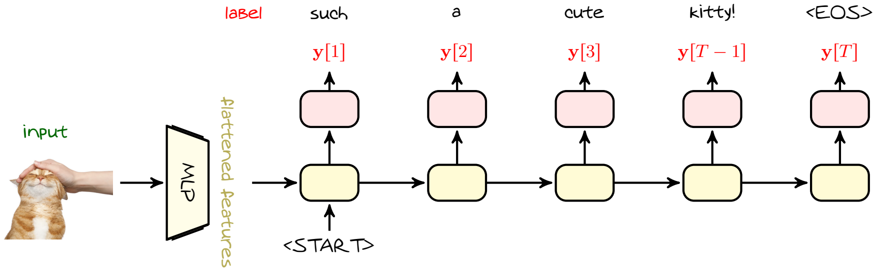
1. *Encoder takes the input sequence and encodes it into vector of features*
2. *Decoder takes vector of features and decodes it into output sequence*

+ *What kind of NNs should we use?*
– Pretty much everything is allowed!



Back to *Caption Generation*

We used a CNN for encoding and an RNN for decoding

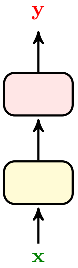


We can replace CNN with any other architecture the extracts features

RNN as Conditional Distribution

+ *But, why should those features make RNN speak about cat?*
– It makes RNN to *generate random words conditional to input image*

When we are dealing with classification: *NN can be seen as a machine that computes distribution and based on its input, it generates random outputs*

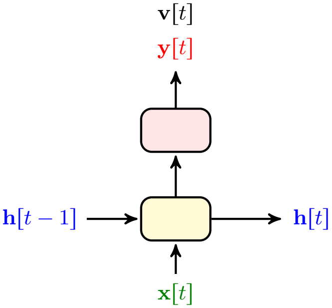


In classification $\mathbf{y}$ *is a vector if probability*

- *Its length equals to the number of classes*
- *Its entry* k *represents the probability of class* k
  - ↳ *We can say that*

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \dots \\ y_K \end{bmatrix} \leftrightsquigarrow y_k \propto \Pr\left\{\text{label} = k | \mathbf{x}\right\}$$

RNN as Conditional Distribution



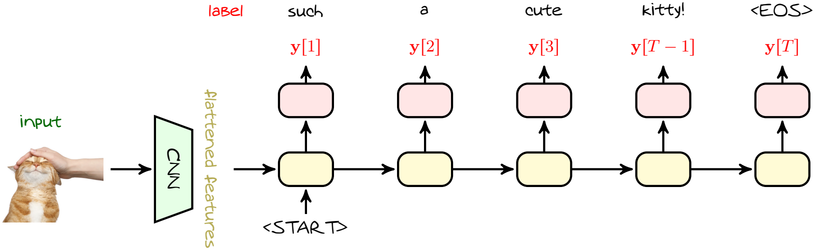
Similarly the output of RNN in each time can be seen as

$$y_k[t] \propto \Pr\left\{\text{label} = k | \mathbf{h}[t-1], \mathbf{x}[t]\right\} = p\left(\mathbf{v}[t] | \mathbf{h}[t-1], \mathbf{x}[t]\right)$$

Since $\mathbf{h}[t-1]$ *already contains memory about* $\mathbf{x}[1:t-1]$, *we could say*

$$y_k[t] \propto p\left(\mathbf{v}[t] | \mathbf{x}[1:t]\right)$$

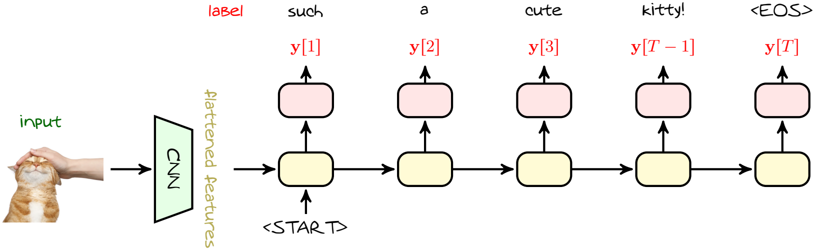
Caption Generation: *Dataset*



To train this architecture, we collect a dataset

- *It contains several images*
  - ↳ *They could potentially be of different classes*
- *For each image, we have a sample caption*
  - ↳ *These sentences are again of different lengths*

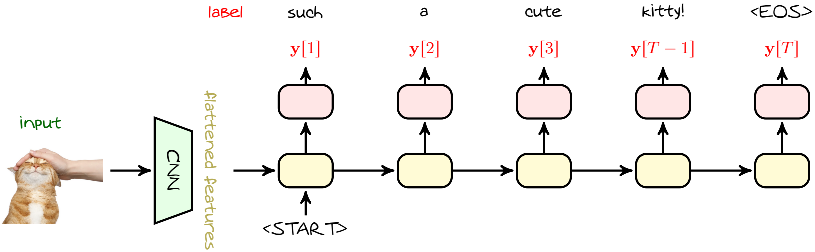
Caption Generation: *Training*



Say we want to train it on one sample: first we pass forward

- *We first tokenize the words in captions to take them as one-hot labels*
- *We pass the image forward through CNN and get the feature vector*
- *We initiate the RNN with the feature vector and give input <START>*
- *We pass forward through time till we see have the output sequence*

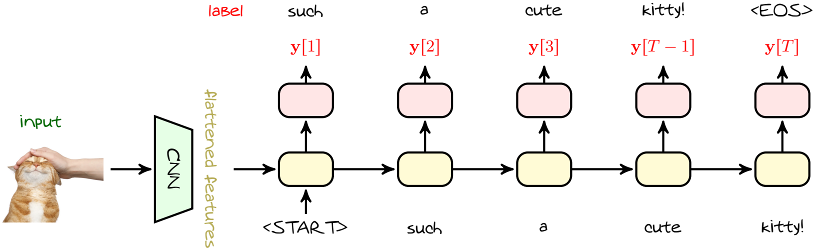
Caption Generation: *Training*



Say we want to train it on one sample: then we pass backward

- *We compute the loss between the output sequence and one-hot labels*
  - ↳ *We can simply use the cross-entropy function*
- *We backpropagate through time till we arrive at the beginning of decoder*
- *We have $\nabla_{\text{features}} \hat{R}$*
  - ↳ *So we backpropagate through the CNN*
- *We update all weights and go for the next round*

Caption Generation: *Inference*



Say we finished with training: we want to caption a new image

- *We send it over network and read the output sequence*
- *We could also set output of each time step as input for next time*
  - ↳ *We could also do it while training*
  - ↳ *Intuitively, it could help the RNN writing more coherent sentence*

Seq2Seq Model: *Basic Translator*

Now, let's take a step further:

we want to build a model that translates German sentences to English

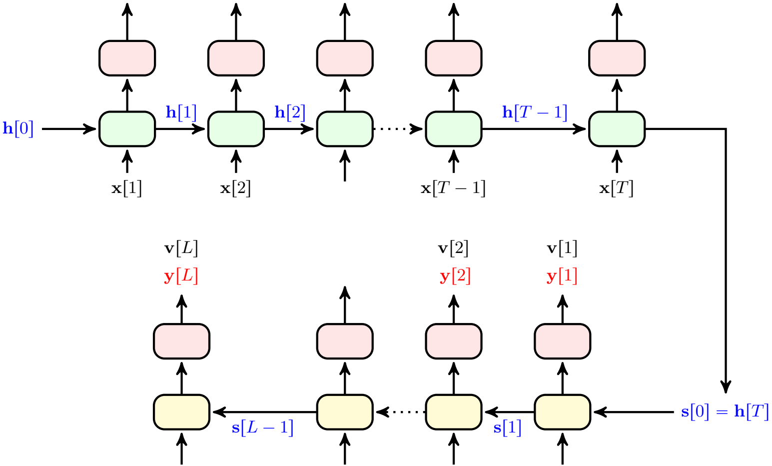
- *We need a Seq2Seq model*
  - ↳ *We have a sequence of input German words*
  - ↳ *We need to return a sequence of English words*
  - ↳ *These sequences could be of different lengths*

We know encoder-decoder model: *we use it to build our translator*

- *We need an encoder that takes a German sentence*
  - ↳ *RNN is a good choice, since we have input sequence*
- *We need a decoder that returns the English translation*
  - ↳ *RNN is again the choice, since we have another sequence*

Basic Translator: *Encoder-Decoder Model*

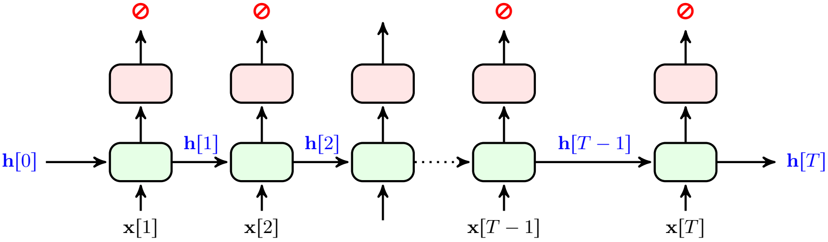
So, the model for our translator looks like this



Basic Translator: *Dataset*

To train this model, we collect some dataset: in this dataset

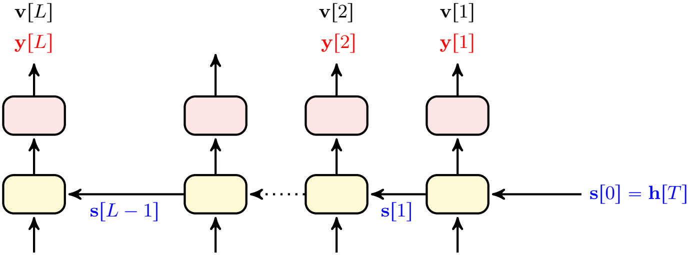
- *We have German sentences*
  - ↳ *We tokenize each sentence and represent it with a sequence* $\mathbf{x}[1:T]$



Basic Translator: *Dataset*

To train this model, we collect some dataset: in this dataset

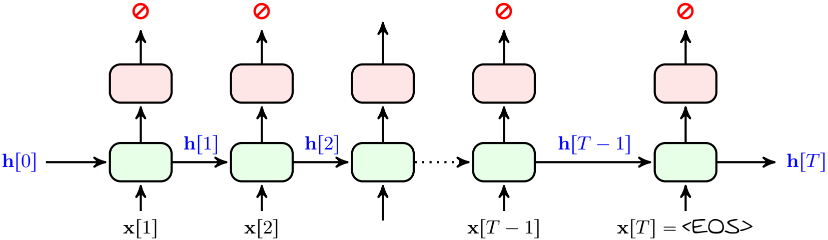
- *Corresponding to each German sentence, we have the English translation*
  - ↳ *We tokenize it as well and represent it with a sequence* $\mathbf{v}[1:L]$
  - ↳ L *and* T *are not of the same length*



Basic Translator: *Training*

Say we want to train it for sample sentence: we start with forward pass

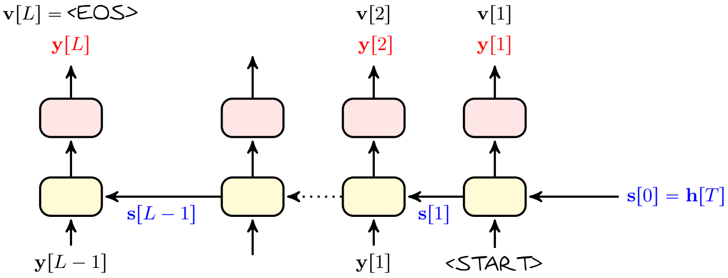
- *We pass the tokens through time forward till we arrive at* <EOS>
  - ↳ *At this point we have* $\mathbf{h}[T]$ *at the output of encoder*
- *We have already computed all variables inside this encoder*
  - ↳ *We need them in them backward pass*



Basic Translator: *Training*

Say we want to train it for sample sentence: we start with forward pass

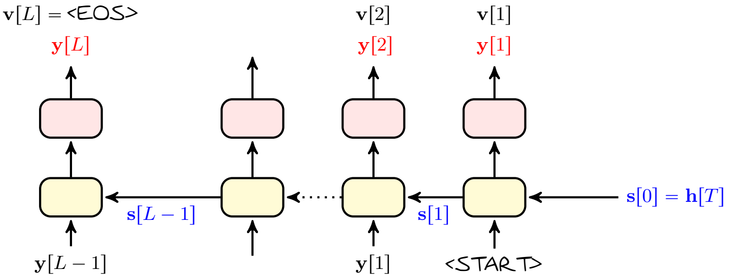
- *We initiate the decoder with* $\mathbf{s}[0] = \mathbf{h}[T]$
  - ↳ *We could also give token of* <START> *as first input*
  - ↳ *We can give* $\mathbf{y}[\ell - 1]$ *as the input at time* ℓ
- *We continue till we get to label* <EOS>



Basic Translator: *Training*

Say we want to train it for sample sentence: now we pass backward

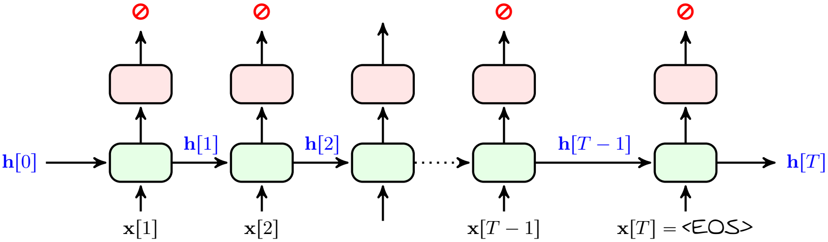
- *We compute loss between the labels and outputs*
  - ↳ *We aggregate cross-entropy losses over time*
- *We backpropagate through time*
  - ↳ *We get to* $\nabla_{\mathbf{s}[0]} \hat{R} = \nabla_{\mathbf{h}[T]} \hat{R}$



Basic Translator: *Training*

Say we want to train it for sample sentence: now we pass backward

- *We have* $\nabla_{\mathbf{h}[T]} \hat{R}$
  - ↳ *We backpropagate again over time*
- *We update all the weights and start a new round*

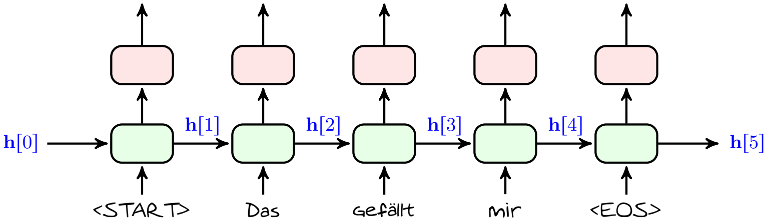


Basic Translator: *Inference*

Say we have trained this model and we want to use it to translate

"Das gefällt mir" $\rightsquigarrow$ *I like it*

Let's start with encoding

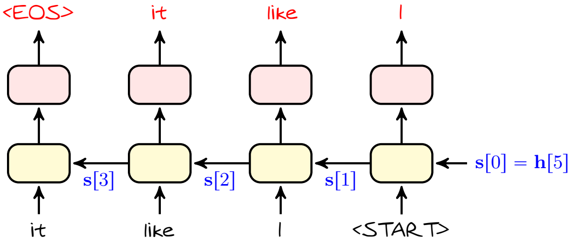


Basic Translator: *Inference*

Say we have trained this model and we want to use it to translate

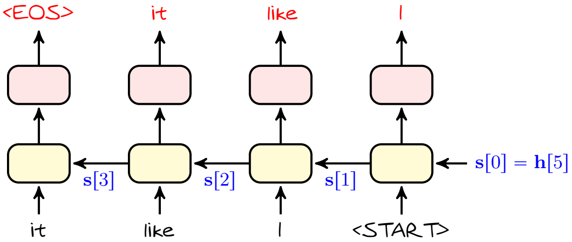
"Das gefällt mir" $\rightsquigarrow$ *I like it*

Now, for decoding we start with $\mathbf{h}[5]$



Basic Translator: *Inference*

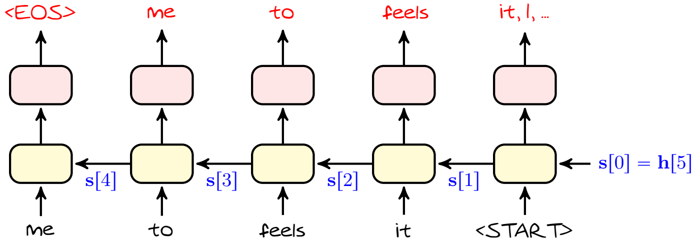
If we are lucky, the token of word "I" has highest probability in $\ell = 1$



and probably the RNN keeps generating a correct sentence

Basic Translator: *Inference*

If we are unlucky, another token might be slightly higher in probability at $\ell = 1$

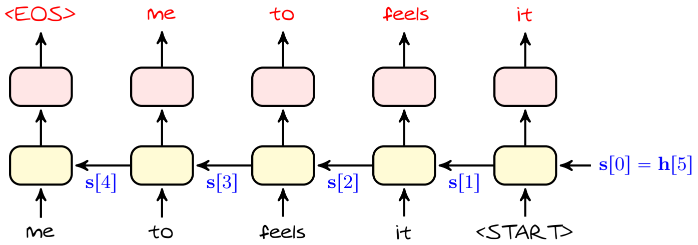


In this case, a small mistake can lead to a sequence of mistakes, e.g., say

- *word "it" has slightly higher chance at the beginning*
  - ↳ *e.g., "it":* 0.121 *and "I":* 0.119*, thus, we choose it as the first word*
- *since we chose "it", RNN needs to generate the next word accordingly*
  - ↳ *with "it" as input: "feels" has higher chances than "like"*

Basic Translator: *Inference via Decoding*

If we are unlucky, another token might be slightly higher in probability at $\ell = 1$



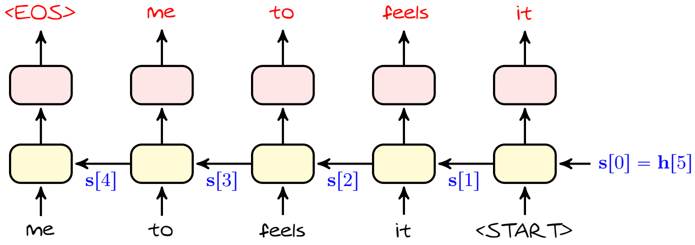
Decoding

To avoid error propagation, a better idea is to find the sequence with highest probability, i.e., sequence $\mathbf{v}^\star[1:L]$ *that has highest conditional probability*

$$\mathbf{v}^\star[1:L] = \underset{\mathbf{v}[1:L]}{\operatorname{argmax}}\, p\left(\mathbf{v}[1:L] | \mathbf{x}[1:T]\right)$$

Basic Translator: *Inference via Decoding*

If we are unlucky, another token might be slightly higher in probability at $\ell = 1$



Intuitively, this means: *we do not classify in each time*

- *We wait till the sentence is over*
- *We consider all possible combinations*
- *We find the one which has highest conditional probability*
  - ↳ *We need to compute this conditional probability*

Basic Translator: *Inference via Decoding*

Let's try finding this probability first

$$p\left(\mathbf{v}[1:L]|\mathbf{x}[1:T]\right) = p\left(\mathbf{v}[1:L]|\mathbf{h}[T]\right)$$

$$= \prod_{\ell=1}^{L} p\left(\mathbf{v}[\ell]|\mathbf{h}[T], \mathbf{v}[1:\ell-1]\right) \rightsquigarrow \text{Bayes chain rule}$$

$$\propto \prod_{\ell=1}^{L} p\left(\mathbf{v}[\ell]|\mathbf{s}[\ell-1], \mathbf{y}[\ell-1]\right) \propto \prod_{\ell=1}^{L} y_{v[\ell]}[\ell]$$

$y_{v[\ell]}[\ell]$ *means the entry of* $\mathbf{y}[\ell]$ *that corresponds to the class of* $\mathbf{v}[\ell]$, *e.g.,*

$$\mathbf{v}[\ell] = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \textit{and} \quad \mathbf{y}[\ell] = \begin{bmatrix} 0.1 \\ 0.3 \\ 0.4 \\ 0.2 \end{bmatrix} \rightsquigarrow y_{v[\ell]}[\ell] = 0.3$$

Basic Translator: *Inference via Decoding*

Since we are more comfortable with sum, we can use log-likelihood

$$\begin{aligned}\mathbf{v}^\star[1:L] &= \operatorname*{argmax}_{\mathbf{v}[1:L]} \log p\left(\mathbf{v}[1:L] | \mathbf{x}[1:T]\right) \\ &= \operatorname*{argmax}_{\mathbf{v}[1:L]} \sum_{\ell=1}^{L} \log y_{v[\ell]}[\ell]\end{aligned}$$

+ *But, is it feasible to find the sequence with highest sum?*
– No! *Since we deal here with an exponentially large case*

For a vocabulary with D words in it, the number of possible sequences is

$$(D+2)^L$$

Basic Translator: *Inference via Decoding*

In practice, however, we know that many of those combinations are invalid, e.g.,

"I are potato" is not an invalid English sentence!

So we can use sub-optimal search methods to find a good sequence

- *We do not necessarily hit the best sequence*
  - ↳ *But, for fair length, we can be close*
  - ↳ *We will be definitely better than naive instant decoding*
- *Most famous approach is beam search via top-k*
  - ↳ *At time step ℓ we find k sequences with highest sum log-likelihoods till ℓ*
  - ↳ *We update top-k sequences by searching among children on sequence tree*
  - ↳ *We finally select the sequence with highest probability among those top-k*
- *Since decoded sequences can be of different length, we usually maximize normalized log-likelihood*

$$\mathbf{v}^\star[1:L] = \operatorname*{argmax}_{\mathbf{v}[1:L]} \frac{1}{L} \sum_{\ell=1}^{L} \log y_{v[\ell]}[\ell]$$