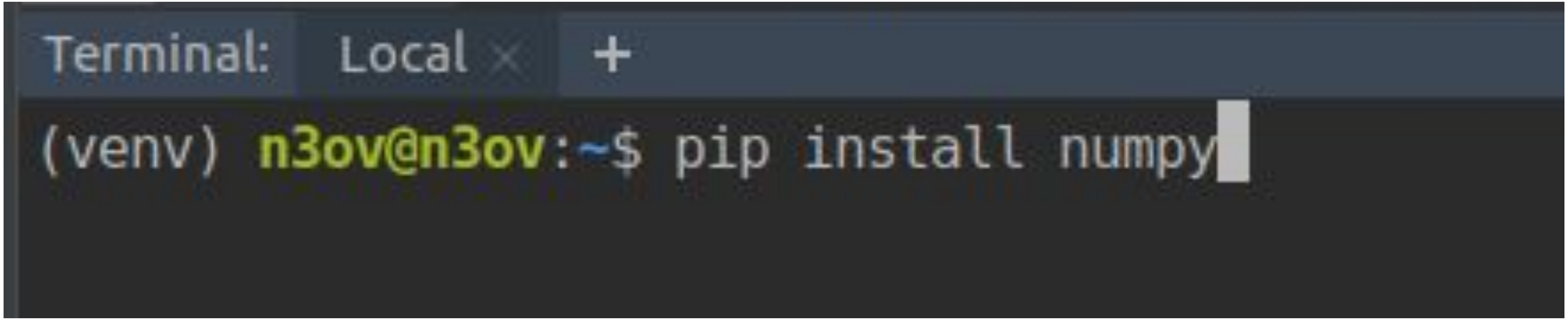
LSTM Hands on

LSTM.py

安裝環境

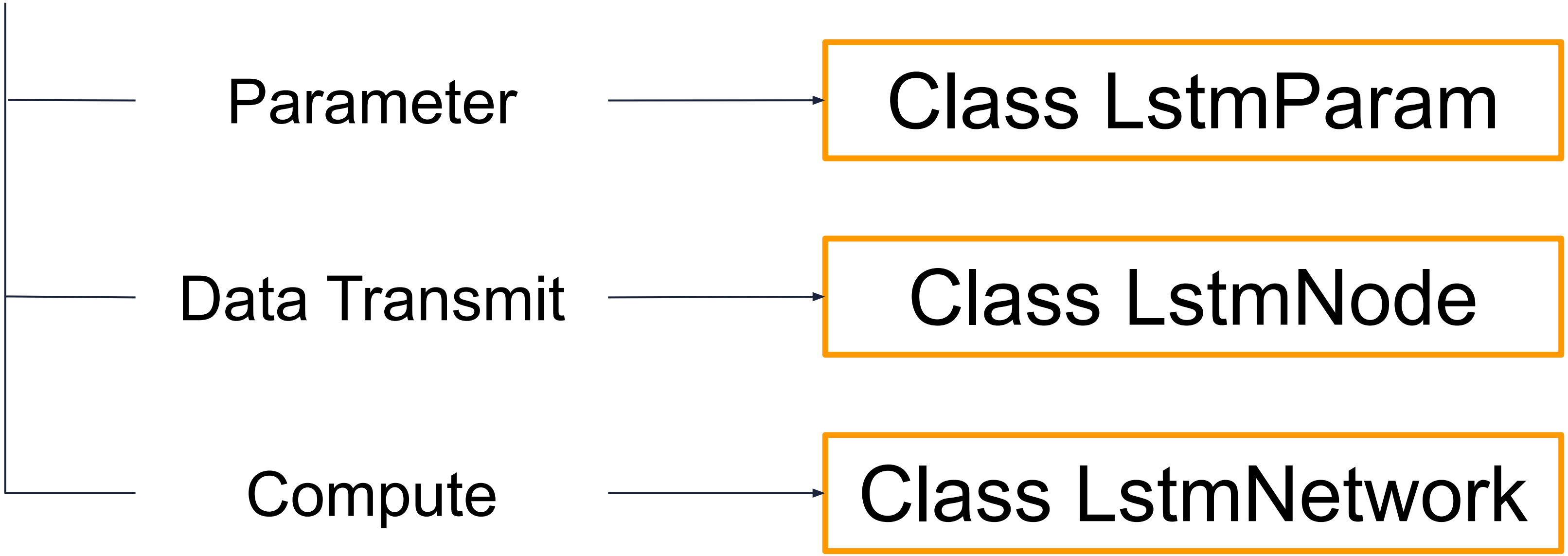


```
(venv) n3ov@n3ov:~$ pip install numpy
```

只需要安裝numpy

Define

Parameter, Feed Forward, Backpropagation



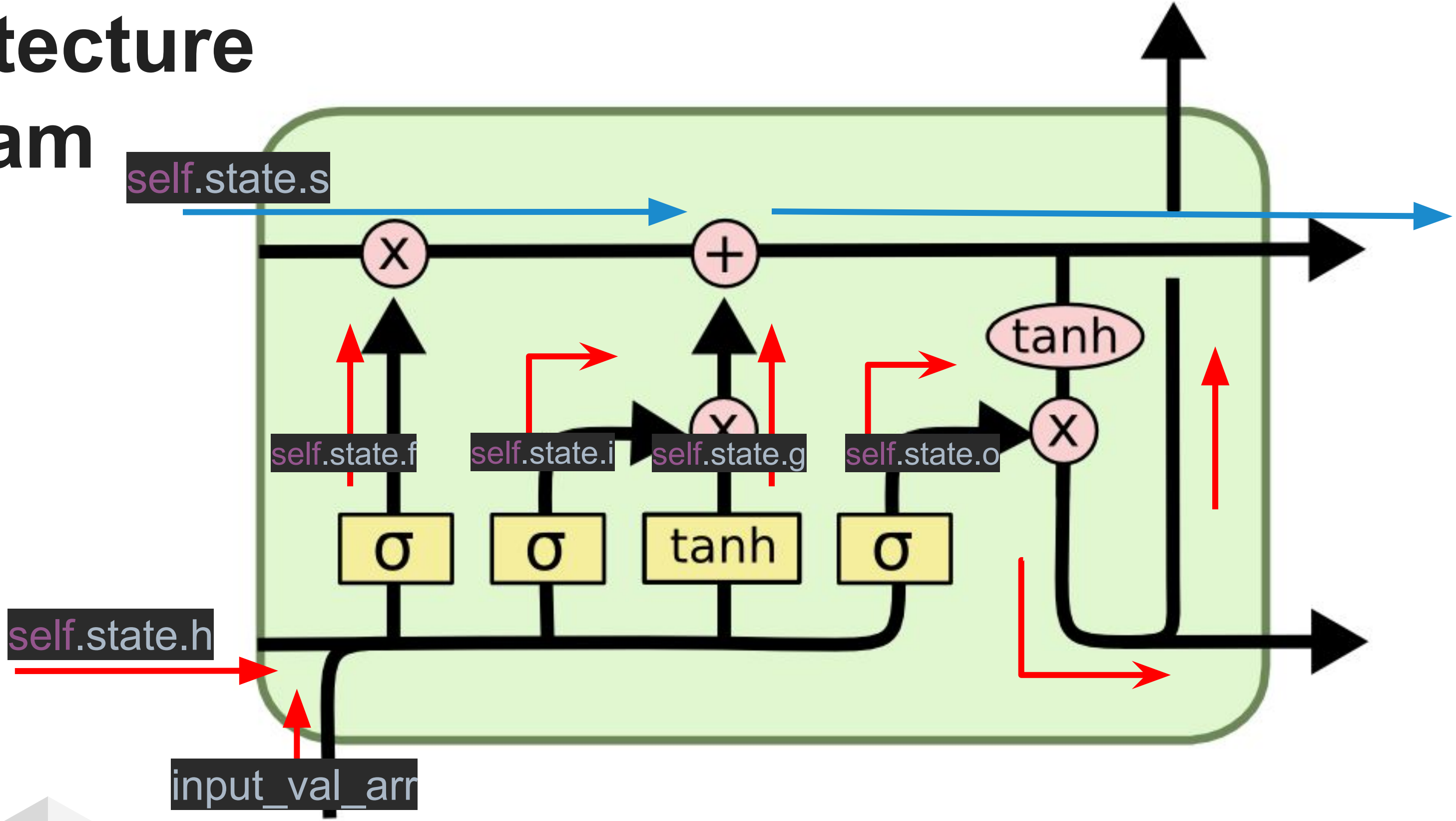
Parameter → Class LstmParam

Data Transmit → Class LstmNode

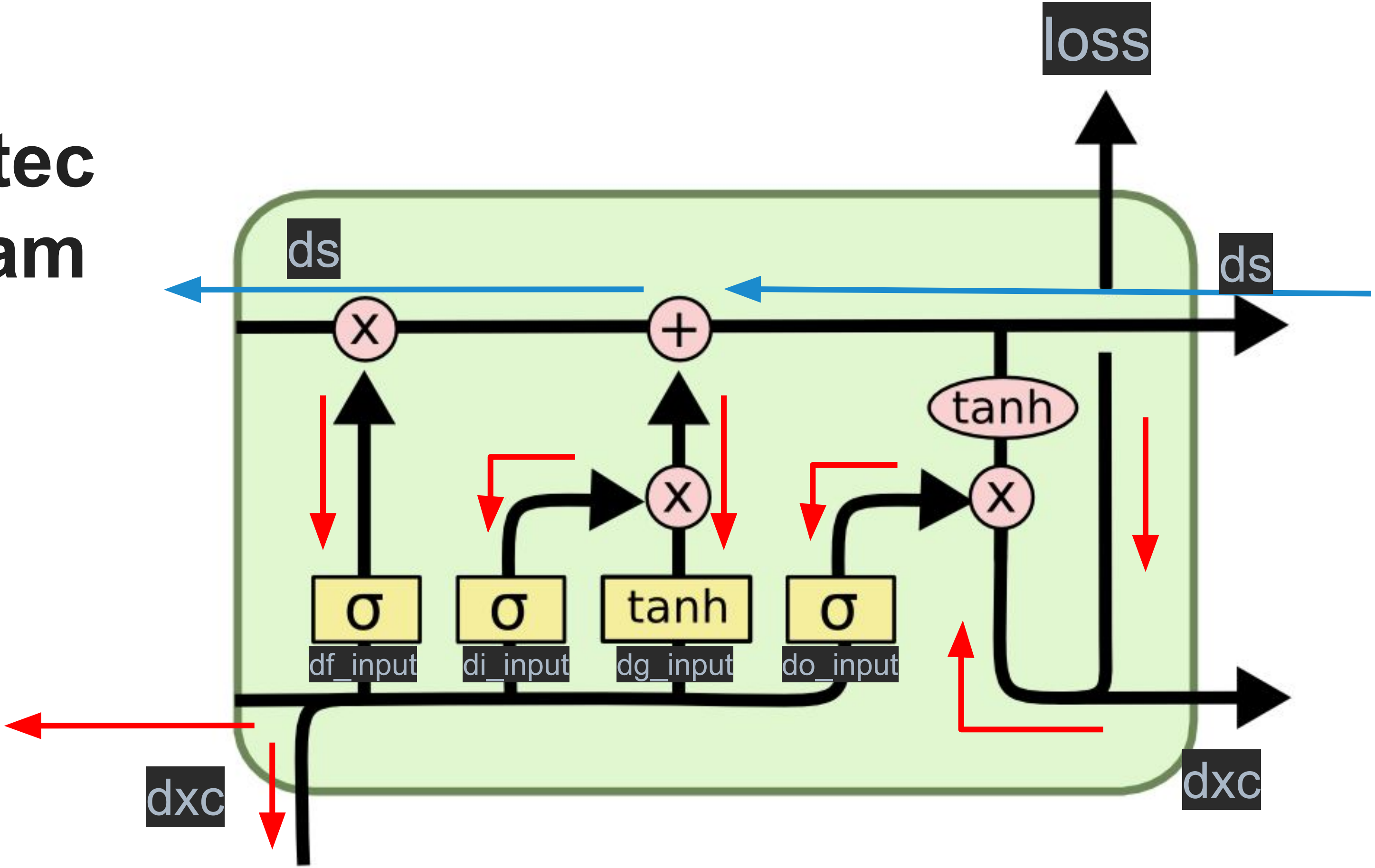
Compute → Class LstmNetwork

Feed Forward Architecture diagram

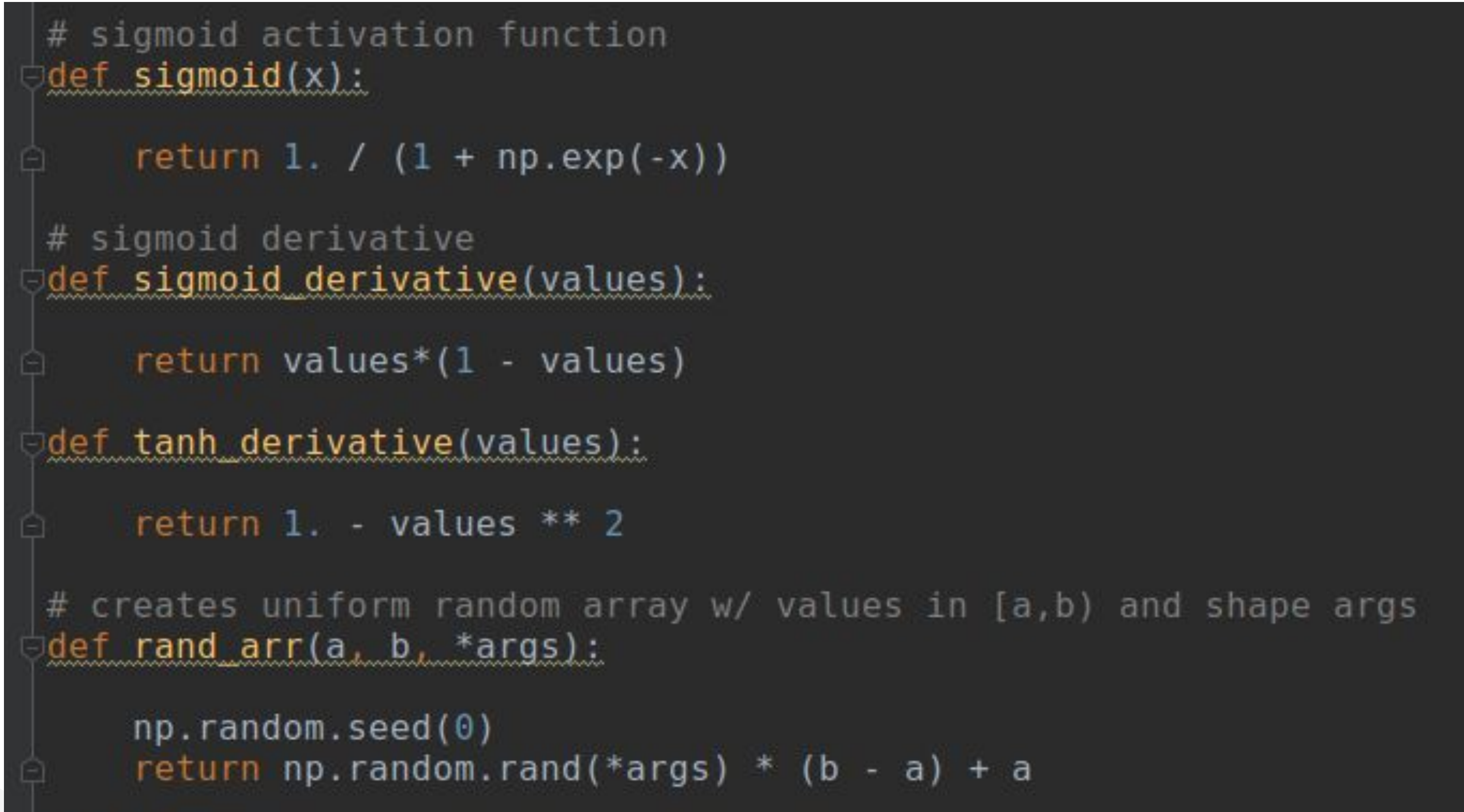
input = input_val_arr
output = lstm_net.lstm_node_list[ind].state.h[0]



BPTT Architec diagram

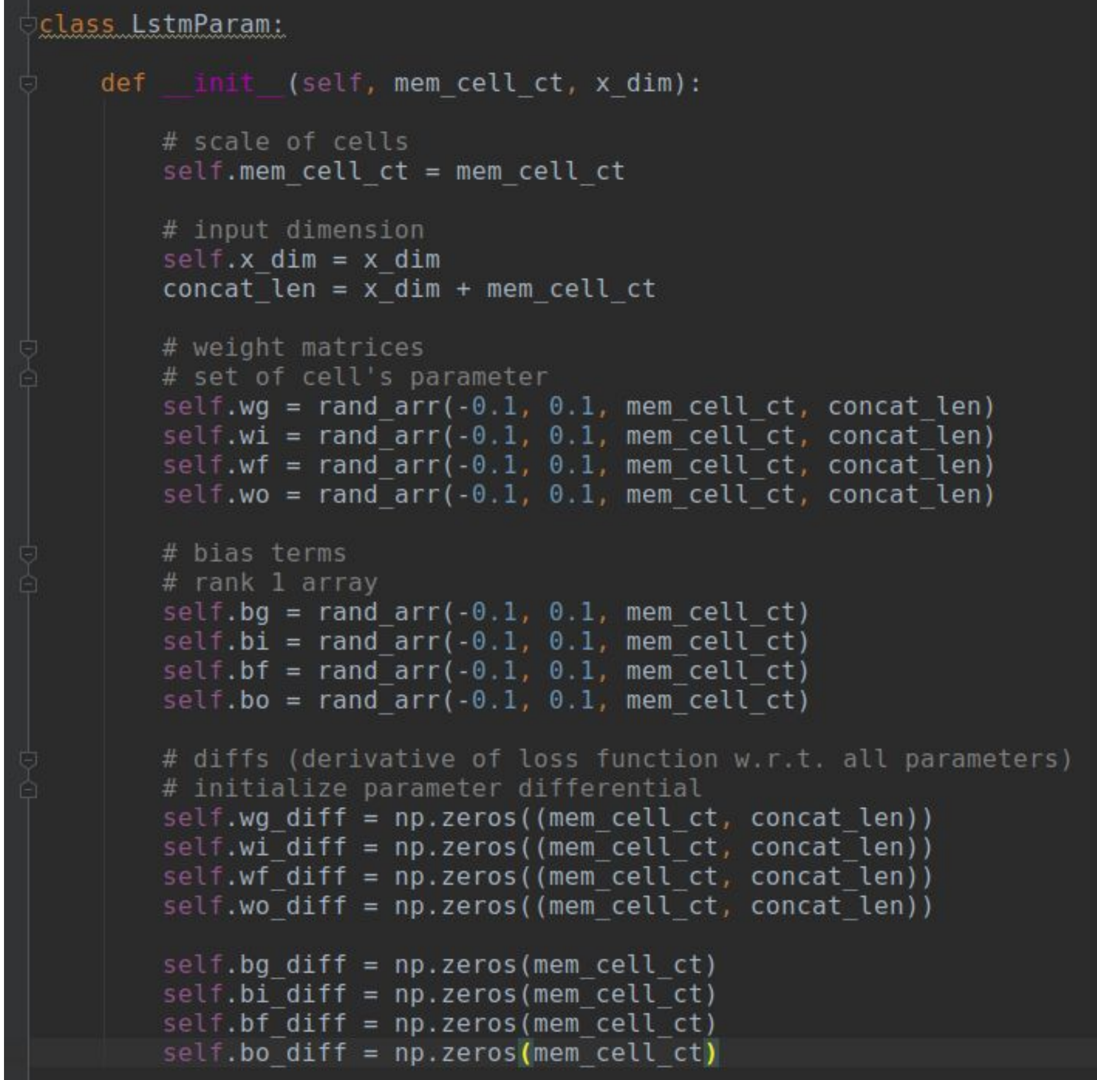


Activation function



```python
# sigmoid activation function
def sigmoid(x):

    return 1. / (1 + np.exp(-x))

# sigmoid derivative
def sigmoid_derivative(values):

    return values*(1 - values)

def tanh_derivative(values):

    return 1. - values ** 2

# creates uniform random array w/ values in [a,b) and shape args
def rand_arr(a, b, *args):

    np.random.seed(0)
    return np.random.rand(*args) * (b - a) + a
```

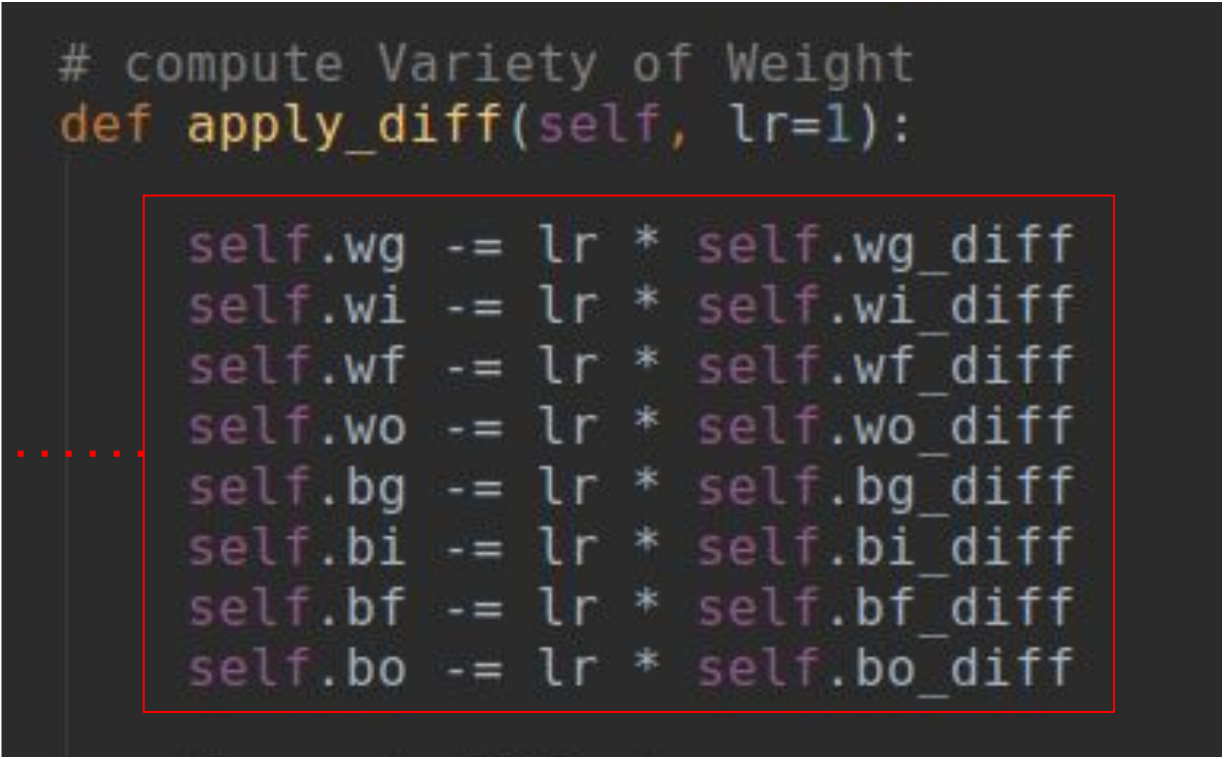
Initialize Parameter



```python
class LstmParam:

    def __init__(self, mem_cell_ct, x_dim):

        # scale of cells
        self.mem_cell_ct = mem_cell_ct

        # input dimension
        self.x_dim = x_dim
        concat_len = x_dim + mem_cell_ct

        # weight matrices
        # set of cell's parameter
        self.wg = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)
        self.wi = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)
        self.wf = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)
        self.wo = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)

        # bias terms
        # rank 1 array
        self.bg = rand_arr(-0.1, 0.1, mem_cell_ct)
        self.bi = rand_arr(-0.1, 0.1, mem_cell_ct)
        self.bf = rand_arr(-0.1, 0.1, mem_cell_ct)
        self.bo = rand_arr(-0.1, 0.1, mem_cell_ct)

        # diffs (derivative of loss function w.r.t. all parameters)
        # initialize parameter differential
        self.wg_diff = np.zeros((mem_cell_ct, concat_len))
        self.wi_diff = np.zeros((mem_cell_ct, concat_len))
        self.wf_diff = np.zeros((mem_cell_ct, concat_len))
        self.wo_diff = np.zeros((mem_cell_ct, concat_len))

        self.bg_diff = np.zeros(mem_cell_ct)
        self.bi_diff = np.zeros(mem_cell_ct)
        self.bf_diff = np.zeros(mem_cell_ct)
        self.bo_diff = np.zeros(mem_cell_ct)
```

Weight Update

$$\boxed{W = W - \Delta W}$$



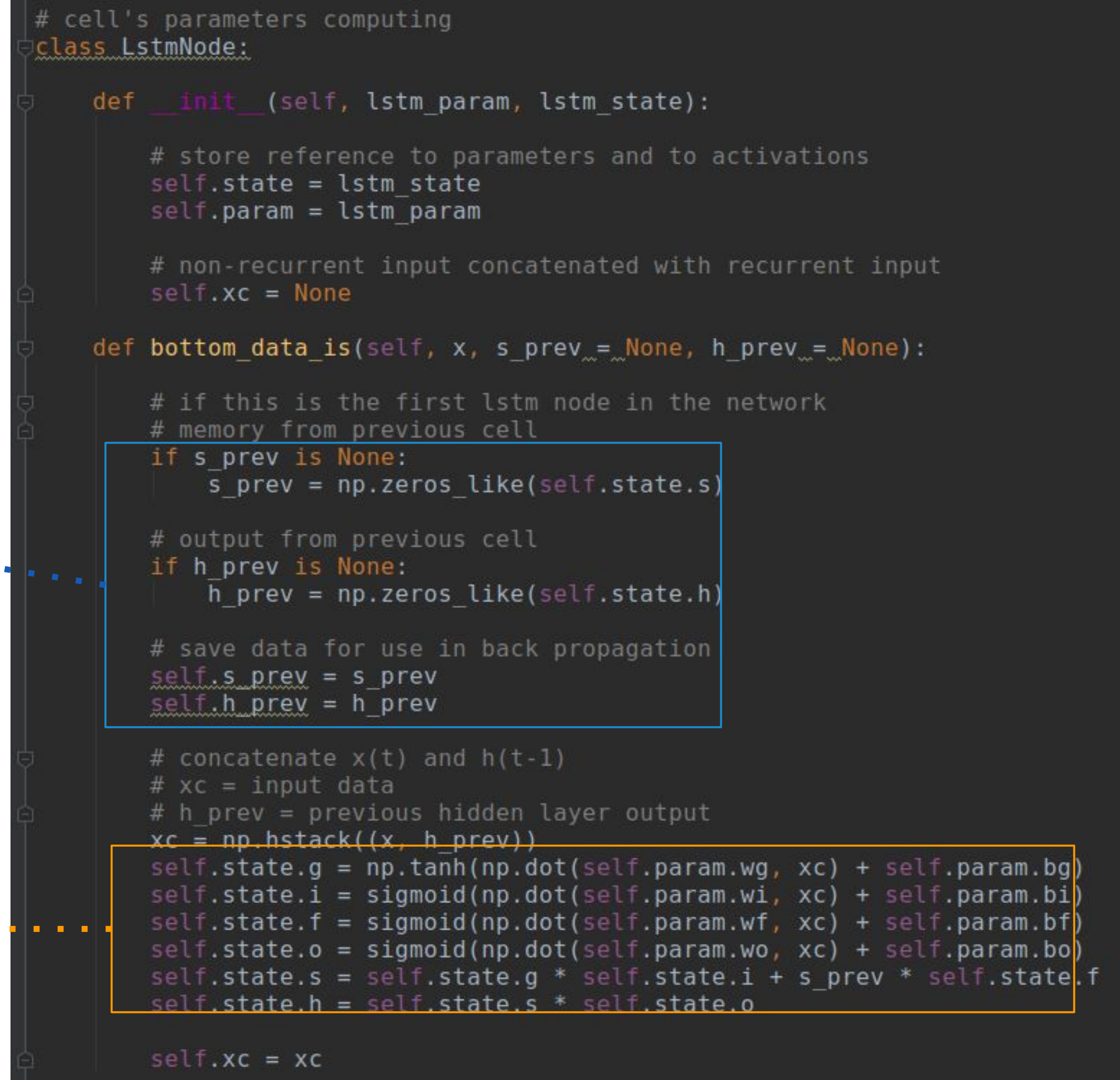
```
# compute Variety of Weight
def apply_diff(self, lr=1):

    self.wg -= lr * self.wg_diff
    self.wi -= lr * self.wi_diff
    self.wf -= lr * self.wf_diff
    self.wo -= lr * self.wo_diff
    self.bg -= lr * self.bg_diff
    self.bi -= lr * self.bi_diff
    self.bf -= lr * self.bf_diff
    self.bo -= lr * self.bo_diff
```

Compute Parameter

s = cell memory

h = hidden layer output

Feed Forword



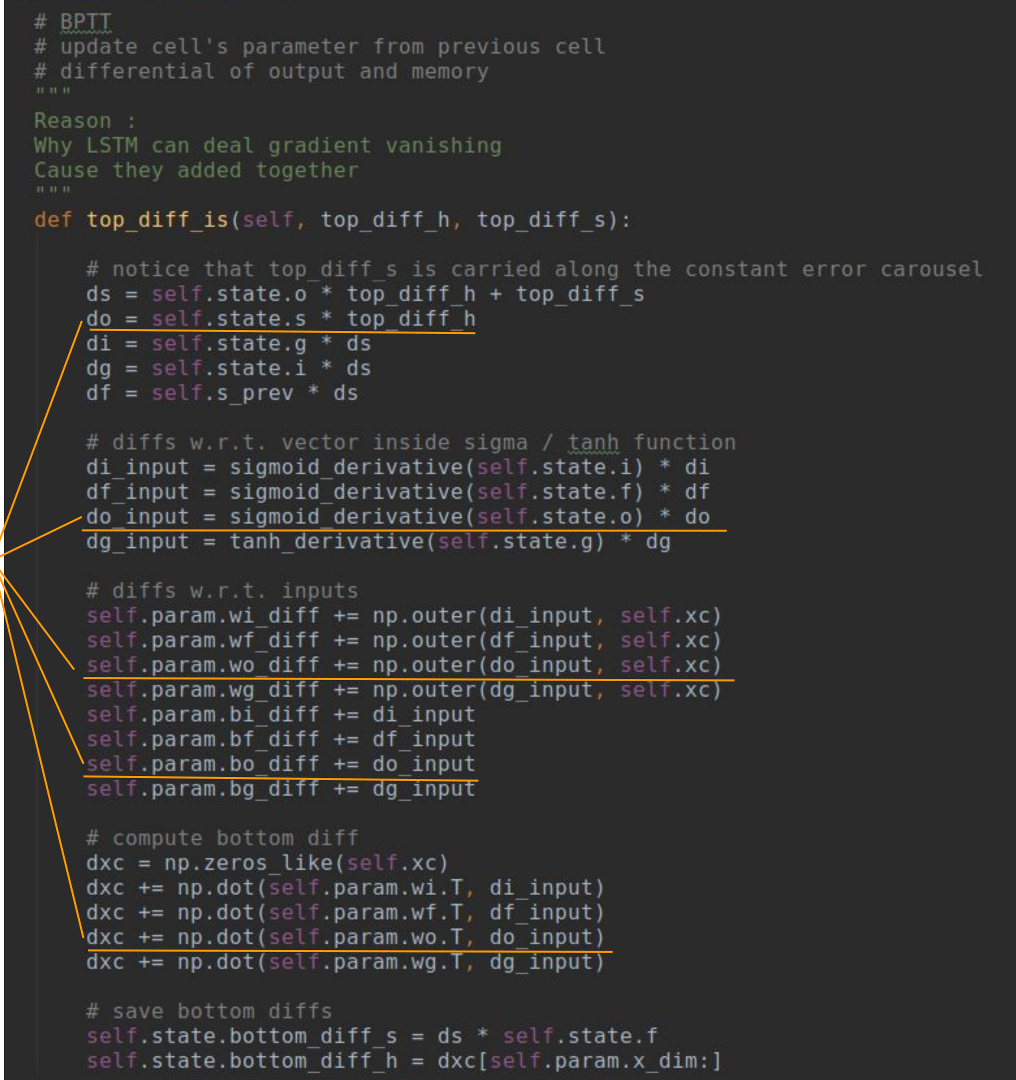
```
# cell's parameters computing
class LstmNode:

    def __init__(self, lstm_param, lstm_state):

        # store reference to parameters and to activations
        self.state = lstm_state
        self.param = lstm_param

        # non-recurrent input concatenated with recurrent input
        self.xc = None

    def bottom_data_is(self, x, s_prev = None, h_prev = None):

        # if this is the first lstm node in the network
        # memory from previous cell
        if s_prev is None:
            s_prev = np.zeros_like(self.state.s)

        # output from previous cell
        if h_prev is None:
            h_prev = np.zeros_like(self.state.h)

        # save data for use in back propagation
        self.s_prev = s_prev
        self.h_prev = h_prev

        # concatenate x(t) and h(t-1)
        # xc = input data
        # h_prev = previous hidden layer output
        xc = np.hstack((x, h_prev))
        self.state.g = np.tanh(np.dot(self.param.wg, xc) + self.param.bg)
        self.state.i = sigmoid(np.dot(self.param.wi, xc) + self.param.bi)
        self.state.f = sigmoid(np.dot(self.param.wf, xc) + self.param.bf)
        self.state.o = sigmoid(np.dot(self.param.wo, xc) + self.param.bo)
        self.state.s = self.state.g * self.state.i + s_prev * self.state.f
        self.state.h = self.state.s * self.state.o

        self.xc = xc
```

BPTT

$$\frac{dCost_2}{dW_o} = \frac{dCost_2}{dOutput_2}\,\frac{dOutput_2}{dO_2(out)}\,\frac{dO_2(out)}{dO_2(in)}\,\frac{dO_2(in)}{dW_o}$$

$$= (Output_2 - \hat{y}_2)\; tanh(C_2)\; (\sigma()\;(1-\sigma()))\; X_2$$

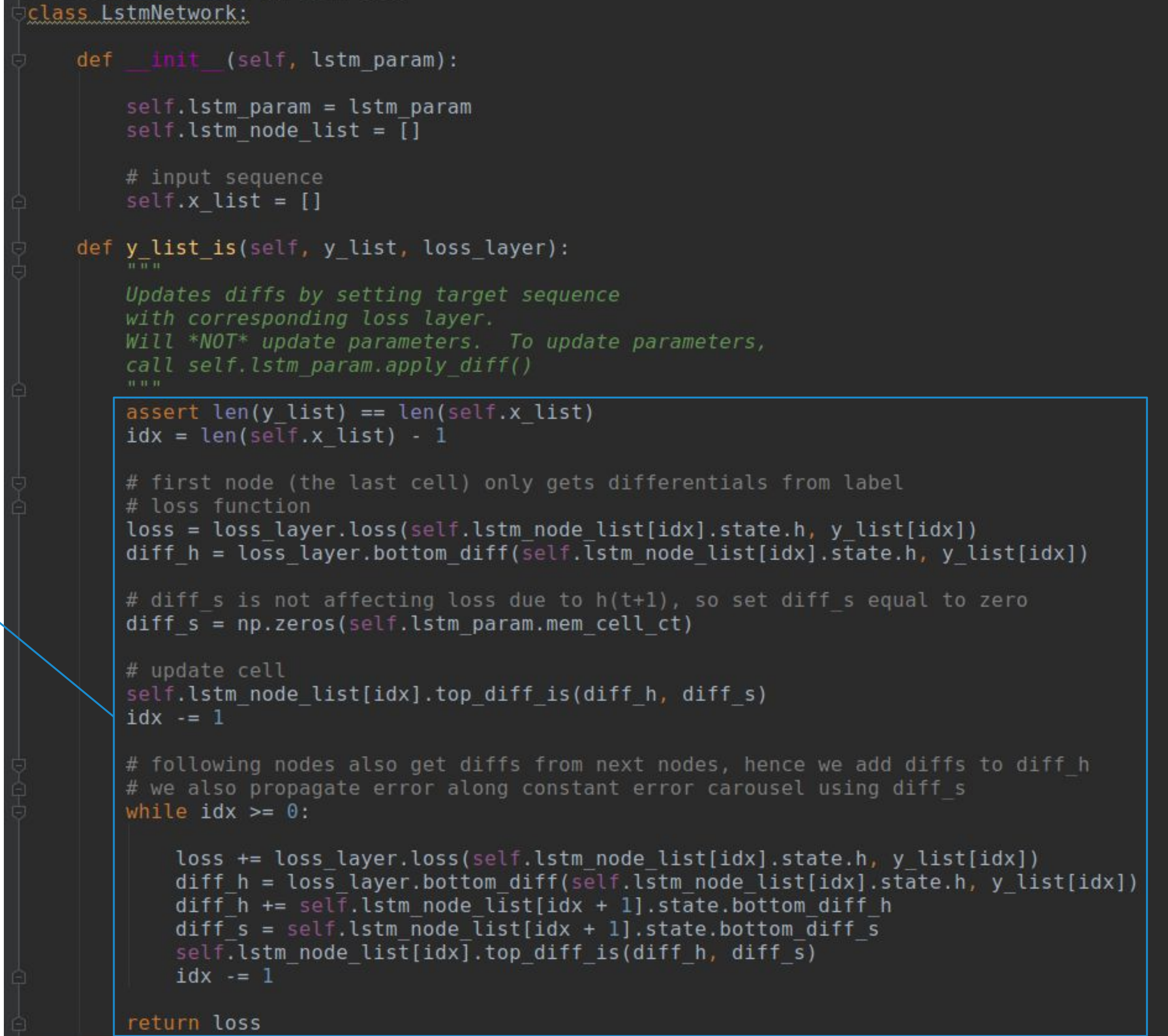
Example: Update the Weight of Output Gate



```
# BPTT
# update cell's parameter from previous cell
# differential of output and memory
"""
Reason :
Why LSTM can deal gradient vanishing
Cause they added together
"""
def top_diff_is(self, top_diff_h, top_diff_s):

    # notice that top_diff_s is carried along the constant error carousel
    ds = self.state.o * top_diff_h + top_diff_s
    do = self.state.s * top_diff_h
    di = self.state.g * ds
    dg = self.state.i * ds
    df = self.s_prev * ds

    # diffs w.r.t. vector inside sigma / tanh function
    di_input = sigmoid_derivative(self.state.i) * di
    df_input = sigmoid_derivative(self.state.f) * df
    do_input = sigmoid_derivative(self.state.o) * do
    dg_input = tanh_derivative(self.state.g) * dg

    # diffs w.r.t. inputs
    self.param.wi_diff += np.outer(di_input, self.xc)
    self.param.wf_diff += np.outer(df_input, self.xc)
    self.param.wo_diff += np.outer(do_input, self.xc)
    self.param.wg_diff += np.outer(dg_input, self.xc)
    self.param.bi_diff += di_input
    self.param.bf_diff += df_input
    self.param.bo_diff += do_input
    self.param.bg_diff += dg_input

    # compute bottom diff
    dxc = np.zeros_like(self.xc)
    dxc += np.dot(self.param.wi.T, di_input)
    dxc += np.dot(self.param.wf.T, df_input)
    dxc += np.dot(self.param.wo.T, do_input)
    dxc += np.dot(self.param.wg.T, dg_input)

    # save bottom diffs
    self.state.bottom_diff_s = ds * self.state.f
    self.state.bottom_diff_h = dxc[self.param.x_dim:]
```

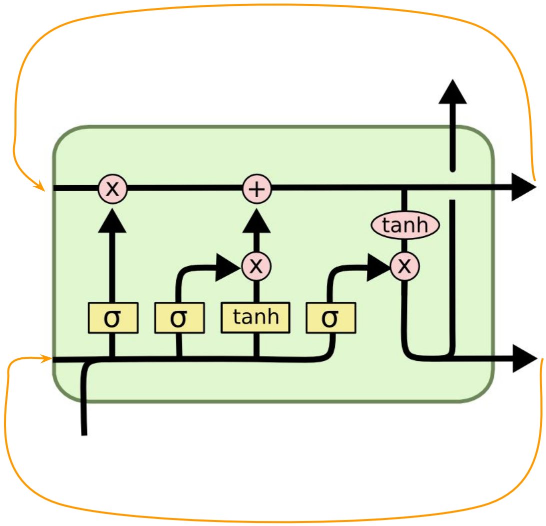
Compute loss

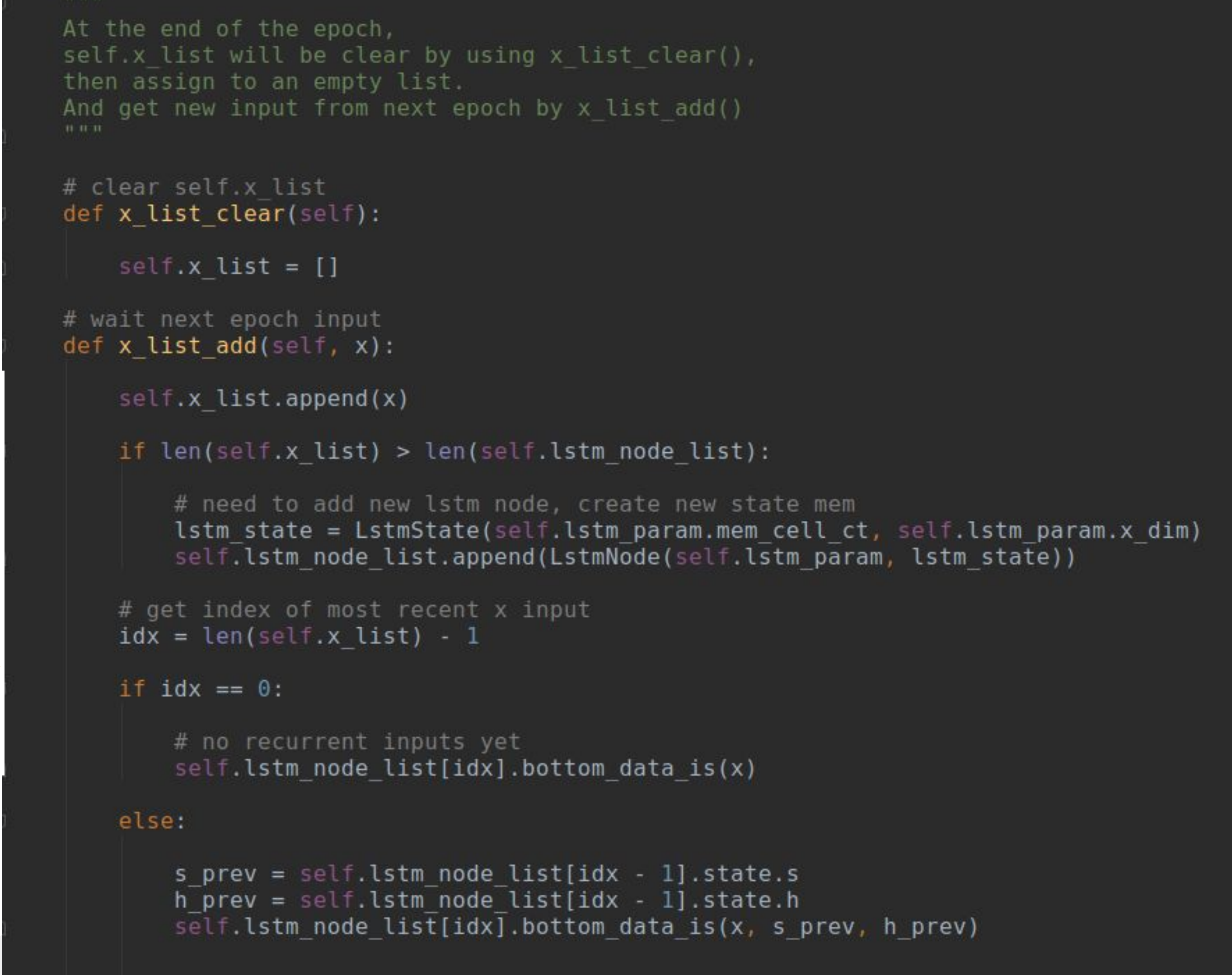
loss



```
class LstmNetwork:

    def __init__(self, lstm_param):

        self.lstm_param = lstm_param
        self.lstm_node_list = []

        # input sequence
        self.x_list = []

    def y_list_is(self, y_list, loss_layer):
        """
        Updates diffs by setting target sequence
        with corresponding loss layer.
        Will *NOT* update parameters.  To update parameters,
        call self.lstm_param.apply_diff()
        """
        assert len(y_list) == len(self.x_list)
        idx = len(self.x_list) - 1

        # first node (the last cell) only gets differentials from label
        # loss function
        loss = loss_layer.loss(self.lstm_node_list[idx].state.h, y_list[idx])
        diff_h = loss_layer.bottom_diff(self.lstm_node_list[idx].state.h, y_list[idx])

        # diff_s is not affecting loss due to h(t+1), so set diff_s equal to zero
        diff_s = np.zeros(self.lstm_param.mem_cell_ct)

        # update cell
        self.lstm_node_list[idx].top_diff_is(diff_h, diff_s)
        idx -= 1

        # following nodes also get diffs from next nodes, hence we add diffs to diff_h
        # we also propagate error along constant error carousel using diff_s
        while idx >= 0:

            loss += loss_layer.loss(self.lstm_node_list[idx].state.h, y_list[idx])
            diff_h = loss_layer.bottom_diff(self.lstm_node_list[idx].state.h, y_list[idx])
            diff_h += self.lstm_node_list[idx + 1].state.bottom_diff_h
            diff_s = self.lstm_node_list[idx + 1].state.bottom_diff_s
            self.lstm_node_list[idx].top_diff_is(diff_h, diff_s)
            idx -= 1

        return loss
```

Reset & Next input

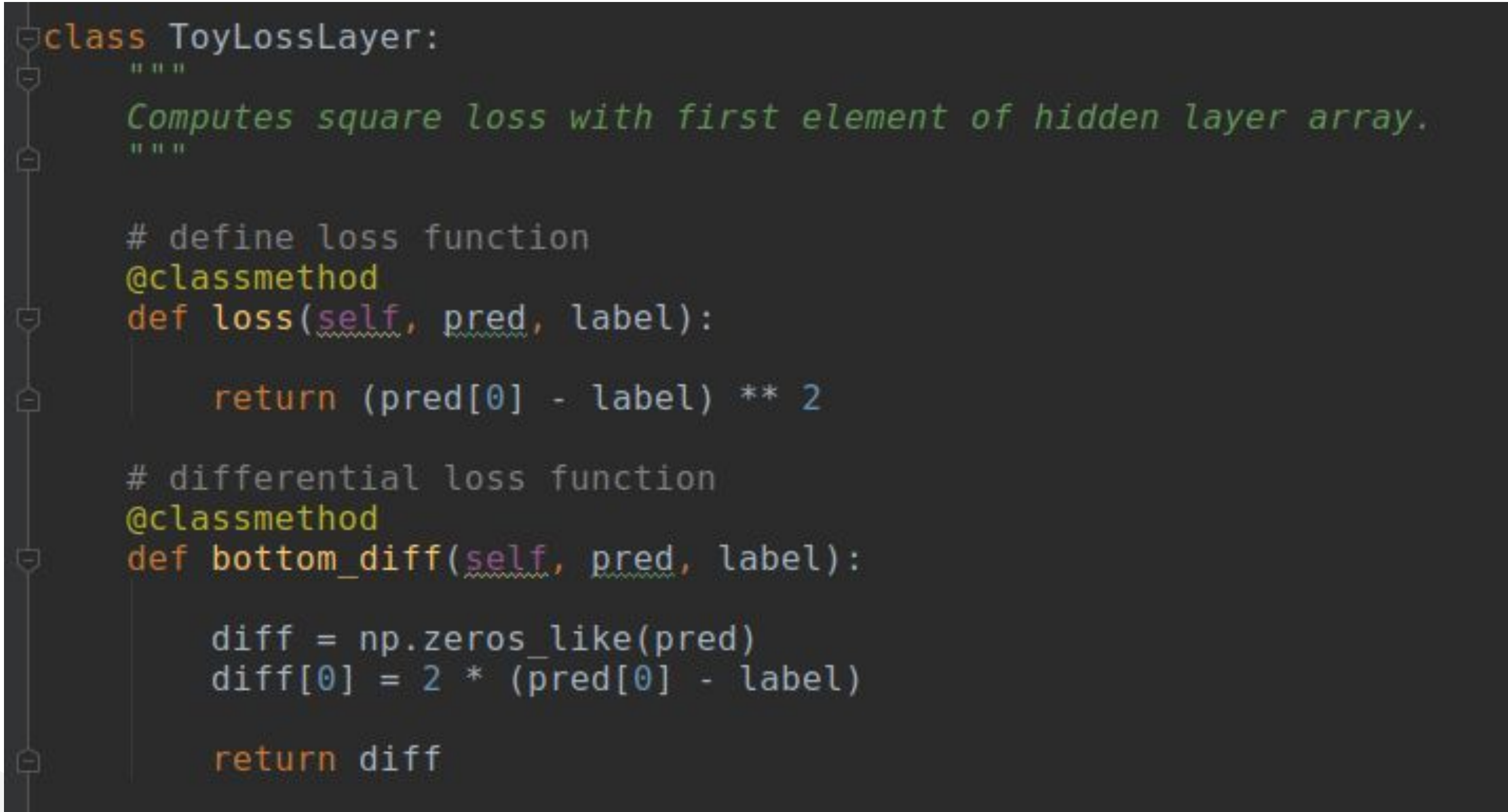




```
At the end of the epoch,
self.x_list will be clear by using x_list_clear(),
then assign to an empty list.
And get new input from next epoch by x_list_add()
"""

# clear self.x_list
def x_list_clear(self):

    self.x_list = []

# wait next epoch input
def x_list_add(self, x):

    self.x_list.append(x)

    if len(self.x_list) > len(self.lstm_node_list):

        # need to add new lstm node, create new state mem
        lstm_state = LstmState(self.lstm_param.mem_cell_ct, self.lstm_param.x_dim)
        self.lstm_node_list.append(LstmNode(self.lstm_param, lstm_state))

    # get index of most recent x input
    idx = len(self.x_list) - 1

    if idx == 0:

        # no recurrent inputs yet
        self.lstm_node_list[idx].bottom_data_is(x)

    else:

        s_prev = self.lstm_node_list[idx - 1].state.s
        h_prev = self.lstm_node_list[idx - 1].state.h
        self.lstm_node_list[idx].bottom_data_is(x, s_prev, h_prev)
```

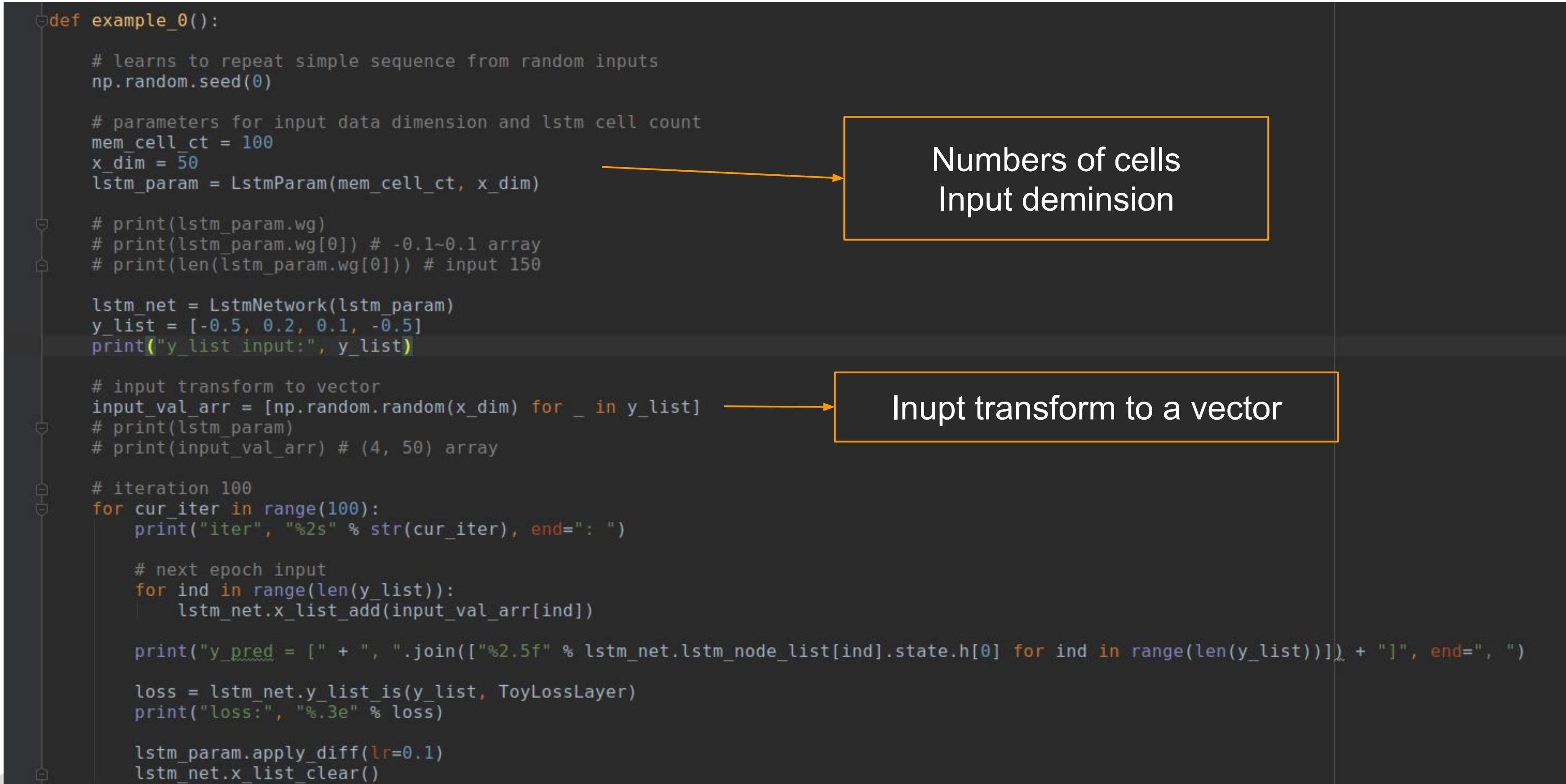
Test.py

Loss Function



```python
class ToyLossLayer:
    """
    Computes square loss with first element of hidden layer array.
    """

    # define loss function
    @classmethod
    def loss(self, pred, label):

        return (pred[0] - label) ** 2

    # differential loss function
    @classmethod
    def bottom_diff(self, pred, label):

        diff = np.zeros_like(pred)
        diff[0] = 2 * (pred[0] - label)

        return diff
```

Input



```
def example_0():

    # learns to repeat simple sequence from random inputs
    np.random.seed(0)

    # parameters for input data dimension and lstm cell count
    mem_cell_ct = 100
    x_dim = 50
    lstm_param = LstmParam(mem_cell_ct, x_dim)

    # print(lstm_param.wg)
    # print(lstm_param.wg[0]) # -0.1~0.1 array
    # print(len(lstm_param.wg[0])) # input 150

    lstm_net = LstmNetwork(lstm_param)
    y_list = [-0.5, 0.2, 0.1, -0.5]
    print("y_list input:", y_list)

    # input transform to vector
    input_val_arr = [np.random.random(x_dim) for _ in y_list]
    # print(lstm_param)
    # print(input_val_arr) # (4, 50) array

    # iteration 100
    for cur_iter in range(100):
        print("iter", "%2s" % str(cur_iter), end=": ")

        # next epoch input
        for ind in range(len(y_list)):
            lstm_net.x_list_add(input_val_arr[ind])

        print("y_pred = [" + ", ".join(["%2.5f" % lstm_net.lstm_node_list[ind].state.h[0] for ind in range(len(y_list))]) + "]", end=", ")

        loss = lstm_net.y_list_is(y_list, ToyLossLayer)
        print("loss:", "%.3e" % loss)

        lstm_param.apply_diff(lr=0.1)
        lstm_net.x_list_clear()
```

Numbers of cells
Input deminsion

Inupt transform to a vector

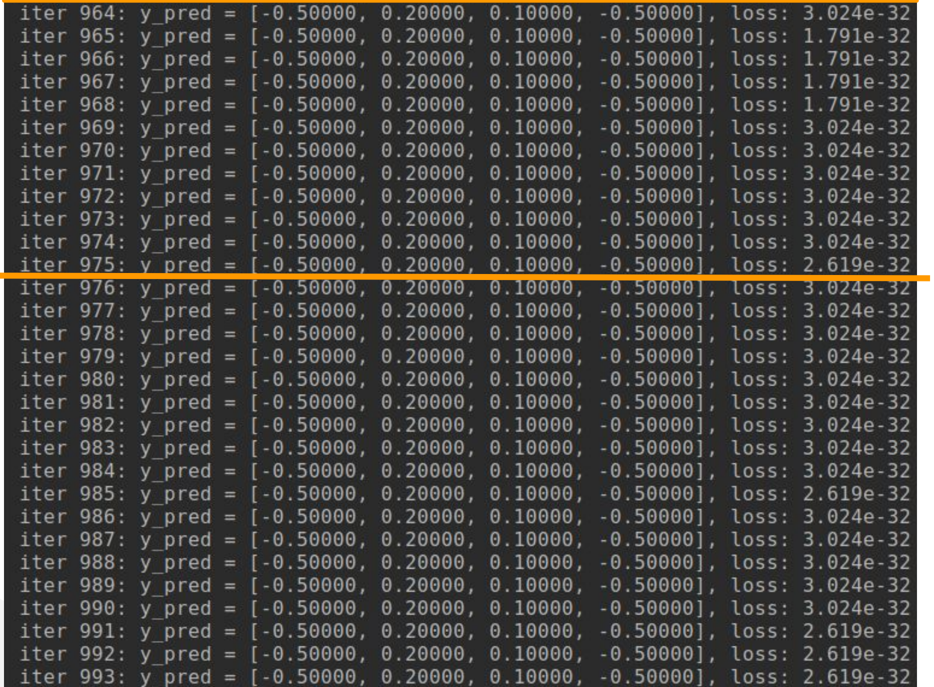
Proof Some Question in LSTM

Compare Weight Variety

Proof Gradient Explode in LSTM

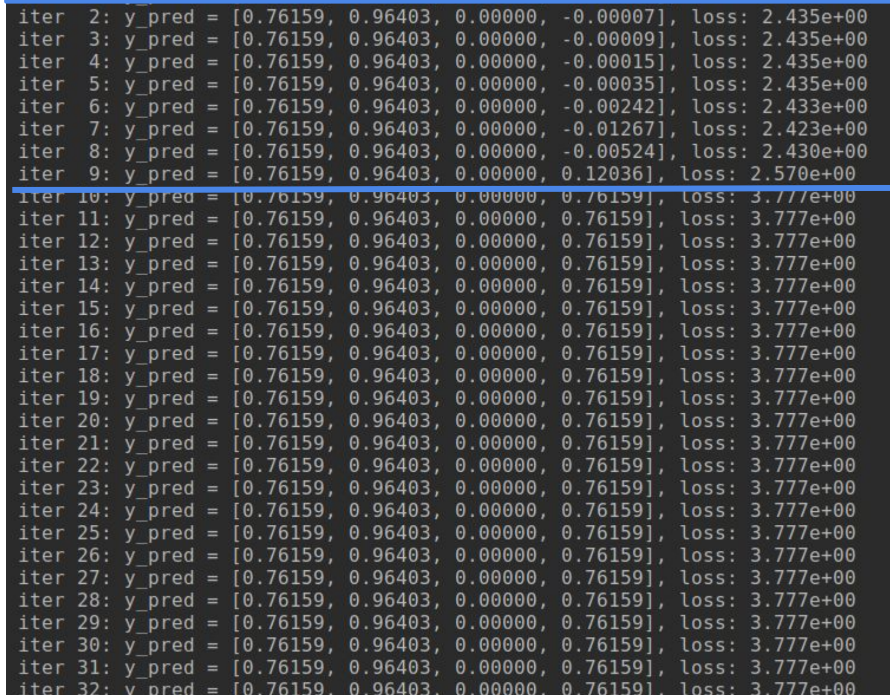
learning rate = 0.1
epoch = 1000
loss = Residual square
input = [-0.5, 0.2, 0.1, -0.5]

init weight = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)



```
iter 964: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 965: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.791e-32
iter 966: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.791e-32
iter 967: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.791e-32
iter 968: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.791e-32
iter 969: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 970: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 971: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 972: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 973: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 974: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 975: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
iter 976: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 977: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 978: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 979: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 980: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 981: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 982: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 983: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 984: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 985: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
iter 986: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 987: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 988: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 989: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 990: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 991: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
iter 992: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
iter 993: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
```

init weight = rand_arr(-10, 10, mem_cell_ct, concat_len)



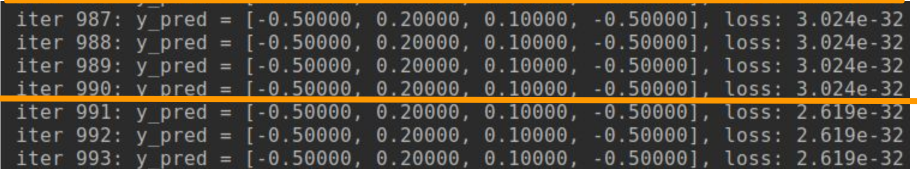
```
iter  2: y_pred = [0.76159, 0.96403, 0.00000, -0.00007], loss: 2.435e+00
iter  3: y_pred = [0.76159, 0.96403, 0.00000, -0.00009], loss: 2.435e+00
iter  4: y_pred = [0.76159, 0.96403, 0.00000, -0.00015], loss: 2.435e+00
iter  5: y_pred = [0.76159, 0.96403, 0.00000, -0.00035], loss: 2.435e+00
iter  6: y_pred = [0.76159, 0.96403, 0.00000, -0.00242], loss: 2.433e+00
iter  7: y_pred = [0.76159, 0.96403, 0.00000, -0.01267], loss: 2.423e+00
iter  8: y_pred = [0.76159, 0.96403, 0.00000, -0.00524], loss: 2.430e+00
iter  9: y_pred = [0.76159, 0.96403, 0.00000, 0.12036], loss: 2.570e+00
iter 10: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 11: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 12: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 13: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 14: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 15: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 16: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 17: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 18: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 19: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 20: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 21: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 22: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 23: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 24: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 25: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 26: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 27: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 28: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 29: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 30: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 31: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 32: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
```

Compare Weight Variety

Proof Gradient Explode in LSTM

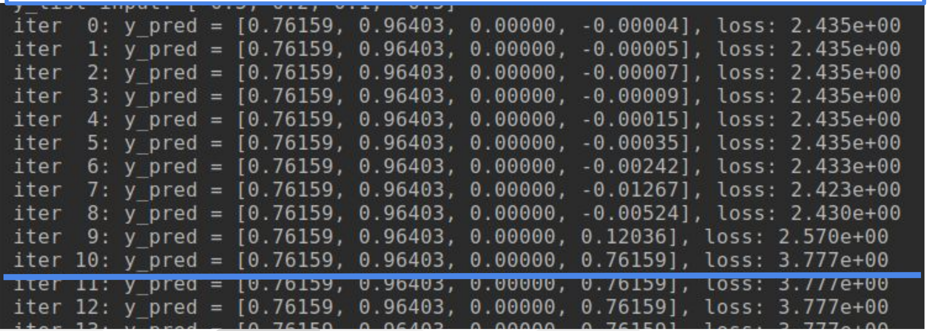
learning rate = 0.1
epoch = 1000
loss = Residual square
input = `[ -0.5,  0.2,  0.1,  -0.5 ]`

init weight = `rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)`



```
iter 987: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 988: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 989: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 990: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 3.024e-32
iter 991: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
iter 992: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
iter 993: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 2.619e-32
```

init weight = `rand_arr(-10,  10, mem_cell_ct, concat_len)`



```
iter  0: y_pred = [0.76159, 0.96403, 0.00000, -0.00004], loss: 2.435e+00
iter  1: y_pred = [0.76159, 0.96403, 0.00000, -0.00005], loss: 2.435e+00
iter  2: y_pred = [0.76159, 0.96403, 0.00000, -0.00007], loss: 2.435e+00
iter  3: y_pred = [0.76159, 0.96403, 0.00000, -0.00009], loss: 2.435e+00
iter  4: y_pred = [0.76159, 0.96403, 0.00000, -0.00015], loss: 2.435e+00
iter  5: y_pred = [0.76159, 0.96403, 0.00000, -0.00035], loss: 2.435e+00
iter  6: y_pred = [0.76159, 0.96403, 0.00000, -0.00242], loss: 2.433e+00
iter  7: y_pred = [0.76159, 0.96403, 0.00000, -0.01267], loss: 2.423e+00
iter  8: y_pred = [0.76159, 0.96403, 0.00000, -0.00524], loss: 2.430e+00
iter  9: y_pred = [0.76159, 0.96403, 0.00000, 0.12036], loss: 2.570e+00
iter 10: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 11: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
iter 12: y_pred = [0.76159, 0.96403, 0.00000, 0.76159], loss: 3.777e+00
```

The orange one shows the model can learn correctly and its loss very close to zero.

But the blue one shows the model interact after 9 times that loss won't be lower forever.

BPTT
LSTM can prevent Gradient Vanishing by using " Plus ”.

But "Plus” **can't** prevent **Gradient Exploding**.

When Large weights plus large weights repeatedly, the result also be large.

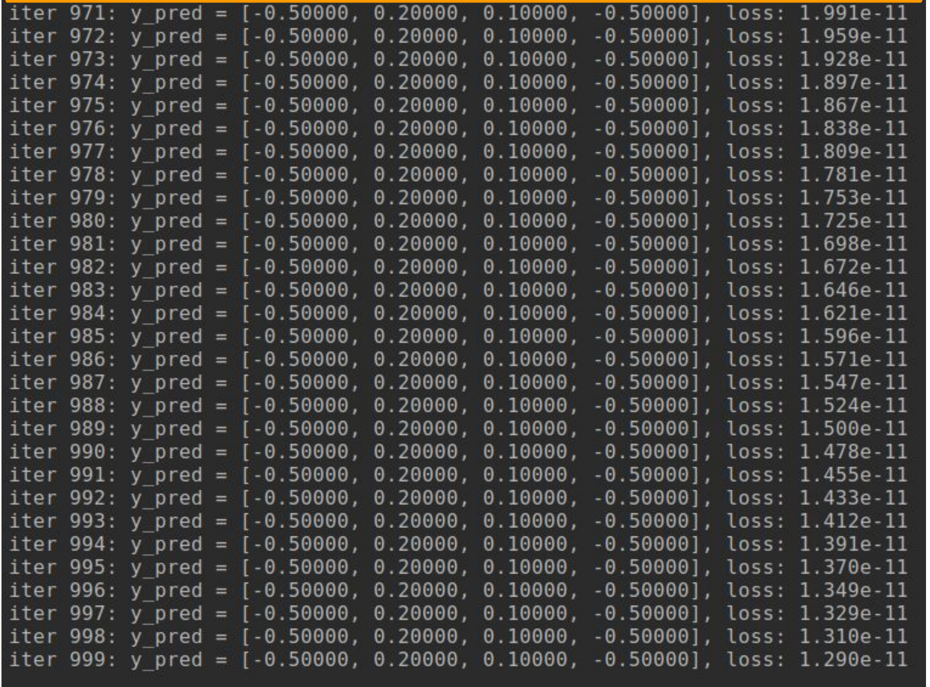
Compare Avtivation Function Variety

Proof tanh Change into ReLU

epoch = 1000
loss = Residual square
input = [-0.5, 0.2, 0.1, -0.5]
learning rate = 0.1

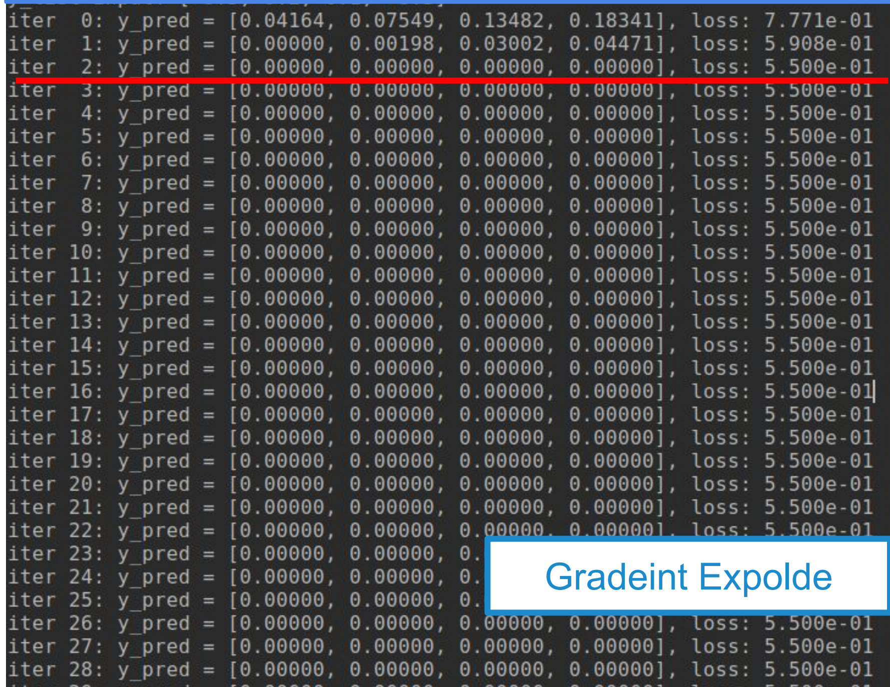
init weight = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)

Activation Function = **tanh()**



```
iter 971: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.991e-11
iter 972: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.959e-11
iter 973: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.928e-11
iter 974: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.897e-11
iter 975: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.867e-11
iter 976: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.838e-11
iter 977: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.809e-11
iter 978: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.781e-11
iter 979: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.753e-11
iter 980: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.725e-11
iter 981: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.698e-11
iter 982: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.672e-11
iter 983: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.646e-11
iter 984: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.621e-11
iter 985: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.596e-11
iter 986: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.571e-11
iter 987: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.547e-11
iter 988: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.524e-11
iter 989: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.500e-11
iter 990: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.478e-11
iter 991: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.455e-11
iter 992: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.433e-11
iter 993: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.412e-11
iter 994: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.391e-11
iter 995: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.370e-11
iter 996: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.349e-11
iter 997: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.329e-11
iter 998: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.310e-11
iter 999: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.290e-11
```

learning rate = **ReLU()**



```
iter  0: y_pred = [0.04164, 0.07549, 0.13482, 0.18341], loss: 7.771e-01
iter  1: y_pred = [0.00000, 0.00198, 0.03002, 0.04471], loss: 5.908e-01
iter  2: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter  3: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter  4: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter  5: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter  6: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter  7: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter  8: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter  9: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 10: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 11: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 12: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 13: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 14: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 15: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 16: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 17: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 18: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 19: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 20: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 21: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 22: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 23: y_pred = [0.00000, 0.00000, 0.
iter 24: y_pred = [0.00000, 0.00000, 0.
iter 25: y_pred = [0.00000, 0.00000, 0.
iter 26: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 27: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter 28: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
```

Gradeint Expolde

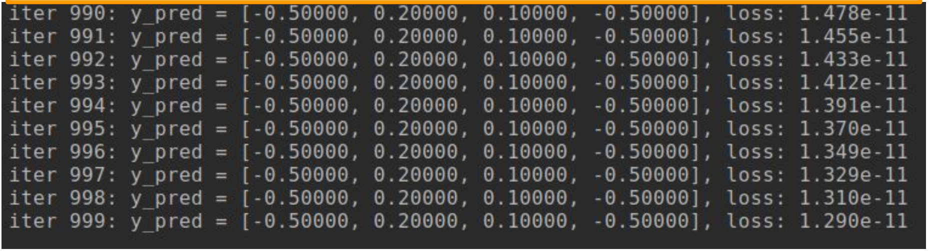
Compare Avtivation Function Variety

Proof tanh Change into ReLU

epoch = 1000
loss = Residual square
input = [-0.5, 0.2, 0.1, -0.5]
learning rate = 0.1

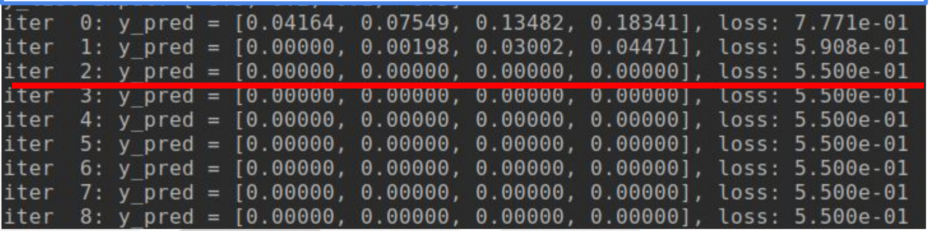
init weight = rand_arr(-0.1, 0.1, mem_cell_ct, concat_len)

Activation Function = **tanh()**



```
iter 990: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.478e-11
iter 991: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.455e-11
iter 992: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.433e-11
iter 993: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.412e-11
iter 994: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.391e-11
iter 995: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.370e-11
iter 996: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.349e-11
iter 997: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.329e-11
iter 998: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.310e-11
iter 999: y_pred = [-0.50000, 0.20000, 0.10000, -0.50000], loss: 1.290e-11
```

Activation Function = **ReLU()**



```
iter   0: y_pred = [0.04164, 0.07549, 0.13482, 0.18341], loss: 7.771e-01
iter   1: y_pred = [0.00000, 0.00198, 0.03002, 0.04471], loss: 5.908e-01
iter   2: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter   3: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter   4: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter   5: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter   6: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter   7: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
iter   8: y_pred = [0.00000, 0.00000, 0.00000, 0.00000], loss: 5.500e-01
```

The orange one shows the model can learn correctly and its loss very close to zero.

But the blue one shows the neuron was close out that can't learn anything from training .

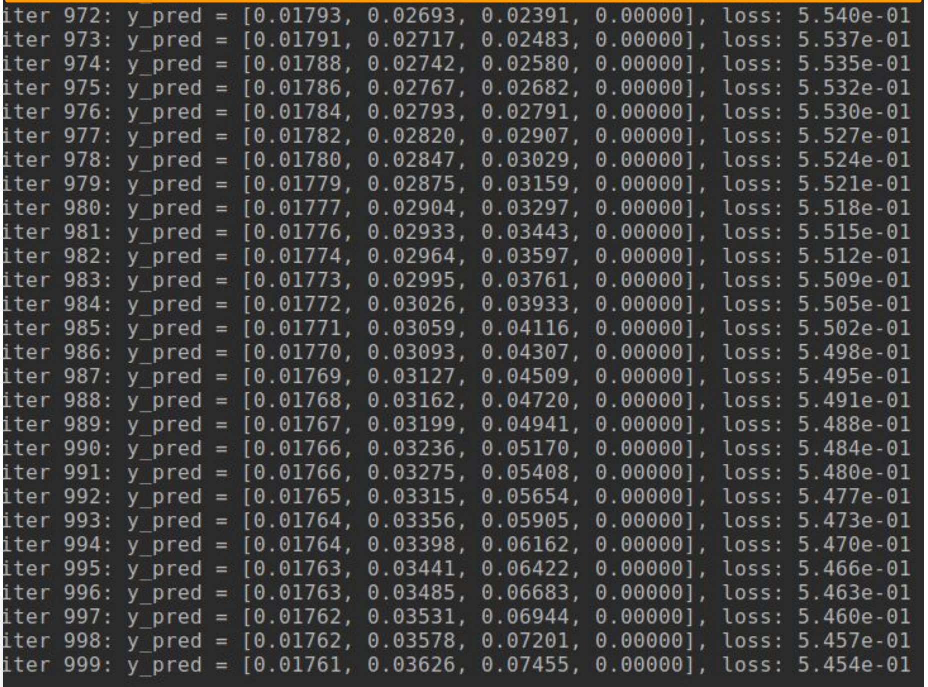
Compare Learning rate Variety

Proof Learning Rate can Slow down Gradient Explode

epoch = 1000
loss = Residual square
input = `[ -0.5,  0.2,  0.1,  -0.5 ]`

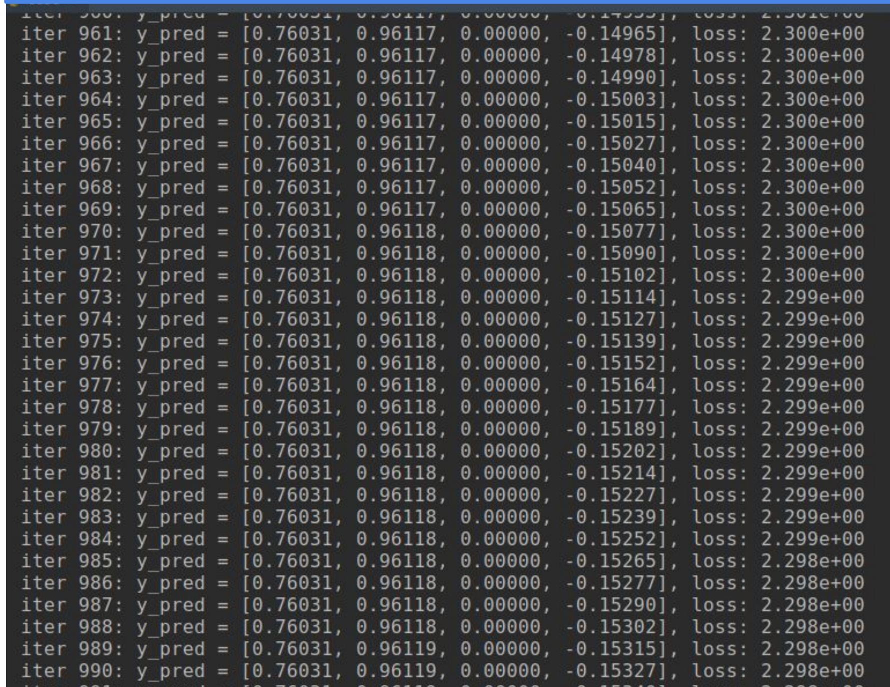
init weight = `rand_arr( -5,  5, mem_cell_ct, concat_len )`

learning rate = **0.01**



```
iter 972: y_pred = [0.01793, 0.02693, 0.02391, 0.00000], loss: 5.540e-01
iter 973: y_pred = [0.01791, 0.02717, 0.02483, 0.00000], loss: 5.537e-01
iter 974: y_pred = [0.01788, 0.02742, 0.02580, 0.00000], loss: 5.535e-01
iter 975: y_pred = [0.01786, 0.02767, 0.02682, 0.00000], loss: 5.532e-01
iter 976: y_pred = [0.01784, 0.02793, 0.02791, 0.00000], loss: 5.530e-01
iter 977: y_pred = [0.01782, 0.02820, 0.02907, 0.00000], loss: 5.527e-01
iter 978: y_pred = [0.01780, 0.02847, 0.03029, 0.00000], loss: 5.524e-01
iter 979: y_pred = [0.01779, 0.02875, 0.03159, 0.00000], loss: 5.521e-01
iter 980: y_pred = [0.01777, 0.02904, 0.03297, 0.00000], loss: 5.518e-01
iter 981: y_pred = [0.01776, 0.02933, 0.03443, 0.00000], loss: 5.515e-01
iter 982: y_pred = [0.01774, 0.02964, 0.03597, 0.00000], loss: 5.512e-01
iter 983: y_pred = [0.01773, 0.02995, 0.03761, 0.00000], loss: 5.509e-01
iter 984: y_pred = [0.01772, 0.03026, 0.03933, 0.00000], loss: 5.505e-01
iter 985: y_pred = [0.01771, 0.03059, 0.04116, 0.00000], loss: 5.502e-01
iter 986: y_pred = [0.01770, 0.03093, 0.04307, 0.00000], loss: 5.498e-01
iter 987: y_pred = [0.01769, 0.03127, 0.04509, 0.00000], loss: 5.495e-01
iter 988: y_pred = [0.01768, 0.03162, 0.04720, 0.00000], loss: 5.491e-01
iter 989: y_pred = [0.01767, 0.03199, 0.04941, 0.00000], loss: 5.488e-01
iter 990: y_pred = [0.01766, 0.03236, 0.05170, 0.00000], loss: 5.484e-01
iter 991: y_pred = [0.01766, 0.03275, 0.05408, 0.00000], loss: 5.480e-01
iter 992: y_pred = [0.01765, 0.03315, 0.05654, 0.00000], loss: 5.477e-01
iter 993: y_pred = [0.01764, 0.03356, 0.05905, 0.00000], loss: 5.473e-01
iter 994: y_pred = [0.01764, 0.03398, 0.06162, 0.00000], loss: 5.470e-01
iter 995: y_pred = [0.01763, 0.03441, 0.06422, 0.00000], loss: 5.466e-01
iter 996: y_pred = [0.01763, 0.03485, 0.06683, 0.00000], loss: 5.463e-01
iter 997: y_pred = [0.01762, 0.03531, 0.06944, 0.00000], loss: 5.460e-01
iter 998: y_pred = [0.01762, 0.03578, 0.07201, 0.00000], loss: 5.457e-01
iter 999: y_pred = [0.01761, 0.03626, 0.07455, 0.00000], loss: 5.454e-01
```

learning rate = **0.0001**



```
iter 961: y_pred = [0.76031, 0.96117, 0.00000, -0.14965], loss: 2.300e+00
iter 962: y_pred = [0.76031, 0.96117, 0.00000, -0.14978], loss: 2.300e+00
iter 963: y_pred = [0.76031, 0.96117, 0.00000, -0.14990], loss: 2.300e+00
iter 964: y_pred = [0.76031, 0.96117, 0.00000, -0.15003], loss: 2.300e+00
iter 965: y_pred = [0.76031, 0.96117, 0.00000, -0.15015], loss: 2.300e+00
iter 966: y_pred = [0.76031, 0.96117, 0.00000, -0.15027], loss: 2.300e+00
iter 967: y_pred = [0.76031, 0.96117, 0.00000, -0.15040], loss: 2.300e+00
iter 968: y_pred = [0.76031, 0.96117, 0.00000, -0.15052], loss: 2.300e+00
iter 969: y_pred = [0.76031, 0.96117, 0.00000, -0.15065], loss: 2.300e+00
iter 970: y_pred = [0.76031, 0.96118, 0.00000, -0.15077], loss: 2.300e+00
iter 971: y_pred = [0.76031, 0.96118, 0.00000, -0.15090], loss: 2.300e+00
iter 972: y_pred = [0.76031, 0.96118, 0.00000, -0.15102], loss: 2.300e+00
iter 973: y_pred = [0.76031, 0.96118, 0.00000, -0.15114], loss: 2.299e+00
iter 974: y_pred = [0.76031, 0.96118, 0.00000, -0.15127], loss: 2.299e+00
iter 975: y_pred = [0.76031, 0.96118, 0.00000, -0.15139], loss: 2.299e+00
iter 976: y_pred = [0.76031, 0.96118, 0.00000, -0.15152], loss: 2.299e+00
iter 977: y_pred = [0.76031, 0.96118, 0.00000, -0.15164], loss: 2.299e+00
iter 978: y_pred = [0.76031, 0.96118, 0.00000, -0.15177], loss: 2.299e+00
iter 979: y_pred = [0.76031, 0.96118, 0.00000, -0.15189], loss: 2.299e+00
iter 980: y_pred = [0.76031, 0.96118, 0.00000, -0.15202], loss: 2.299e+00
iter 981: y_pred = [0.76031, 0.96118, 0.00000, -0.15214], loss: 2.299e+00
iter 982: y_pred = [0.76031, 0.96118, 0.00000, -0.15227], loss: 2.299e+00
iter 983: y_pred = [0.76031, 0.96118, 0.00000, -0.15239], loss: 2.299e+00
iter 984: y_pred = [0.76031, 0.96118, 0.00000, -0.15252], loss: 2.299e+00
iter 985: y_pred = [0.76031, 0.96118, 0.00000, -0.15265], loss: 2.298e+00
iter 986: y_pred = [0.76031, 0.96118, 0.00000, -0.15277], loss: 2.298e+00
iter 987: y_pred = [0.76031, 0.96118, 0.00000, -0.15290], loss: 2.298e+00
iter 988: y_pred = [0.76031, 0.96118, 0.00000, -0.15302], loss: 2.298e+00
iter 989: y_pred = [0.76031, 0.96119, 0.00000, -0.15315], loss: 2.298e+00
iter 990: y_pred = [0.76031, 0.96119, 0.00000, -0.15327], loss: 2.298e+00
```

Compare Learning rate Variety

Proof Learning Rate can Slow down Gradient Explode

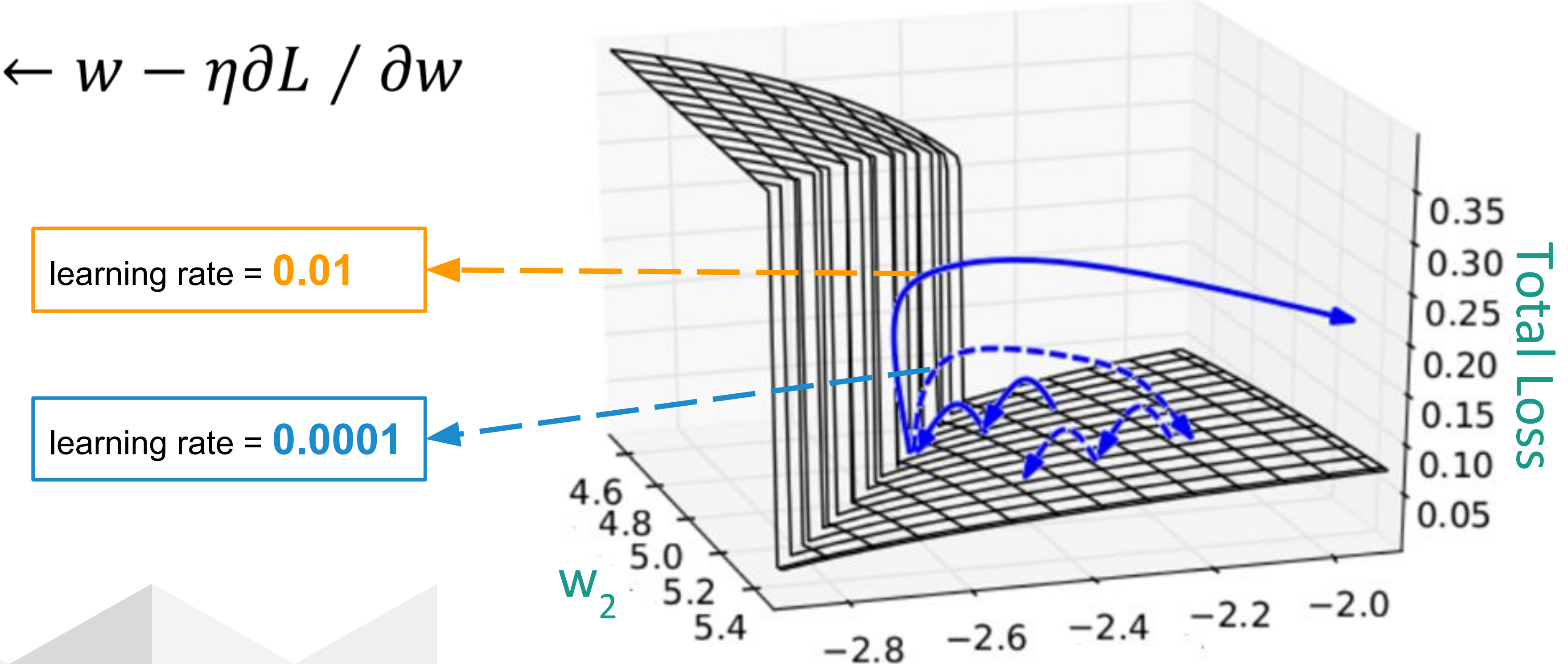
epoch = 1000

loss = Residual square

input = `[ -0.5, 0.2, 0.1, -0.5 ]`

init weight = `rand_arr( -5, 5, mem_cell_ct, concat_len )`

$$w \leftarrow w - \eta \partial L \;/\; \partial w$$



learning rate = **0.01**

learning rate = **0.0001**