Quasi-Recurrent Neural Networks

input-dependent pooling + gated linear combination of convolution features.

Model

- Parallel computation across both timestep and minibatch dimensions, enabling high throughput and good scaling to long sequences.
- Allow the output to depend on the overall order of elements in the sequences.

Each layer of Quasi-RNN consists of two kinds of subcomponents:

1. the convolutional components
   - ***fully parallel computation*** across minibatches and spatial dimensions (sequence dimension)
2. the pooling components
   - ***fully parallel computation*** across minibatches and feature dimensions.

Equations

- Input $\mathbf{X} \in \mathcal{R}^{T \times n}$.
- T: number of time step.
- n: hidden dimension.
- $*$: masked convolition along time dimension

The convolution part

$$\mathbf{Z} = \tanh(\mathbf{W} * \mathbf{X})$$

$$\mathbf{F} = \sigma(\mathbf{W}_f * \mathbf{X})$$

$$\mathbf{O} = \sigma(\mathbf{W}_o * \mathbf{X})$$

The pooling part

1. f-pooling: ***only forget gate***

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{f}_t) \odot \mathbf{z}_t$$

2. fo-pooling: ***forget gate and output gate***

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (\mathbf{1} - \mathbf{f}_t) \odot \mathbf{z}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t$$

3. ifo-pooling: ***forget gate, input gate and output gate***

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{z}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t$$

The recurrent parts must be calculated for each timestep in sequence.

Variants

1. Dropout
   - choose a new subset of channels to "zone out" at each time step
   - for these chosen channels, the network copies states from one timestep to the next ***without modification***.
   - modify the forget gates. The pooling function itself does not need to modify.

$$\mathbf{F} = \mathbf{1} - \text{dropout}(1 - \sigma(\mathbf{W}_f * \mathbf{X}))$$

2. Densely-connected layers
   - for sequence classification tasks, the author found it is helpful to use skip connections between ***every QRCNN***.
   - add connections between embeddings and every QRCNN layer, and between every pair of QRCNN layers.
     - ***concatenate*** each QRCNN's input to its output along the channel dimension before feeding the state to the next layer
3. Encoder-Decoder models
   - simply feeding the last encoder hidden state would not allow the encoder state to affect the gate or update values that are provided to the decoder's pooling layer.
   - ***how to fix***
     - for the l-th decoder QRNN layer, outputs of its convolution functions is added with a linearly projected copy of the l-th encoder's last encoder state:

$$\mathbf{Z}^l = \tanh(\mathbf{W}_z^l * \mathbf{X}^l + \mathbf{V}_z^l \tilde{\mathbf{h}}_T^l)$$

$$\mathbf{F}^l = \tanh(\mathbf{W}_f^l * \mathbf{X}^l + \mathbf{V}_f^l \tilde{\mathbf{h}}_T^l)$$

$$\mathbf{O}^l = \tanh(\mathbf{W}_o^l * \mathbf{X}^l + \mathbf{V}_o^l \tilde{\mathbf{h}}_T^l)$$

- attention, in the below equations, L is the last layer.

$$\alpha_{st} = \text{softmax}(\mathbf{c}_t^L \cdot \tilde{\mathbf{h}}_s^L)$$

$$\mathbf{k}_t = \sum_{\alpha} \alpha_{st} \tilde{\mathbf{h}}_s^L$$

$$\mathbf{h_t}^L = \mathbf{o}_t \odot (\mathbf{W}_k \mathbf{k}_t + \mathbf{W}_c \mathbf{c}_t^L)$$

  - use dot products of encoder hidden states with the decoder's last layer's ***un-gated*** hidden states.