50.039 – Theory and practice of deep learning

Alex

Week 09: Attention Models

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

key idea for graded knowledge

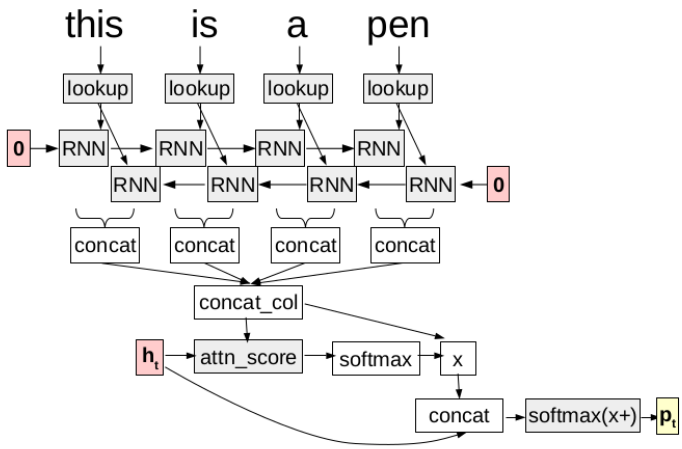
- attention models as learnable weights for partitions of a feature
- on what attention can be used in sequence encoders
- on what attention can be used in computer vision
- two example attention models
- Temporal convolutional NNs

1 Neural attention models

This paper is out of class. One of the most read papers is:

Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, arxiv.org `https://arxiv.org/abs/1409.0473`

Typical graph-based explanations can be somewhat confusing:

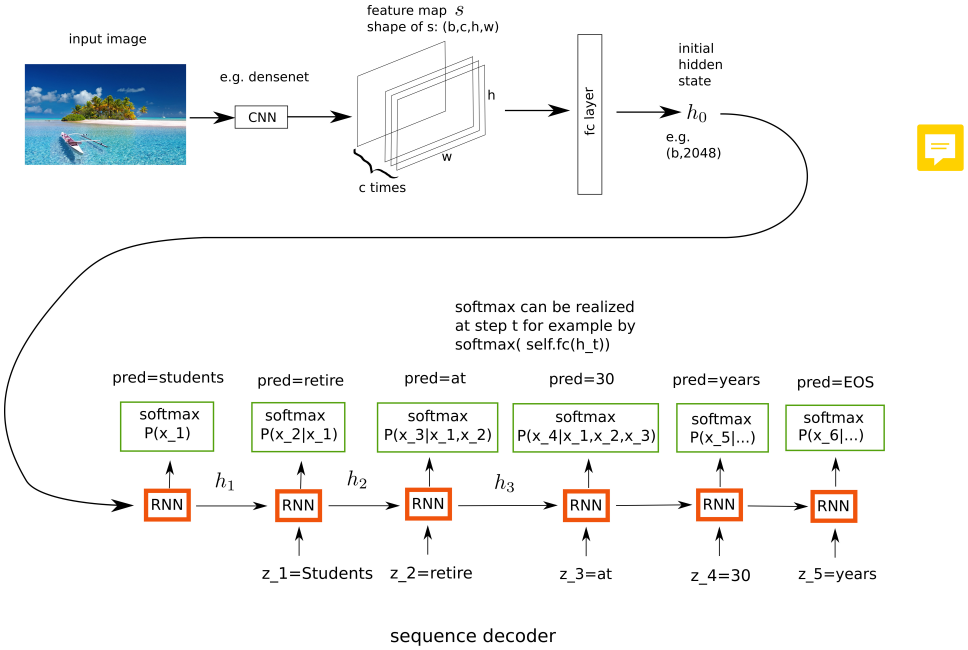


Lets go for a bit different explanation. We will have two exemplary uses cases. Lets recap how models for this can look like.

- image captioning
- sequence-to-sequence models

1.1 image captioning

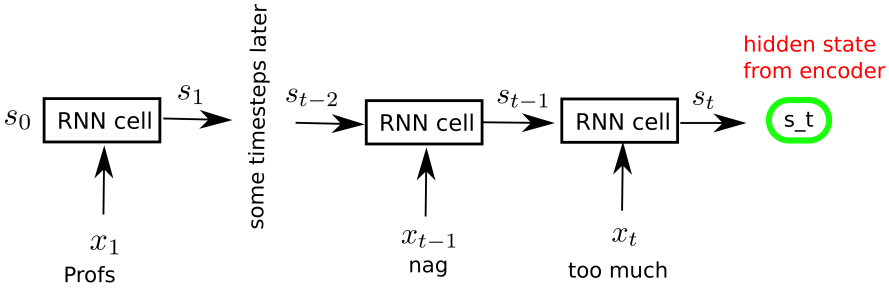
- Input: Image
- Output: Sequence of Words
- loss: a sum over output probabilities for every element in the sequence
- a possible model ? draw one!

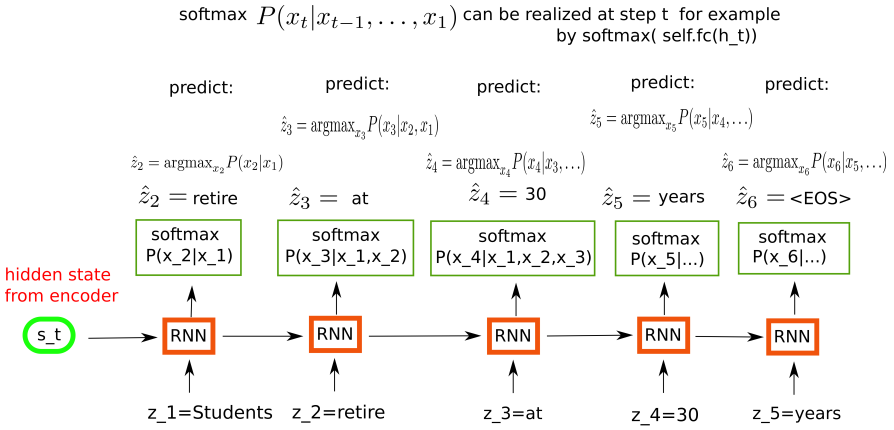


1.2 sequence to sequence

Example: summarize a text. Translate a text

- Input: Sequence of Words
- Output: Sequence of Words
- loss: a sum over output probabilities for every element in the sequence
- a possible model ?





1.3 towards attention

What is attention in a nutshell ?

- take some intermediate feature s, partition it: $s = (s_1, \ldots, s_R)$
- for above use cases what could be partitioned and how ?
- compute weights $w_i^{(t)}$ for each element. These weights are different in every decoder step t (when the hidden state h_t is used to produce an output z_t), therefore the index $^{(t)}$
- use weighted feature $r^{(t)} = \sum_i w_i^{(t)} s_i$ to help in decoding task / prediction task. How ?
  - augment prediction model, use a concatenation $[r^{(t)}, h_t]$ instead of only h_t
- How to compute the weights $w_i^{(t)}$?

$$(w_1^{(t)}, \ldots, w_i^{(t)}, \ldots w_R^{(t)}) = f(h_t, s, x)$$

  - They should depend on the current decoder hidden state h_t, and on the intermediate feature s and/or on the inputs x, by some function f
  - They should be weights
  - they should contrain trainable parameters and thus be learned (gradient descent) during training
  - what are simple ideas to realize such weights with trainable parameters? suppose $h_t \in \mathbb{R}^d$, $s = (s_1, \ldots, s_R)$, $s_i \in \mathbb{R}^e$
- can use attention whenever one can partition an feature s into subsets $s = (s_1, \ldots, s_R)$
- introduces more parameters – works when you have sufficient data!

Some attention models, differing in parametrization
`http://ruder.io/deep-learning-nlp-best-practices/index.html#attention`
Let be h_t the current hidden state, s_j a feature map part to compute a weight for, then:

$$\begin{aligned}(w_1^{(t)}, \ldots, w_i^{(t)}, \ldots w_R^{(t)}) &= f(h_t, s, x)\\ w_j^{(t)} &= v^\top \cdot tanh(W \cdot [h_t, s_j]) \in \mathbb{R}^1\\ W, v \text{ trainable, } W &\in \mathbb{R}^{k \times (d_1 + d_2)}, v \in \mathbb{R}^{k \times 1}\end{aligned}$$

This is from Bahdanau 2015, a neural network with one hidden layer, and two weights layers. First layer: $[h_t, s_j] \mapsto tanh(W[h_t, s_j])$, then on top a linear weighting.

Another mechanism:

$$\begin{aligned}(w_1^{(t)}, \ldots, w_i^{(t)}, \ldots w_R^{(t)}) &= f(h_t, s, x)\\ w_j^{(t)} &= h_t^\top \cdot W \cdot s_j \in \mathbb{R}^1,\ W \text{ trainable, } W \in \mathbb{R}^{d_1 \times d_2}\end{aligned}$$

1.4 Attention Models for Vision and any types of inputs

- Images: we know, in higher feature layers the feature map is something like

  $$(batchsize, channels, height, width)$$

- Naturally we can replace one encoder state s_i by one spatial element of that feature map

  $$s_{ik} = (batchsize, channels, height = i, width = k),$$

  now indexed by the spatial coordinates i, k. Same formulas as above apply. So, an element in the set is:

  $$s_{ik} = (batchsize, channels, height = i, width = k)$$

- now apply attention not to get weights for encoder states, but to get weights for spatial regions s_{ik} of feature maps :)

  $$c_t = \sum_{i,k} \alpha_{t,ik} s_{ik}$$

- all what one need is: partition your input into elements, compute one feature per element. That allows to define attention weights for each time step t and each element.

an example for a nice well written paper: `https://arxiv.org/abs/1704.03162`

(**out of class**) : A more complex attention mechanism is eq (9) in `https://arxiv.org/pdf/1805.07932.pdf`, `https://visualqa.org/static/slides/2018_workshop_jin-hwa_slides.pdf` – but the paper is not very well written. Suppose you have the i-th word encoded as $x_i \in \mathbb{R}^N$ and the j-th element of a image feature map as $y_j \in \mathbb{R}^M$. Then you can project them by a trainable projections $U \in \mathbb{R}^{K \times N}, V \in \mathbb{R}^{K \times M}$ into a common embedding space

$$\begin{aligned} x_i \mapsto ex_i = U^T x_i & \qquad \in \mathbb{R}^K \\ y_j \mapsto ey_j = V^T y_j & \qquad \in \mathbb{R}^K \end{aligned}$$

Now both are in the same space and you can element-wise multiply them

$$ex_i \circ ey_j \in \mathbb{R}^K$$

Finally you can take an inner product with some vector p, to obtain an attention logit for the pair of word x_i and y_j.

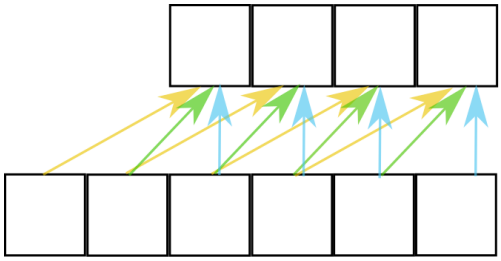
$$\alpha_{i,j} = p \cdot (ex_i \circ ey_j) \in \mathbb{R}^1$$

What needs to be done, is to take all those logits $\alpha_{i,j}, i = 1, \ldots, C, j = 1, \ldots, D$ and turn them by a softmax into a true attention weight, which sums up over all $(i, j), i = 1, \ldots, C, j = 1, \ldots, D$ to 1.

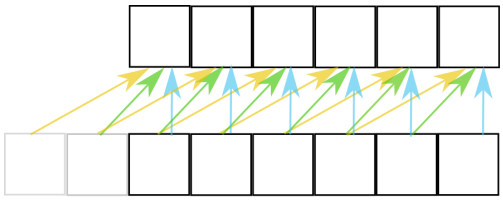
2 TCN

github code in pytorch: `https://github.com/locuslab/TCN`
a third recurrent NN architecture

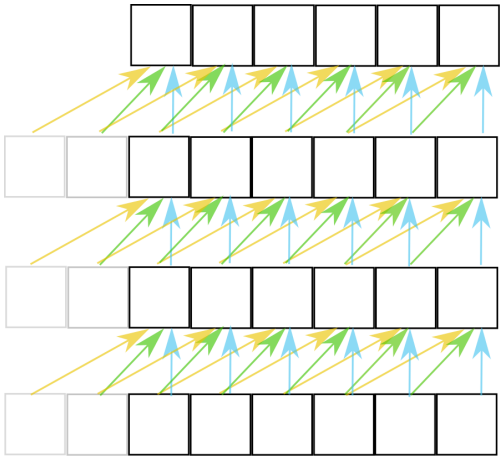
- LSTM
- GRU
- TCN
- Basic idea: use 1d-cnn but only causal connections from one layer to the next, that is no connections to the next layer from the future of the layer below



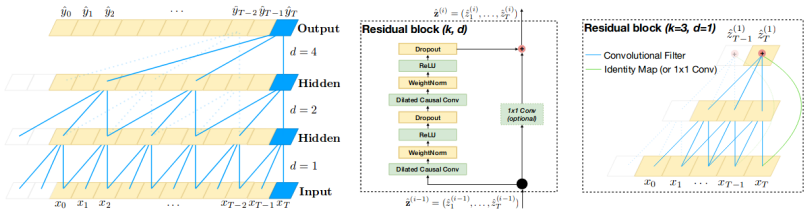
- In order to have same property as RNN: input length = output length, pad each layer with kernelsize − 1 at the start



- stack multiple layers



- use larger strides in higher layers (see subfigure a for stride 1,2,4) – to look at larger time scales without the need to stack too much!



(a) (b) (c)

Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

- use residual connections across layers that jump up 2 levels, see subfigure b