Applied Deep Learning

Chapter 7: Sequence-to-Sequence Models

Ali Bereyhi

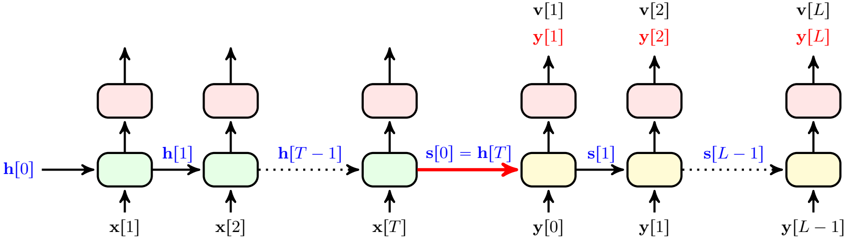
ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

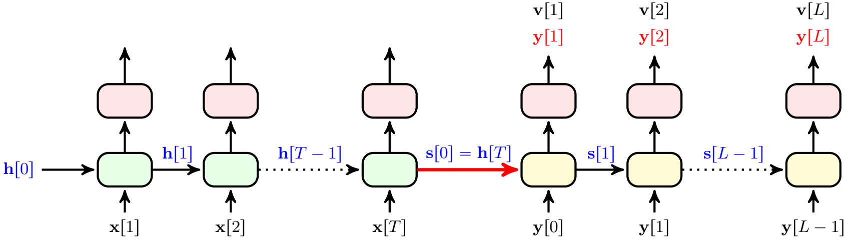
Translating Long Texts



With standard RNN encoder and decoder: *model works up to some length*

- *The decoder could get lost at some point*
  - ↳ *It could miss the case of the word or its order*
  - ↳ *Imagine it wants to translate:*
    "Ich habe den Apfel genommen, über den wir beim letzten Mal gesprochen haben" ⇝ *"I took the apple, about what we talked about last time"*

Information Bottleneck



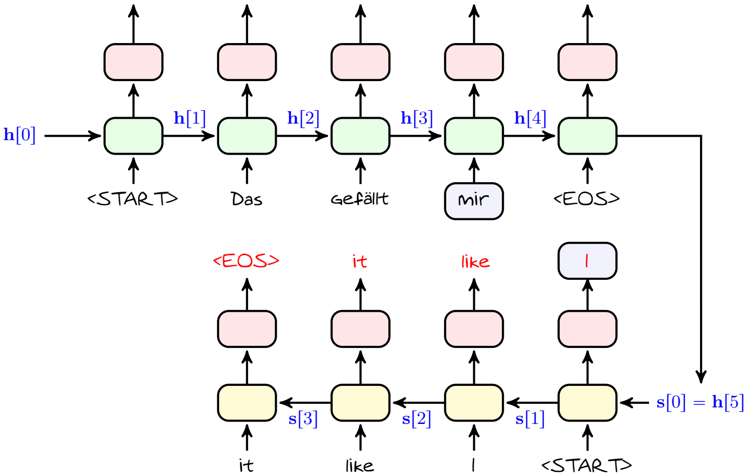
The source of this problem is that

decoder gets all its information through a single bottleneck

This problem is known as information bottleneck problem

Attention: *Finding Relevant Input*

Let's look back at our translator: "I" is given in translation because of "mir"



Attention: *Finding Relevant Input*

What if we could tell the decoder:

use hidden state of time $t = 4$ to generate its output word at time $t = 1$

We could intuitively say that in this case

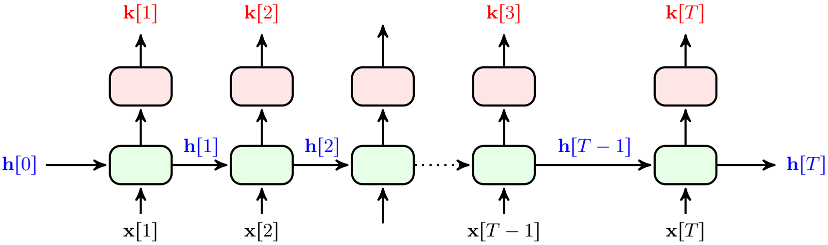
$$\mathbf{y}[1] = f\left(\mathbf{s}[0], \text{<START>}, \text{mir}\right)$$

which is more likely to be "I" as compared to the case in which

$$\mathbf{y}[1] = f\left(\mathbf{s}[0], \text{<START>}\right)$$

Attention mechanism formulates mathematically this intuitive idea

Attention: *Generating Keys*



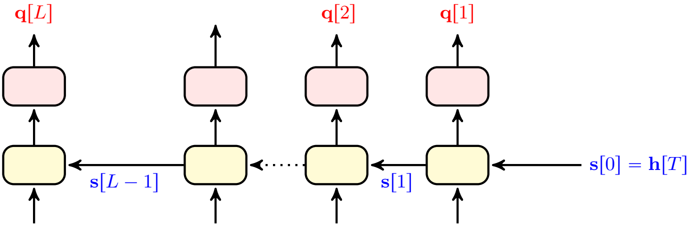
With attention, we generate a key for each time step at encoding

- *Keys are learned by an arbitrary layer that is fixed over time, e.g.,*

$$\mathbf{k}[t] = \sigma\left(\mathbf{W}_{\rm k}\mathbf{h}[t]\right)$$

- *We can even use multiple layer*
  ↳ *Not really needed in most applications*

Attention: *Generating Queries*



We next generate a query for each time step at decoding

- *Queries are again learned by an arbitrary layer, e.g.,*

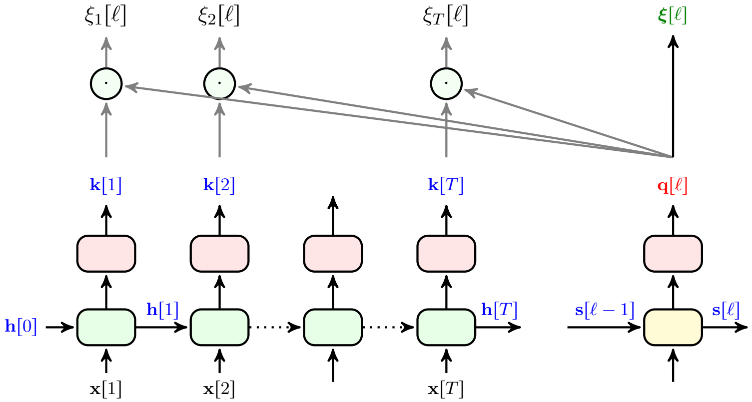
$$\mathbf{q}[t] = \sigma\left(\mathbf{W}_{\mathrm{q}}\mathbf{s}[t]\right)$$

- *It's in general a new learnable layer different from the key generator*

Attention: *Score at Time* ℓ

At decoder, we find score of query at time ℓ by comparing it to all keys

$$\boxed{\xi_t[\ell] = \mathbf{k}^{\mathsf{T}}[t]\mathbf{q}[\ell] \rightsquigarrow \boldsymbol{\xi}[\ell] = [\xi_1[\ell], \ldots, \xi_T[\ell]]}$$



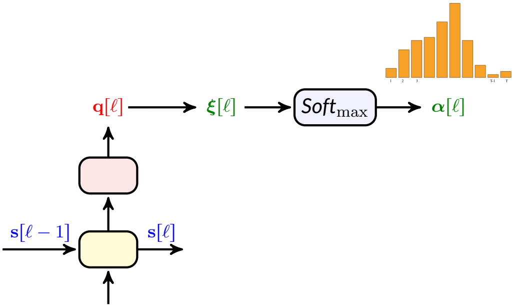
Attention: *Attention Weight at Time ℓ*

We can pass the scores through softmax to find

$$\textit{chance of } \mathbf{v}[\ell] \textit{ being related to input entry } \mathbf{x}[t] \equiv \alpha_t[\ell]$$

Softmax gives us

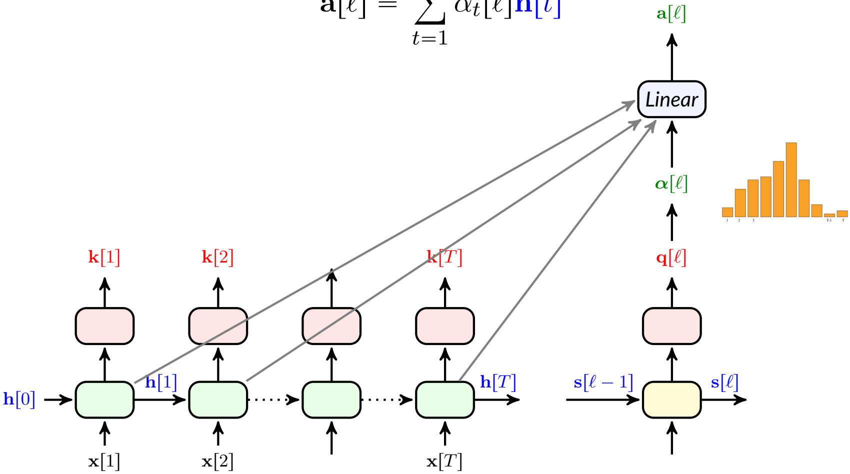
$$\boldsymbol{\alpha}[\ell] = \mathrm{Soft}_{\max}\left(\boldsymbol{\xi}[\ell]\right) = [\alpha_1[\ell], \ldots, \alpha_T[\ell]]$$



Attention: *Attention Feature*

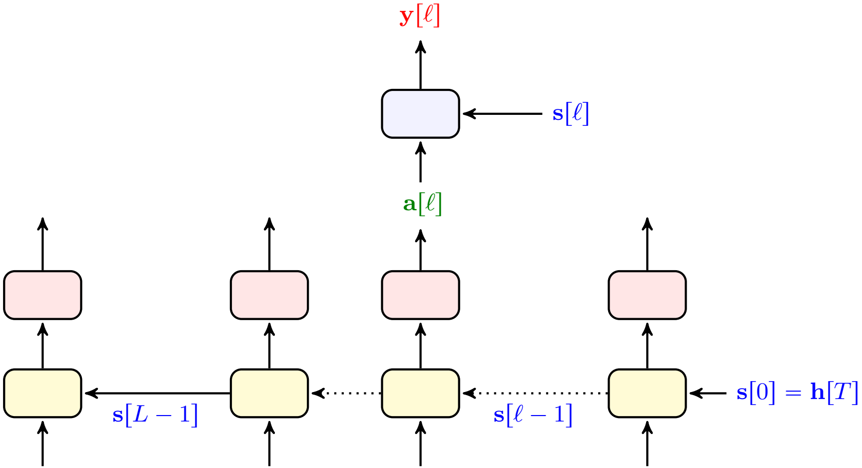
We now use these weights to make a vector of attention features

$$\mathbf{a}[\ell] = \sum_{t=1}^{T} \alpha_t[\ell] \mathbf{h}[t]$$



Attention: *Computing Output*

We now use attention features to build the output sequence



Attention: *Computing Output*

We use attention features to build the output sequence

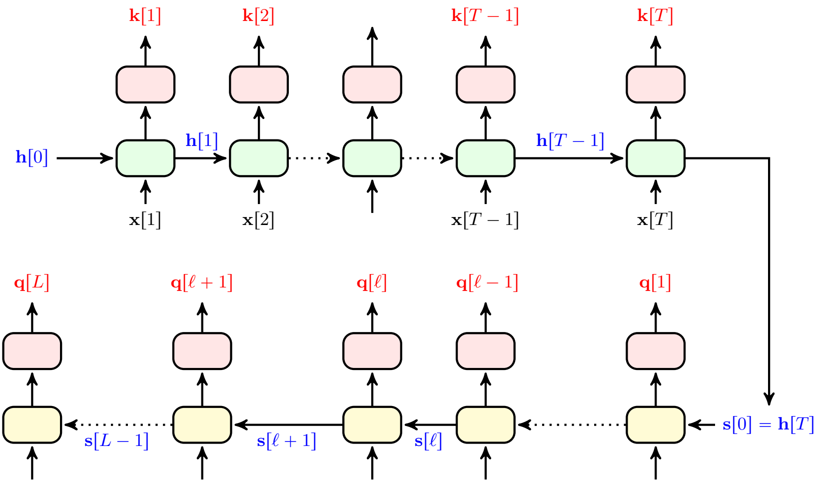
- *This can be any layer, as in standard RNN, e.g.,*

$$\mathbf{y}[\ell] = \mathit{Soft}_{\max}\left(\mathbf{W}_{\rm out}\mathbf{s}[\ell] + \mathbf{W}_{\rm att}\mathbf{a}[\ell]\right)$$

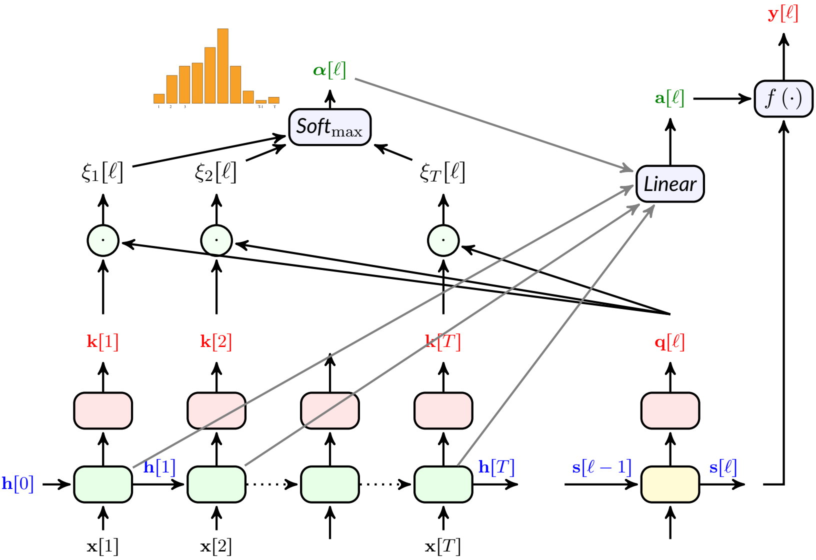
- *We could also use $\mathbf{a}[t]$ as a new state*
  - ↳ *We could combine it with the current state*
  - ↳ *We can pass it to higher layer*

It turns out that attention can hugely help in practice!

Attention: *End to End Architecture*



Attention: *End to End Architecture*



Attention: *Training*

Let's look at training: *assume we want to train it over a single pair of sequences*

- *We have a sequence of inputs* $\mathbf{x}[t]$
  - ↳ *For instance a German sentence*
- *We have a sequence of labels* $\mathbf{v}[\ell]$
  - ↳ *For instance the English translation*
  - ↳ *We can compare each output* $\mathbf{y}[\ell]$ *with its label*

We start with forward pass

- *Pass forward through the encoder*
  - ↳ *Also generate the keys*
- *Pass the encoder's state and its keys to the decoder*
- *Pass forward through the decoder*
  - ↳ *Generate queries and compare them to the keys*
  - ↳ *Compute attention and the outputs*
- *Compute loss by aggregating* $\mathcal{L}\left(\mathbf{y}[\ell], \mathbf{v}[\ell]\right)$

Attention: *Training*

Now we should pass backward

- *Pass backward through the output layer*
  - ↳ *Compute* $\nabla_{\mathbf{a}[\ell]} \hat{R}[\ell]$ *and* $\nabla_{\mathbf{s}[\ell]} \hat{R}[\ell]$
- *Pass backward through the attention layer*
  - ↳ *Compute* $\nabla_{\boldsymbol{\alpha}[\ell]} \hat{R}[\ell]$, $\nabla_{\boldsymbol{\xi}[\ell]} \hat{R}[\ell]$, $\nabla_{\mathbf{k}[\ell]} \hat{R}[\ell]$ *and* $\nabla_{\mathbf{q}[\ell]} \hat{R}[\ell]$
- *Pass backward through time at the decoder*

$$\nabla_{\mathbf{s}[\ell-1]} \hat{R}[\ell] = \nabla_{\mathbf{s}[\ell]} \hat{R}[\ell] \circ \nabla_{\mathbf{s}[\ell-1]} \mathbf{s}[\ell] + \nabla_{\mathbf{q}[\ell]} \hat{R}[\ell] \circ \nabla_{\mathbf{s}[\ell-1]} \mathbf{q}[\ell]$$

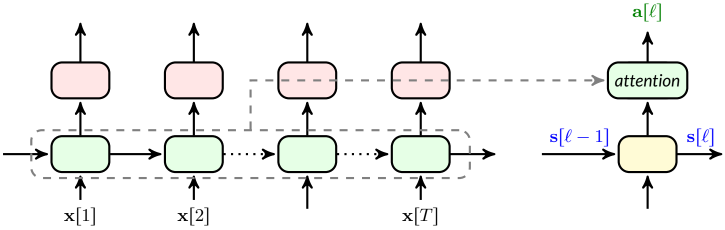
- *Pass backward through time at the encoder*

$$\begin{aligned}\nabla_{\mathbf{h}[t-1]} \hat{R}[\ell] = \nabla_{\mathbf{h}[t]} \hat{R}[\ell] \circ \nabla_{\mathbf{h}[t-1]} \mathbf{h}[t] + \nabla_{\mathbf{k}[t]} \hat{R}[\ell] \circ \nabla_{\mathbf{h}[t-1]} \mathbf{q}[t] \\ + \nabla_{\mathbf{a}[\ell]} \hat{R}[\ell] \circ \nabla_{\mathbf{h}[t-1]} \mathbf{a}[\ell]\end{aligned}$$

- *Aggregate over* ℓ, *update all weights and go for the next round*

Attention as a Layer

We can look at the whole attention mechanism as a layer



This comes in handy once we want to look at self-attention