Reconfigurable Computing

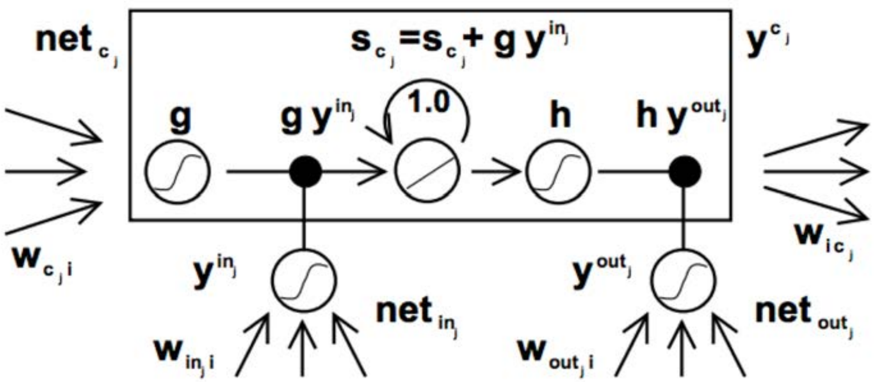
Long Short-Term Memory

Philip Leong 梁恆惠 (Slides Rui Tang)
School of Electrical and Information Engineering

http://www.ee.usyd.edu.au/people/philip.leong

THE UNIVERSITY OF SYDNEY

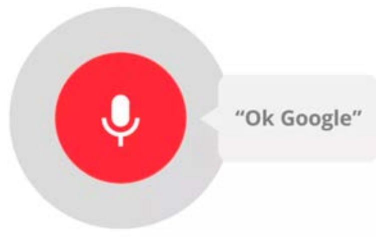
What is an LSTM?

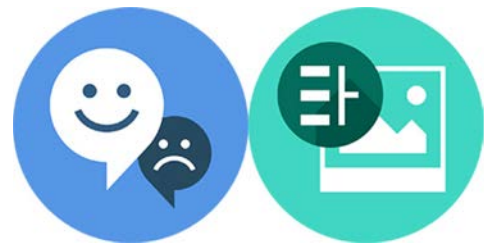


(Hocreiter and Schmidhuber, 1997)

- Long Short-Term Memory (LSTM) is a type of **gated** Recurrent Neural Network (RNN)
- Proposed by Hocreiter and Schmidhuber in 1997

Applications





↑ a living room with a couch and a television

↑ a man riding a bike on a beach

a man is walking down the street with a suitcase ↗

Applications

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

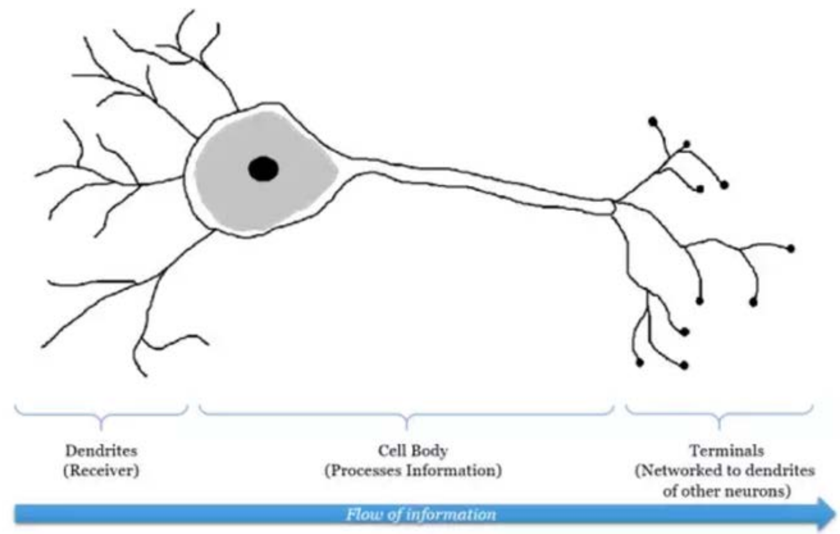
Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

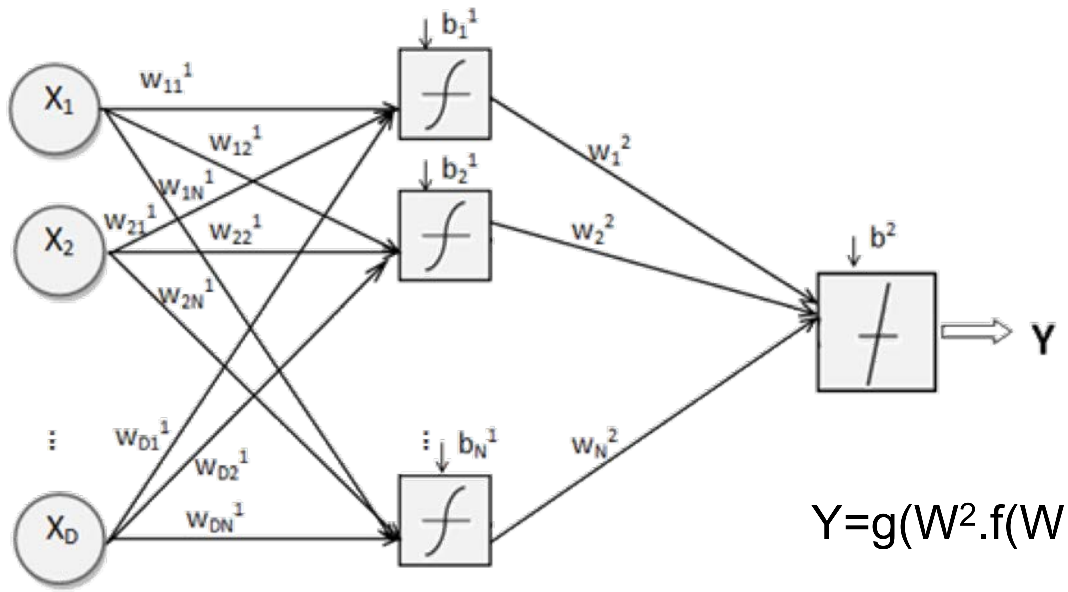
where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Source: Andre Karpthy Blog

Neural Networks



Feedforward Neural Network (Multilayer Perceptron)



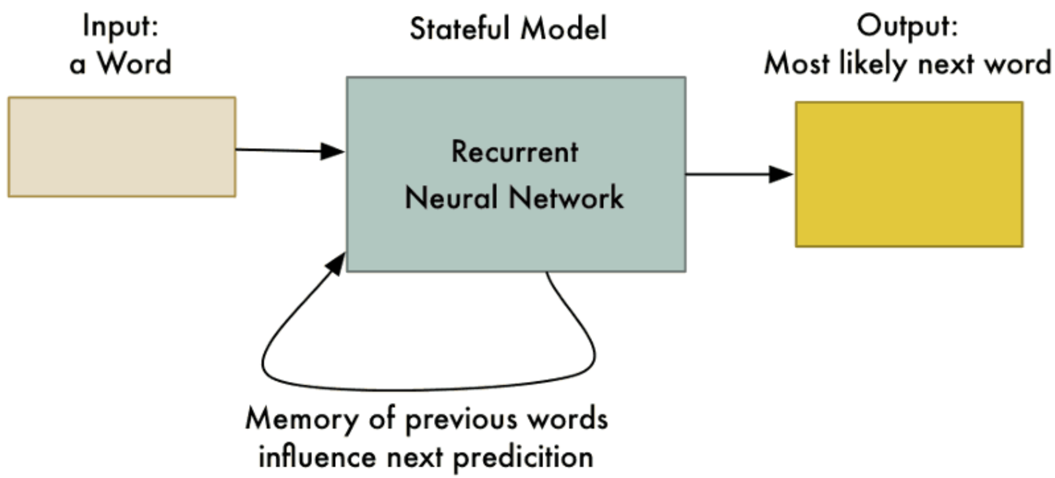
$$Y=g(W^2.f(W^1.X+b^1)+b^2)$$

(Matheus, 2016)

Disadvantages of MLP

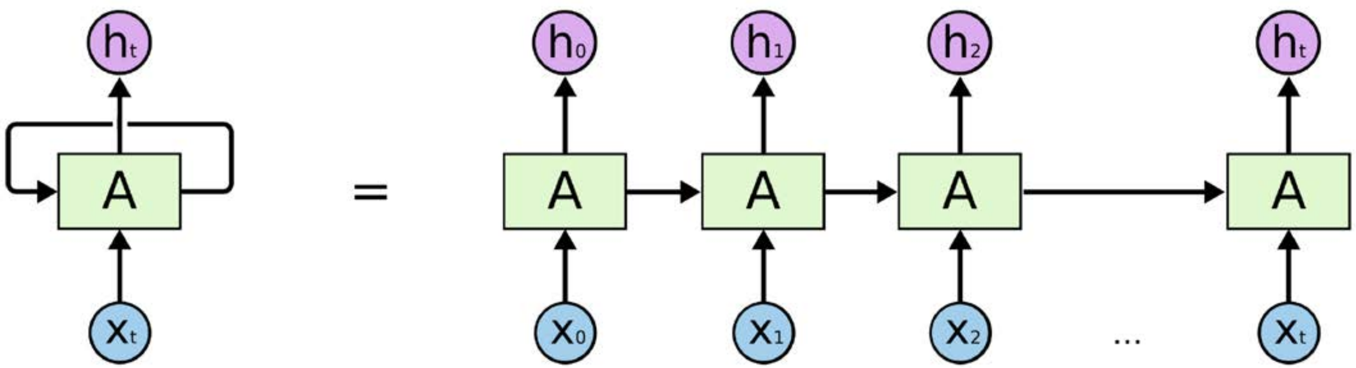
Fixed architecture → No memory → Can’t deal with sequential data

Recurrent Neural Network (RNN)



Output so far:

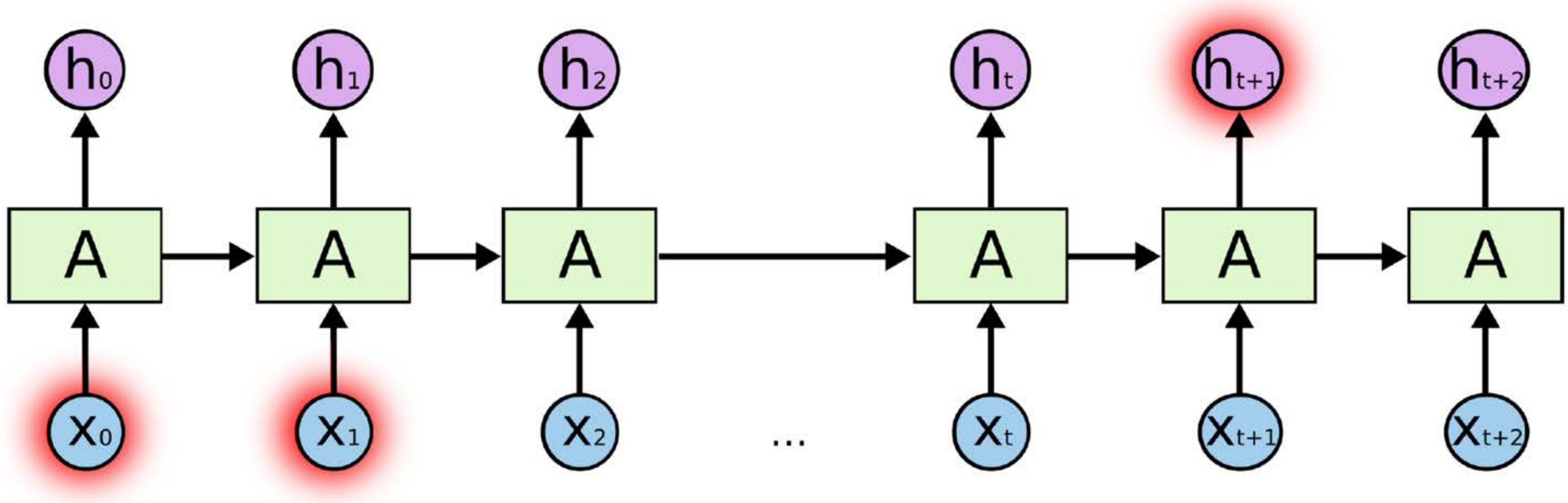
Machine



An unrolled recurrent neural network.

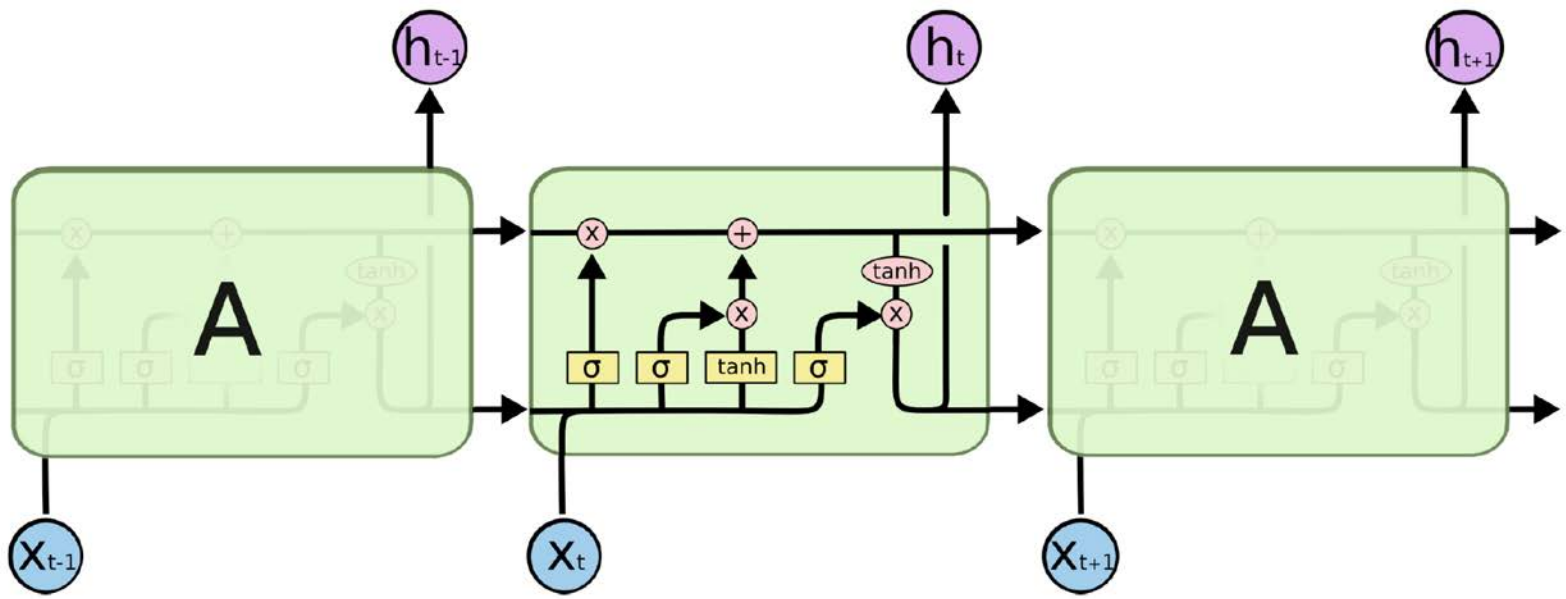
Source: colah's blog

Hard for RNNs to Learn Long Term Dependencies

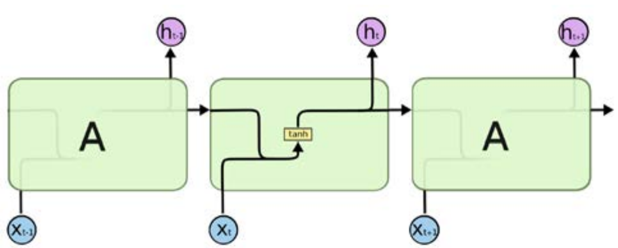


Source: colah's blog

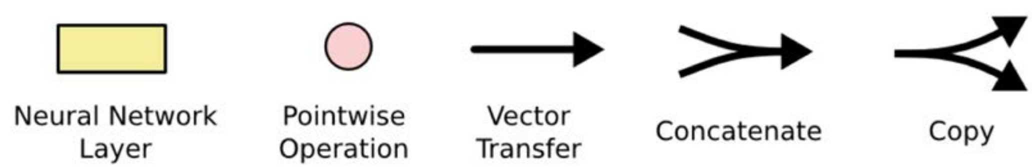
LSTM



The repeating module in an LSTM contains four interacting layers.

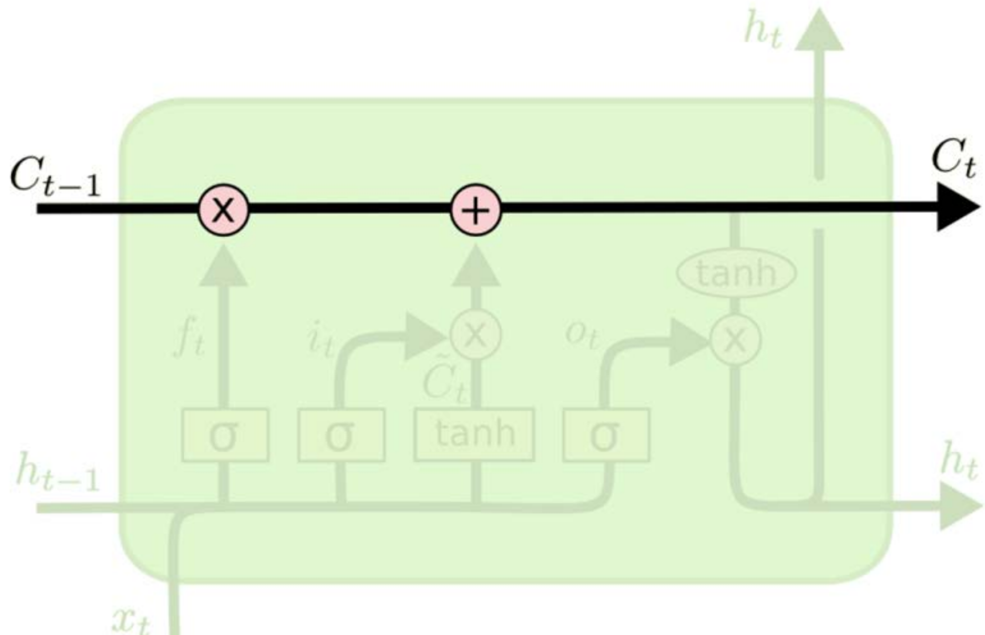


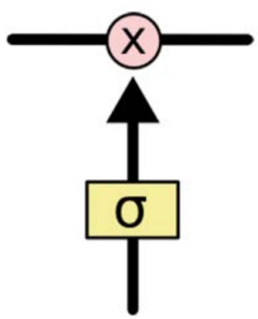
The repeating module in a standard RNN contains a single layer.



Source: colah's blog

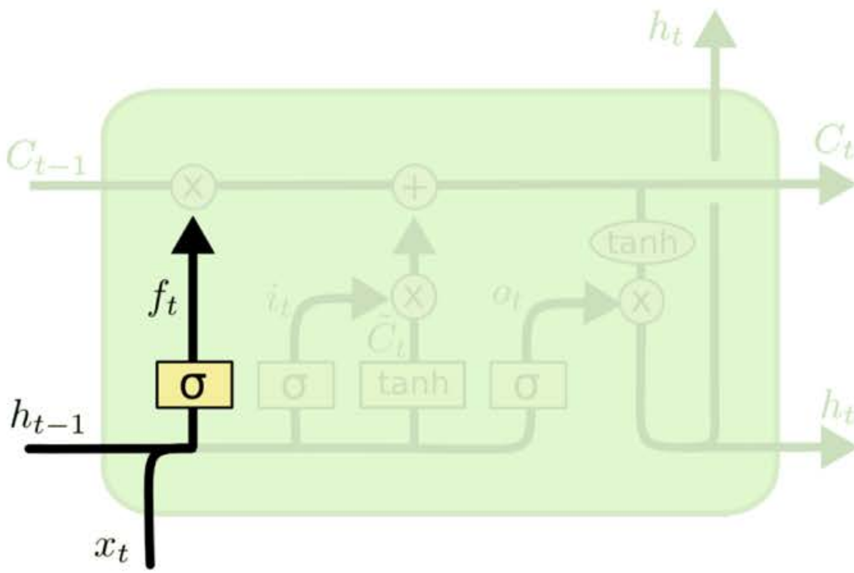
Cell State





Gate optionally lets information through

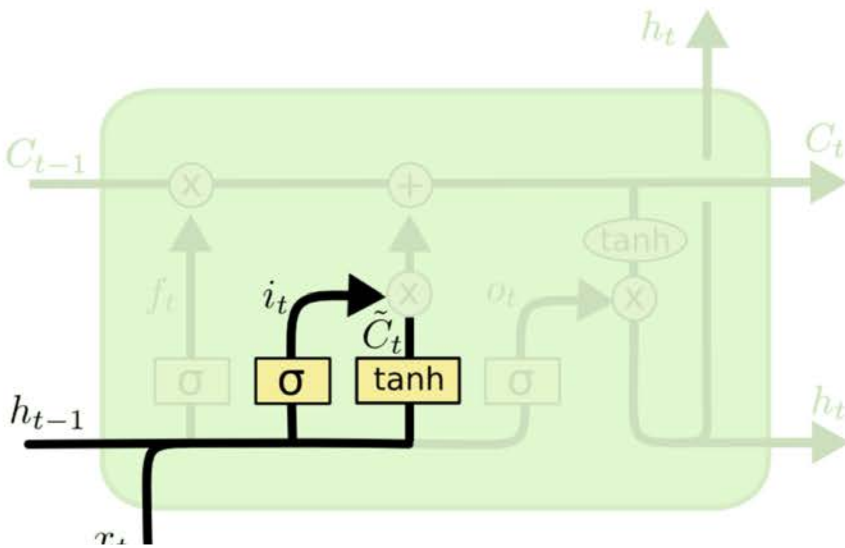
Forget Gate controls what state we forget



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Source: colah's blog

State Update

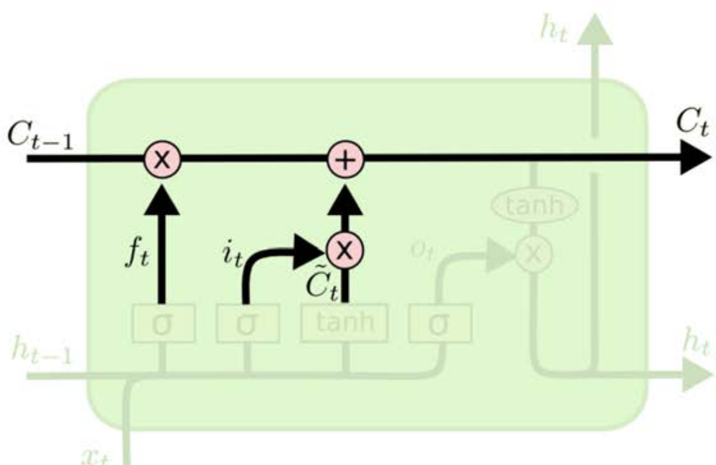


Input gate decides which values we will update

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Candidate values could be added to the state

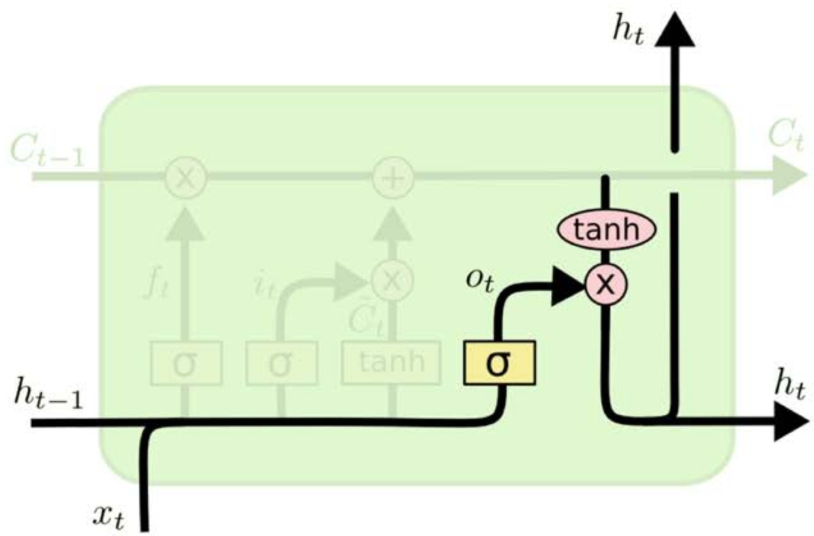


Apply forget and scaled candidate values

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source: colah's blog

Output Values



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Source: colah's blog

BasicLSTMCell

$$LSTM : h_t^{l-1}, h_{t-1}^l, c_{t-1}^l \rightarrow h_t^l, c_t^l$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} = T_{(n_{l-1}+n_l),(4n_l)} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$\begin{aligned} c_t^l &= f \odot c_{t-1}^l + i \odot g \\ h_l^t &= o \odot \tanh(c_t^l) \end{aligned}$$

How to get lab materials

- Firset need VM
- Install git
- git clone https://github.com/phwl/hlslstm.git

THE UNIVERSITY OF SYDNEY

Files

- xsimple.py – generates a simple random LSTM network using Tensorflow and calls lstmgen.py
- lstmgen.py – can verify a LSTM generated above and also output a C program
- simple.cpp – C program implementing an LSTM generated from the xsimple.py network

Simple.cpp

```
void
lstm(double c[], double h[], double x[])
{
    /* constant offsets used to access the i, j, f and o parts of r */
    #define pi (r + 0 * L_YDIM)
    #define pj (r + 1 * L_YDIM)
    #define pf (r + 2 * L_YDIM)
    #define po (r + 3 * L_YDIM)


    /* state information */
    static double new_c[L_YDIM];


    static double new_h[L_YDIM];
    static double xc[L_XDIM + L_YDIM];
    static double r[4 * L_YDIM];


    /* xc = np.hstack((x,  h)) */
    vcp((double *)xc, x, L_XDIM);
    vcp(xc + L_XDIM, h, L_YDIM);

    /* [i, j, f, o] = np.split(np.dot(xc, self.w) + self.b, 4) */
    maxpb(r, xc, (double *)l_w, l_b, 4 * L_YDIM,
                L_XDIM + L_YDIM);


    vplusbias(pf, 1.0, L_YDIM);
    vsigmoid(pf, L_YDIM);
    vsigmoid(pi, L_YDIM);
    vtanh(pj, L_YDIM);
    vmulsum(new_c, c, pf, pi, pj, L_YDIM);


    /* new_h = self.act(new_c) * sigmoid(o) */
    vcp(c, new_c, L_YDIM);
    vtanh(new_c, L_YDIM);
    vsigmoid(po, L_YDIM);
    vmul(new_h, new_c, po, L_YDIM);


    /* self.state = [new_c, new_h] */
    vcp(h, new_h, L_YDIM);
    vprint("y_pred", h, L_YDIM);
}
```

Matrix Multiply

```
void maxpb(double y[], double x[], double w[], double b[], const int ni, const int nj)
{
        double acc;      // move clearing of y to inner loop to make a perfect loop

        for (int i = 0; i < ni; i++)
                y[i] = b[i];
        axi: for (int i = 0; i < ni; i++)   {
                axj: for (int j = 0; j < nj; j++) {
                        y[i] += x[j] * w[i * nj + j];
                }
        }
}
```

Vector Subroutines

```
void vplusbias(double a[], double b, int n) {
        for (int i = 0; i < n; i++)
                a[i] = a[i] + b;
}
/* apply sigmoid to a vector */
void vsigmoid(double a[], int n) {
        for (int i = 0; i < n; i++)
                a[i] = mysigmoid(a[i]);
}
```

THE UNIVERSITY OF SYDNEY

gen.h

```
#define nelts(x) (sizeof((x)) / sizeof((x)[0]))
#define L_XDIM 2
#define L_YDIM 3
#define L_PATS 4
extern void lstm(double c[L_YDIM], double h[L_YDIM], double x[L_XDIM]);
extern double l_x[4][2];
extern double l_y[4][3];
```

gen_w.h

double l_w[12][5] = {4.183394909e-01,2.469940186e-01,-1.266442835e-01,4.577356577e-01,-3.173722625e-01,-2.636492252e-03,-6.688433886e-02,9.145200253e-02,-2.286953330e-01,-4.959584475e-01,-4.348166585e-01,-…,-2.686411440e-01,-8.889436722e-03,-1.292907298e-01,4.951380491e-01,-4.314445853e-01,-1.879357994e-01,1.519984007e-02,-3.789746761e-01};

double l_b[12] = {0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00,0.000000000e+00};

gen_io.h

```
double l_x[4][2] = {2.216531949e-01,2.192355439e-01,1.712471809e-01,3.179686384e-01,6.996901971e-02,2.125932948e-01,3.136201263e-01,1.788043651e-01};

double l_y[4][3] = {2.500389516e-02,-2.618626226e-03,1.892238483e-02,5.262799934e-02,-4.173215944e-03,2.595511824e-02,6.007545069e-02,-7.846147753e-03,2.025869675e-02,6.615213305e-02,-2.034597099e-02,3.455903754e-02};
```

lstmgen.py

```
def ff(self, x, state):
        c, h = state
        xc = np.hstack([x, h])
        r = np.split(np.dot(xc, self.w) + self.b, 4)
        vprint('w', np.ndarray.flatten(np.array(self.w)))
        vprint('r', np.ndarray.flatten(np.array(r)))
        [i, j, f, o] = r
        new_c = (c * sigmoid(f + self.forget_bias) + sigmoid(i) * self.act(j))
        vprint('new_c', new_c)
        new_h = self.act(new_c) * sigmoid(o)
        state = [new_c, new_h]
        vprint('y_pred', new_h)
        return new_h, state
```

lstmgen.py

```
# generate a program with all the parameters and test set
def gen(self, x, y):
        print('** Generating output file %s' % self.fname)
        # generate include file
        fh = open(self.fname + '.h', 'w')
        fh.write('#define nelts(x) (sizeof((x)) / sizeof((x)[0]))\n')
        fh.write('#define %s %d\n' % ('L_XDIM', self.xdim))
        fh.write('#define %s %d\n' % ('L_YDIM', self.ydim))
        fh.write('#define %s %d\n' % ('L_PATS', y.shape[0]))
        fh.write('extern void lstm(double c[L_YDIM], double h[L_YDIM], double x[L_XDIM]);\n')
        fh.write(self.genarray(0, "l_x", x))
        fh.write(self.genarray(0, "l_y", y))
        fh.close()
        # transpose w so we access adjacent elements for x * W
        fh = open(self.fname + '_w.h', 'w')
        fh.write(self.genarray(1, "l_w", np.transpose(self.w)))
        fh.write(self.genarray(1, "l_b", self.b))
        fh.close()
        fh = open(self.fname + '_io.h', 'w')
        fh.write(self.genarray(1, "l_x", x))
        fh.write(self.genarray(1, "l_y", y))
        fh.close()
```

xsimple.py

```
with tf.Session() as sess:
        sess.run(init_op)
        inp = train_input[0:batch_size]
        outputs = sess.run(output, {input_placeholder: inp})
        print("output values: ")
        print(outputs)
        states = sess.run(state,{input_placeholder: inp})
        print("internal states: ")
        print(states)    # print weights
        cg = cgen(input_length, num_hidden,
        tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES))
        state = [np.zeros(num_hidden), np.zeros(num_hidden)]
        for p in train_input[0]:
                output, state = cg.ff(p, state)
        cg.gen((np.array(train_input))[0], outputs[0])
```

```
CFLAGS= -p
simple:  gen.h simple.cpp
         g++ $(CFLAGS) -o simple simple.cpp –lm
test:    simple
         grep pred xsimple.out > /tmp/xsimple.out
         echo "testing simple ..."
         -(./simple | grep pred | sdiff -w80 /tmp/xsimple.out -)
         -rm -f /tmp/xsimple.out
gen.h:   xsimple.py lstmgen.py
         python xsimple.py >xsimple.out
clean:
         -rm -rf *.o *.dSYM *.pyc gen xsimple.out gen.h gen_w.h gen_dotp.h
gen_streamx.h gen_streamy.h gen_io.h simple.o simple __pycache__ *.data gen.cpp
```