A Preparation steps before read and write in neural Turing machine

Both the read and write actions contains two preparation steps before the actual interaction with the memory: 1) an analysis step to the controller state and 2) an addressing step. In the controller state analysis step, first, the specific parameters for guiding the read and write are extracted from the controller state:

$$\begin{aligned}
[\mathbf{k}^r, \beta^r, g^r, \mathbf{s}^r, \gamma^r]^T &= W_c^r \mathbf{o}_t^{(c)} + \mathbf{b}_c^r \\
[\mathbf{k}^w, \beta^w, g^w, \mathbf{s}^w, \gamma^w, \mathbf{e}, \mathbf{a}]^T &= W_c^w \mathbf{o}_t^{(c)} + \mathbf{b}_c^w,
\end{aligned} \tag{11}$$

where $\mathbf{s} \in \mathbb{R}^3$, $\mathbf{k} \in \mathbb{R}^M$, $\mathbf{e} \in \mathbb{R}^M$, $\mathbf{a} \in \mathbb{R}^M$, all the other analyzed parameters are scalars, and the superscripts r and w are noted for read and write separately; and then the parameters are preprocessed:

$$\begin{aligned}
\beta &:= \mathrm{softplus}(\beta) \\
g &:= \mathrm{sigmoid}(g) \\
\mathbf{s} &:= \mathrm{softmax}(\mathbf{s}) \\
\gamma &:= 1 + \mathrm{softplus}(\gamma)
\end{aligned} \tag{12}$$

Next comes the addressing step. The attending position $\mathbf{w}$ for a specific memory cell is computed successively by content addressing to get $\mathbf{w_c}$, interpolation with last time step to get $\mathbf{w_g}$, shift operation to get $\mathbf{w_w}$ and sharpening operation to get the final $\mathbf{w}$:

$$\begin{aligned}
\mathbf{w}_c[i] &= \frac{\exp(\beta R[\mathbf{k}, M_t[i]])}{\sum_j \exp(\beta R[\mathbf{k}, M_t[j]])}, i = 0, 1, \cdots, M \\
\mathbf{w}_g &= g \cdot \mathbf{w}_c + (1-g) \cdot \mathbf{w}_{prev} \\
\mathbf{w}_w[i] &= \sum_{j=0}^{N-1} \mathbf{w}_g[j] \mathbf{s}[i-j], i = 0, 1, \cdots, M \\
\mathbf{w}[i] &= \frac{\mathbf{w}_w[i]^\gamma}{\sum_j \mathbf{w}_w[j]^\gamma}
\end{aligned} \tag{13}$$

where the $R[\cdot, \cdot]$ is cosine similarity. Note that, for simplicity, we leave out the time notation t.

B Deterministic algorithm for evaluating samples in *M10AE*

The evaluation result of examples in *M10AE* can be computed by a deterministic algorithm in a shift-reduce style with a stack. As shown in Algorithm 1, for each time step t, an input x_t and its next x_t are received, and x_t is pushed onto the stack S. After that push, the stack is reduced (shown in Algorithm 2) whenever it is reducible (shown in Algorithm 3).

Algorithm 1 Deterministic algorithm for evaluating samples in *M10AE*

input: arithmetic expression $e = x_0, x_1, \cdots, x_L$
output: the evaluation result of e

1: empty stack $S = [0, 0, \cdots, 0]^T \in \mathbb{R}^N$
2: **for** each input x_t and its next x_{t+1} **do**
3: push x_t
4: **while** REDUCIBLE(S, x_{t+1}) **do**
5: $r \leftarrow$ REDUCE(S)
6: push r
7: **end while**
8: **end forreturn** l_0

Algorithm 2 Reduction conditions

```
function REDUCIBLE(S, x)
    if S[0] is an operator and S[1] is a numeral then
        return TRUE
    else if S[0] is a numeral and s[1][0] is a numeral and S[1][1] is an operator then
        if S[1][1] ∈ {+, −} and x ∉ {∗, /} then
            return TRUE
        end if
        if S[1][1] ∈ {∗, /} then
            return TRUE
        end if
        return FALSE
    else if S[0] is ')' and S[1][0] is '(' and S[1][1] is a numeral then
        return TRUE
    else if S[0] is a numeral and S[1][1] is '(' and x is ')' then
        return TRUE
    end if
end function
```

Algorithm 3 Reduce

```
function REDUCE(S)
    if S[0] is an operator and S[1] is a numeral then
        return (S[1], S[0])
    else if S[0] is a numeral and s[1][0] is a numeral and S[1][1] is an operator then
        return eval(S[1][1], S[1][0], S[0])
    else if S[0] is a numeral and S[1] is '(' then
        return (S[1], S[0])
    end if
end function
```