Personal Proceeding on Time Series (4)

--PLSTM implement and Fast weights

(Mar 29, 2017)

YANG Jiancheng

Outline

- I. How to implement PLSTM
- II. Using Fast Weights to Attend to the Recent Past

• I. How to implement PLSTM

- **Review**

$$\phi_t = \frac{(t - s) \bmod \tau}{\tau}, \qquad k_t = \begin{cases} \frac{2\phi_t}{r_{on}}, & \text{if } \phi_t < \frac{1}{2} r_{on} \\ 2 - \frac{2\phi_t}{r_{on}}, & \text{if } \frac{1}{2} r_{on} < \phi_t < r_{on} \\ \alpha \phi_t, & \text{otherwise} \end{cases}$$

$$\widetilde{c_j} = f_j \odot c_{j-1} + i_j \odot \sigma_c(x_j W_{xc} + h_{j-1} W_{hc} + b_c) \tag{7}$$

$$c_j = k_j \odot \widetilde{c_j} + (1 - k_j) \odot c_{j-1} \tag{8}$$

$$\widetilde{h_j} = o_j \odot \sigma_h(\widetilde{c_j}) \tag{9}$$

$$h_j = k_j \odot \widetilde{h_j} + (1 - k_j) \odot h_{j-1} \tag{10}$$

• I. How to implement PLSTM

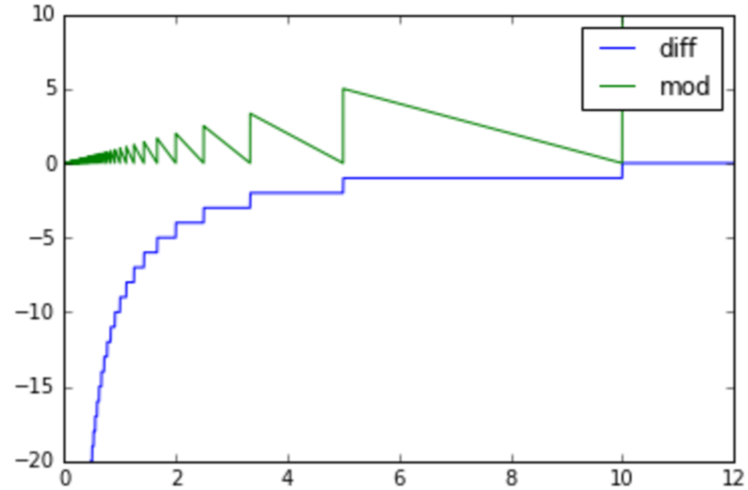
- **How to implement MOD: register gradient**

$$f(x,y) = x\%y = x - y\left\lfloor\frac{x}{y}\right\rfloor$$

$$\frac{\partial f}{\partial x} = 1$$

$$\frac{\partial f}{\partial y} = -\left\lfloor\frac{x}{y}\right\rfloor - y \times \left(\frac{\partial}{\partial y}\left\lfloor\frac{x}{y}\right\rfloor\right)$$

$$\frac{\partial f}{\partial y} = \begin{cases} -\left\lfloor\frac{x}{y}\right\rfloor, & x\%y \neq 0 \\ undefined, & x\%y = 0 \end{cases}$$




```python
# Here we need to register the gradient for the mod operation
@ops.RegisterGradient("FloorMod")
def _mod_grad(op, grad):
    x, y = op.inputs
    gz = grad
    x_grad = gz
    y_grad = tf.reduce_mean(-(x // y) * gz, axis=[0], keep_dims=True)
    return x_grad, y_grad
```

• I. How to implement PLSTM

- **How to implement MOD: write explicitly**

$$f(x,y) = x\%y = x - y\left\lfloor \frac{x}{y} \right\rfloor$$

```
# modulo operation not implemented in Tensorflow backend, so write explicitly.
# a mod n = a - (n * int(a/n))
# phi = ((t - shift) % period) / period
phi = ((t - shift) - (period * ((t - shift) // period))) / period
```

• I. How to implement PLSTM

- **How to implement *if***

a) Use ***where*** or ***switch*** function

```python
        # calculate kronos gate
        phi = tf.div(tf.mod(tf.mod(times - s_broadcast, tau_broadcast) + tau_broadcast, tau_broadcast),
                     tau_broadcast)
        is_up = tf.less(phi, (r_on_broadcast * 0.5))
        is_down = tf.logical_and(tf.less(phi, r_on_broadcast), tf.logical_not(is_up))

        k = tf.where(is_up, phi / (r_on_broadcast * 0.5),
                     tf.where(is_down, 2. - 2. * (phi / r_on_broadcast), self.alpha * phi))
```

b) Write explicitly

```python
    # K.switch not consistent between Theano and Tensorflow backend, so write explicitly.
    up = K.cast(K.less_equal(phi, r_on * 0.5), K.floatx()) * 2 * phi / r_on
    mid = K.cast(K.less_equal(phi, r_on), K.floatx()) * \
          K.cast(K.greater(phi, r_on * 0.5), K.floatx()) * (2 - (2 * phi / r_on))
    end = K.cast(K.greater(phi, r_on), K.floatx()) * self.alpha * phi
    k = up + mid + end
```

• II. Fast Weights

- **Intuition**

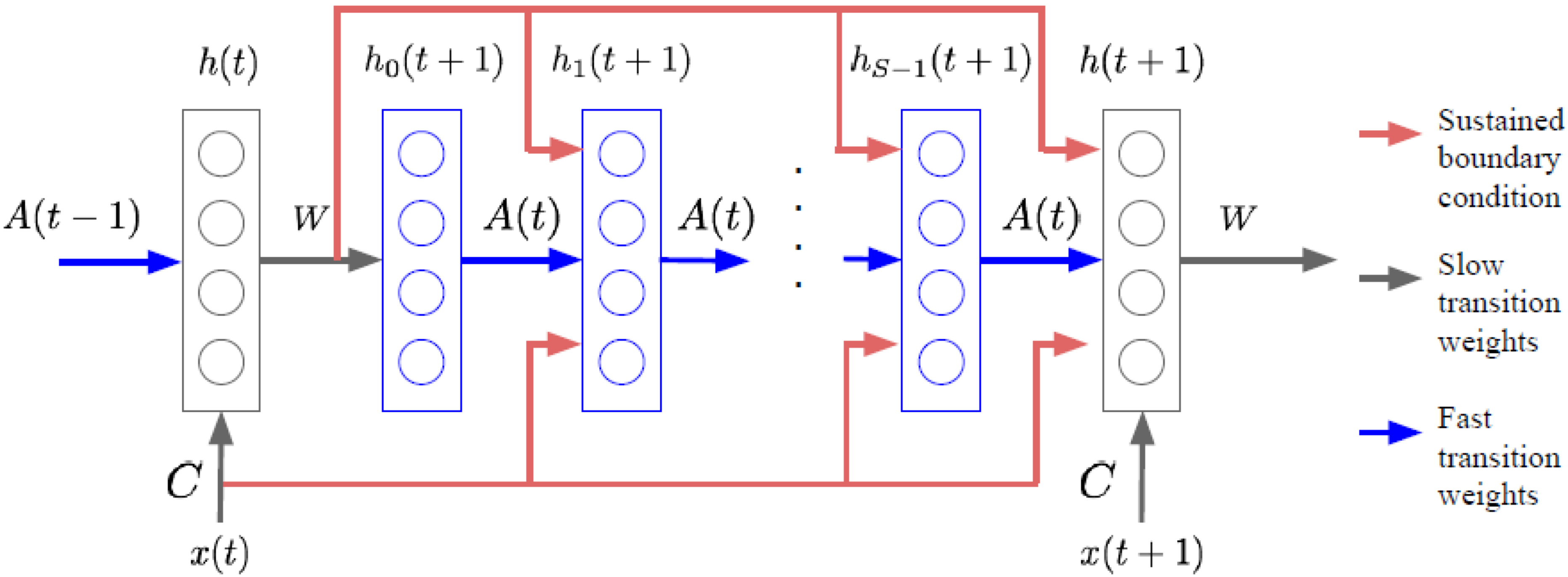


Figure 1: The fast associative memory model.

Slow weights < Fast weights < Activity

***Distributed* Memory < Intermediate Memory < Stateless**

• II. Fast Weights

- **Easy Formula means trivial implementation**

$$A(t+1) = \lambda A(t) + \eta h(t)h(t)^T \qquad (1)$$

$$h_{s+1}(t+1) = f([Wh(t) + Cx(t)] + A(t)h_s(t+1)), \qquad (2)$$

A Fast trick

$$A(t) = \eta \sum_{\tau=1}^{\tau=t} \lambda^{t-\tau} h(\tau)h(\tau)^T \qquad (3)$$

$$A(t)h_{s+1}(t+1) = \eta \sum_{\tau=1}^{\tau=t} \lambda^{t-\tau} h(\tau)[h(\tau)^T h_s(t+1)] \qquad (4)$$

• II. Fast Weights

- **Experiment**

Input string	Target
c9k8j3f1??c	9
j0a5s5z2??a	5

Model	R=20	R=50	R=100
IRNN	62.11%	60.23%	0.34%
LSTM	60.81%	1.85%	0%
A-LSTM	60.13%	1.62%	0%
Fast weights	1.81%	0%	0%

[1]Table 1: Classification error rate comparison on the associative retrieval task.

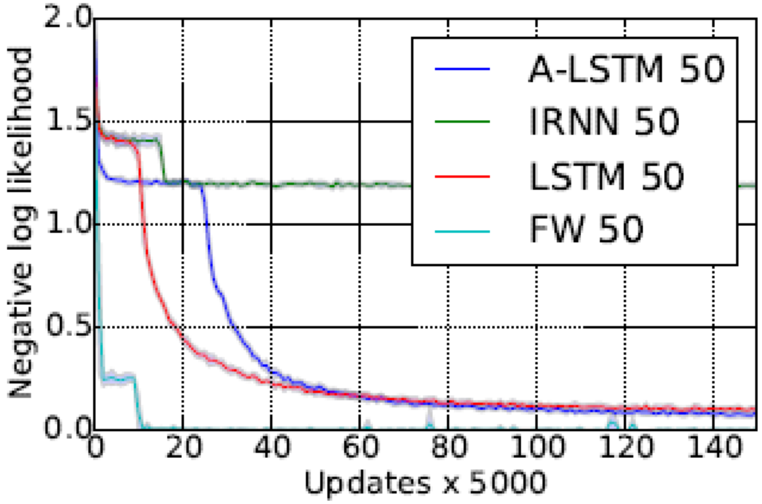


Figure 2: Comparison of the test log likelihood on the associative retrieval task with 50 recurrent hidden units.

• II. Fast Weights

- **Highlights**

a) It strengthens IRNN

Fast weights: The fast weights learning rate, η, is set to 0.5 and the fast weights decay rate, λ, is set to 0.9. The fast weights are updated once at every time step. We experimented with more iterations for the "inner loop" and the performance are similar. The recurrent slow weights are initialized to an identity matrix scaled by 0.05. We use the ReLU activation for $f(\cdot)$ in the recurrent layer.

IRNN: The recurrent slow weights are initialized to an identity matrix scaled by 0.5. ReLU is used as the non-linearity in the recurrent layer.

b) $s = 1$

c) Layer Normalization is good

$$h_{s+1}(t+1) = f(\mathcal{LN}[Wh(t) + Cx(t) + A(t)h_s(t+1)]) \tag{5}$$

Bibliography

- PLSTM TensorFlow implement
- PLSTM Keras implement
- Using Fast Weights to Attend to the Recent Past (arXiv)
- Fast Weights TensorFlow implement

Thanks for listening!