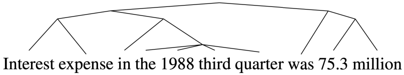
Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks[1]

May 21, 2020

[1] "Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks" by Shen et al.

Overview

- Human language has a tree-like structure



- Human brain can infer it without any parse trees supervision
- How can we force neural networks to acquire this structure without supervision as well?
- Authors propose a specific inductive bias to do that:
  - divide a hidden state in a LSTM cell into chunks, each chunk keeps a hidden state for its own tree level
  - implement such a mechanism that can erase higher-level nodes only when all lower-level nodes are erased
  - this mechanism works by making forget/inputs gates of higher level nodes be dependent on the corresponding gates of lower-level ones
- They obtained good results for several tasks

Vanilla LSTM

LSTM cell at timestep t takes an input x_t, updates its cell state c_t and hidden state h_t in the following way:

1. Compute the following quantities:
   1.1 $f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right)$ — "forget gate" to erase c_{t-1};
   1.2 $i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right)$ — "input gate" to erase new *raw* cell state $\hat{c}_t$
   1.3 $o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right)$ — "output gate" to erase new hidden state h_t
2. Then compute the following quantities
   2.1 $\hat{c}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right)$ — new "raw" cell state
   2.2 $c_t = f_t \circ c_{t-1} + i_t \circ \hat{c}_t$ — new cell state
   2.3 $h_t = o_t \circ \tanh\left(c_t\right)$ — new hidden state

Authors change the gating mechanism in such a way that erasement now occurs under some hierarchy.

ON-LSTM (1/3)

- Imagine that we have some arbitrary tree and we observe only leaves (i.e. our model takes leaves as inputs one by one)
- At each timestep, we want to keep information not only about a leaf node, but also about every node on the path up to the root
- Let's store the information about each node of a path in a separate chunk of our state vector:

$$\boldsymbol{v} = [\underbrace{v_1^{(0)}, v_2^{(0)}, ..., v_m^{(0)}}_{\text{leaf state}}, \underbrace{v_1^{(1)}, v_2^{(1)}, ..., v_m^{(1)}}_{\text{leaf's parent state}}, ..., \underbrace{v_1^{(L)}, v_2^{(L)}, ..., v_m^{(L)}}_{\text{root state}},]$$

- We know that erasing is the core operation in LSTM to update its state
- So now we want a mechanism that would erase a higher level state only if all lower-level states are erased as well

ON-LSTM (2/3)

- To implement the desired mechanism, our erasement vector $\boldsymbol{g}$ for $\boldsymbol{v}$ should have the following structure:

$$g = [0, 0, 0, 0, ..., 1, 1, 1],$$

  i.e. we have a segment of zeros followed by a segment of ones.
- The idea is based on a novel activation function cumax:

$$\text{cumax}(...) = \text{cumsum}(\text{softmax}(...))$$

  (it operates on vectors instead of scalars).
- Let d be a *split point*, i.e. a number when the first "1" occured.
- Then d follows categorical distribution and we can model it with softmax:

$$p(d) = \text{softmax}(\hat{\boldsymbol{d}})$$

ON-LSTM (3/4)

- Using d we can compute $p(g_k = 1)$, i.e. a probability of k-th element of g being equal to 1:

$$p(g_k = 1) = p(d \leq k) = \sum_{i \leq k} p(d = i)$$

- To compute a vector of probabilities $[p(g_1 = 1), ..., p(g_n = 1)]$ we need to apply cumax on $\hat{\boldsymbol{d}}$:

$$p(\boldsymbol{g}) = \text{cumax}(\hat{\boldsymbol{d}})$$

- So we use a continuous relaxation for erasement operation (like in a vanilla LSTM) and erase a part of $\boldsymbol{v}$ by computing

$$p(\boldsymbol{g}) \circ \boldsymbol{v}$$

- In this way, higher level segments are erased only if all the lower-level ones are erased

ON-LSTM (4/5)

Imagine that at the current timestep we are at some level ℓ, i.e.:

- we are going to “close” the branch of height ℓ
- we are going to erase the state below it
- we are going to keep all the higher-level context

ON-LSTM (5/6)

First we should compute the master gates:

1. $\tilde{f}_t = \operatorname{cumax}\left(W_{\tilde{f}} x_t + U_{\tilde{f}} h_{t-1} + b_{\tilde{f}}\right)$ — a mask, which discards all lower-level elements
2. $\tilde{i}_t = 1 - \operatorname{cumax}\left(W_{\tilde{i}} x_t + U_{\tilde{i}} h_{t-1} + b_{\tilde{i}}\right)$ — a mask which keeps only lower-level elements
3. Compute the mask for the current-level segment $\omega_t = \tilde{f}_t \circ \tilde{i}_t$.

These vectors would look like this (ideally):

$$\begin{aligned} \tilde{f}_t &= [0, 0, 0, 0, 0, 1, 1, 1, 1] \\ \tilde{i}_t &= [1, 1, 1, 1, 1, 1, 1, 0, 0] \\ \omega_t &= [0, 0, 0, 0, 0, 1, 1, 0, 0] \end{aligned} \tag{1}$$

ON-LSTM (6/6)

Then we update the states the following way:

3. Compute old element-wise forget and inputs gates f_t, i_t and new raw cell state $\hat{c}_t$ without changes.
4. $\hat{f}_t = f_t \circ \omega_t + (\tilde{f}_t - \omega_t)$ — forget gate that affects current level ℓ, erases all lower-level elements and keeps all higher-level elements
5. $\hat{i}_t = i_t \circ \omega_t + (\tilde{i}_t - \omega_t)$ — forget gate that affects current level ℓ, erases all higher-level elements and keeps all lower-level elements
6. $c_t = \hat{f}_t \circ c_{t-1} + \hat{i}_t \circ \hat{c}_t$ — new cell state
7. h_t is computed without changes

In this way we have updated only those part of the hidden state which corresponds to some specific level of a tree.

Conclusion

- LSTM still performs better for short-term dependencies
- We repeat each dimension C times, before the element-wise multiplication
- It is interesting to see that there is room for exploration even in such an extensively explored model as LSTM
- Can we bring the idea into Transformers? Graph NNs?