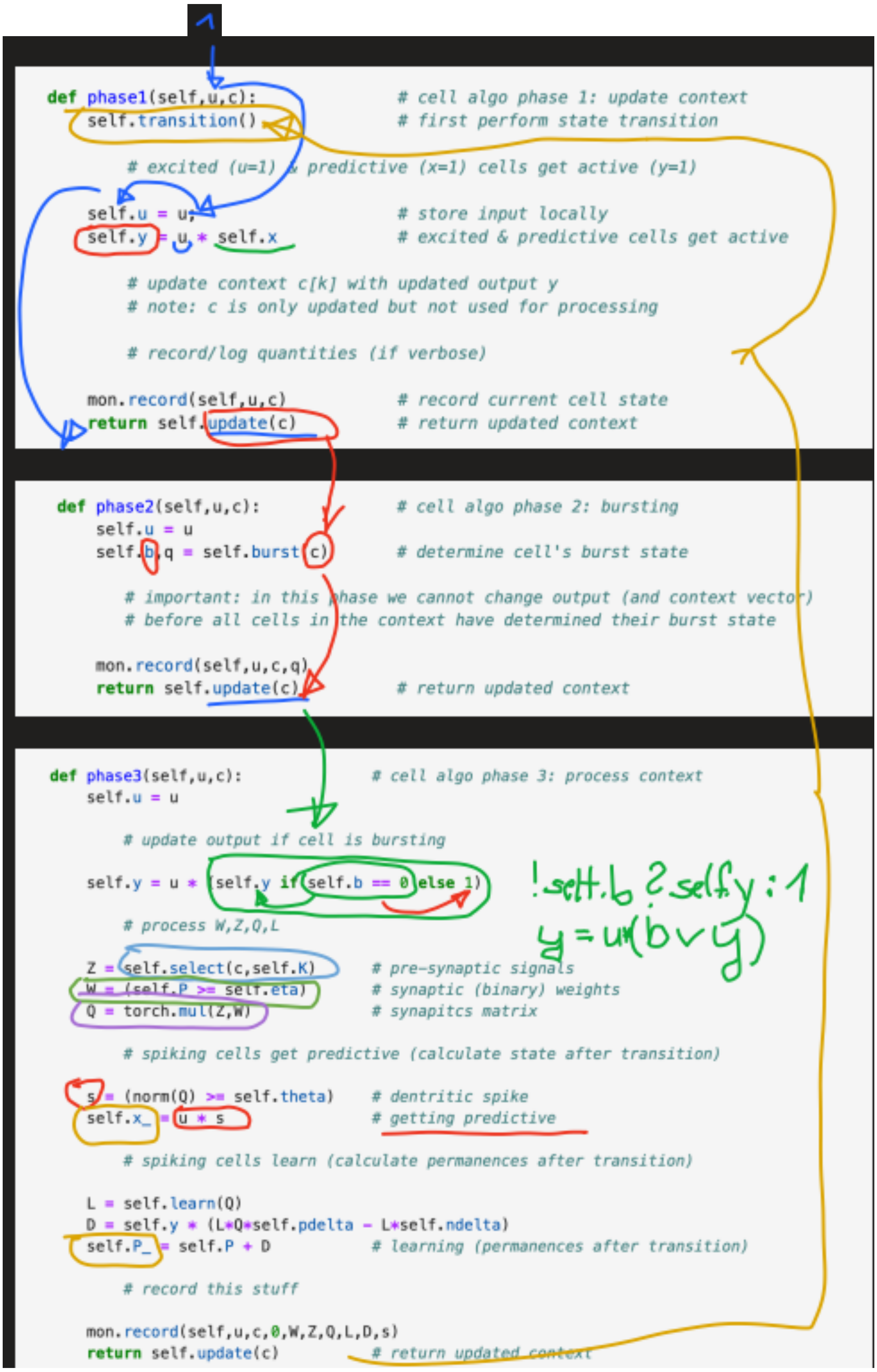
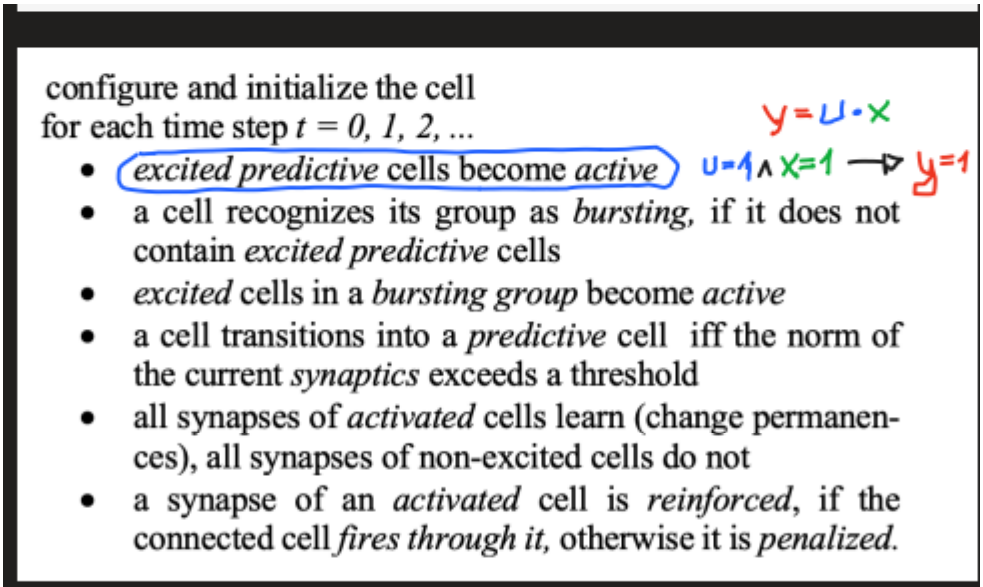
Betreff: 3 Phases
Datum: Samstag, 21. Oktober 2023 um 21:04:41 Mitteleuropäische Sommerzeit
Von: Hugo Pristauz
An: emv@fnet.cc, Walter Eder, Hugo Pristauz (Bluenetics)
Anlagen: PNG-Bild, PNG-Bild[9], PNG-Bild[80], PNG-Bild[81], PNG-Bild[55], PNG-Bild[5], PNG-Bild[40], PNG-Bild[34], PNG-Bild[97], PNG-Bild[59], image001.png

3 Phases

Samstag, 21. Oktober 2023
18:31



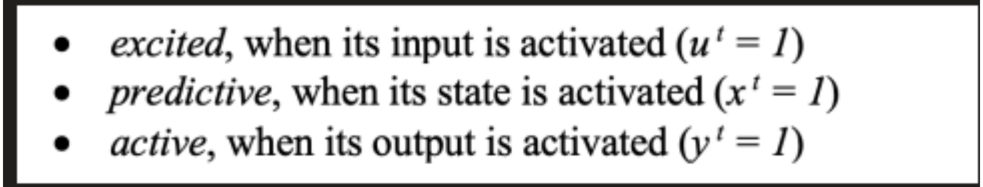
```
def phase1(self,u,c):               # cell algo phase 1: update context
    self.transition()               # first perform state transition

      # excited (u=1) & predictive (x=1) cells get active (y=1)

    self.u = u;                     # store input locally
    self.y = u * self.x             # excited & predictive cells get active

      # update context c[k] with updated output y
      # note: c is only updated but not used for processing

      # record/log quantities (if verbose)

    mon.record(self,u,c)            # record current cell state
    return self.update(c)           # return updated context

def phase2(self,u,c):               # cell algo phase 2: bursting
    self.u = u
    self.b,q = self.burst(c)        # determine cell's burst state

      # important: in this phase we cannot change output (and context vector)
      # before all cells in the context have determined their burst state

    mon.record(self,u,c,q)
    return self.update(c)           # return updated context

def phase3(self,u,c):               # cell algo phase 3: process context
    self.u = u

      # update output if cell is bursting

    self.y = u * (self.y if self.b == 0 else 1)

      # process W,Z,Q,L

    Z = self.select(c,self.K)       # pre-synaptic signals
    W = (self.P >= self.eta)        # synaptic (binary) weights
    Q = torch.mul(Z,W)              # synapitcs matrix

      # spiking cells get predictive (calculate state after transition)

    s = (norm(Q) >= self.theta)     # dentritic spike
    self.x_ = u * s                 # getting predictive

      # spiking cells learn (calculate permanences after transition)

    L = self.learn(Q)
    D = self.y * (L*Q*self.pdelta - L*self.ndelta)
    self.P_ = self.P + D            # learning (permanences after transition)

      # record this stuff

    mon.record(self,u,c,0,W,Z,Q,L,D,s)
    return self.update(c)           # return updated context
```



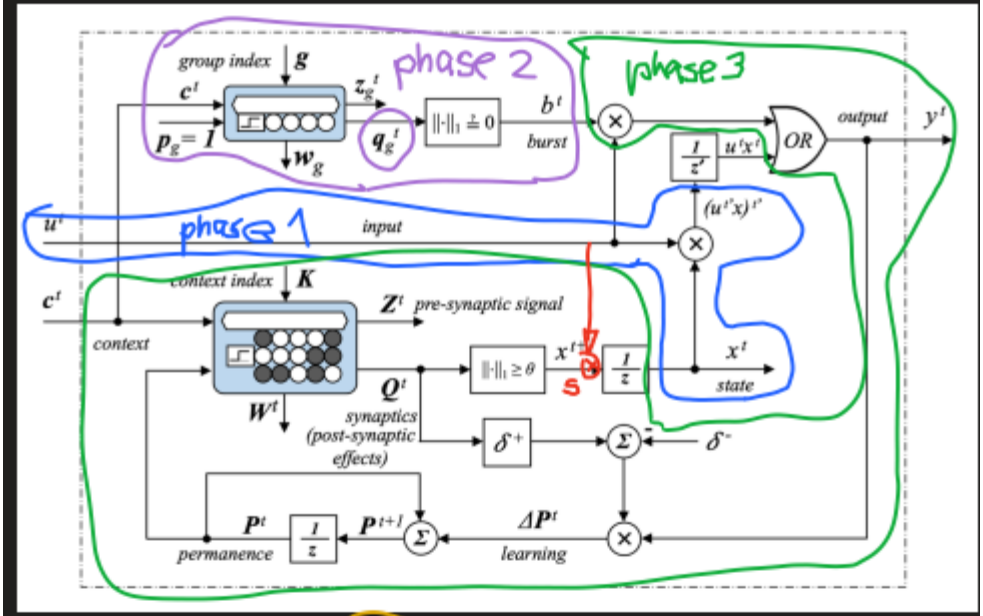
configure and initialize the cell
for each time step $t = 0, 1, 2, \ldots$

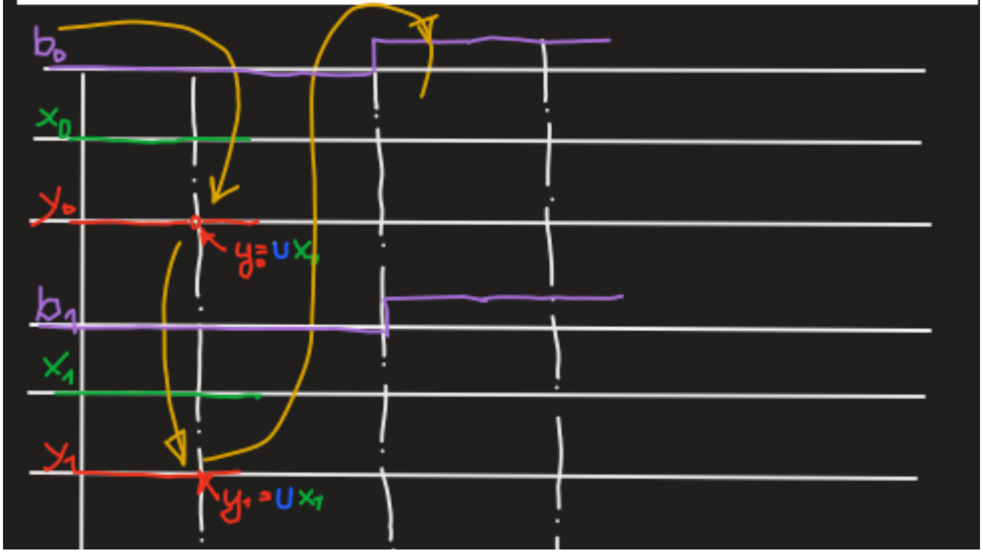
$y = u \cdot x$

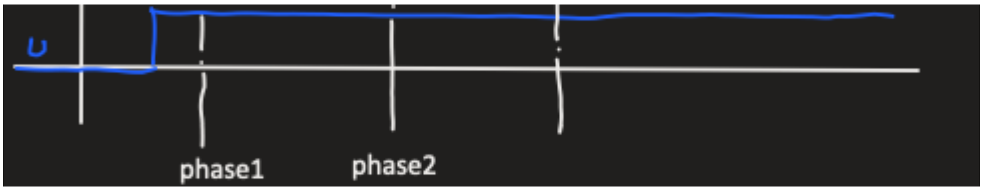
- *excited predictive* cells become *active* $u=1 \wedge x=1 \rightarrow y=1$
- a cell recognizes its group as *bursting*, if it does not contain *excited predictive* cells
- *excited* cells in a *bursting group* become *active*
- a cell transitions into a *predictive* cell iff the norm of the current *synaptics* exceeds a threshold
- all synapses of *activated* cells learn (change permanences), all synapses of non-excited cells do not
- a synapse of an *activated* cell is *reinforced*, if the connected cell *fires through it*, otherwise it is *penalized*.

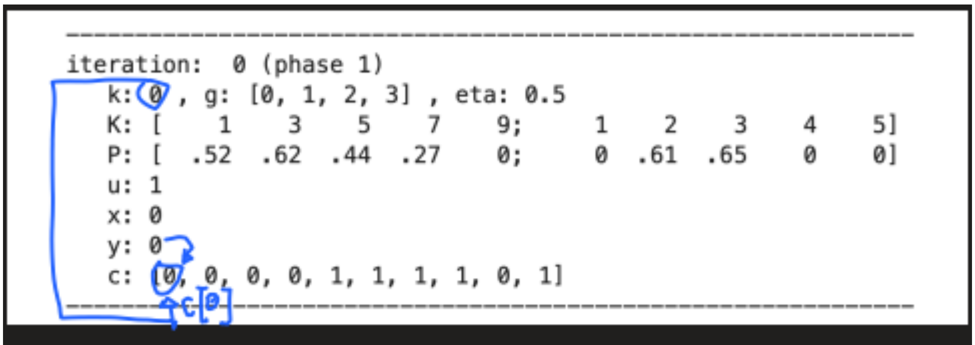


- *excited*, when its input is activated ($u^t = 1$)
- *predictive*, when its state is activated ($x^t = 1$)
- *active*, when its output is activated ($y^t = 1$)



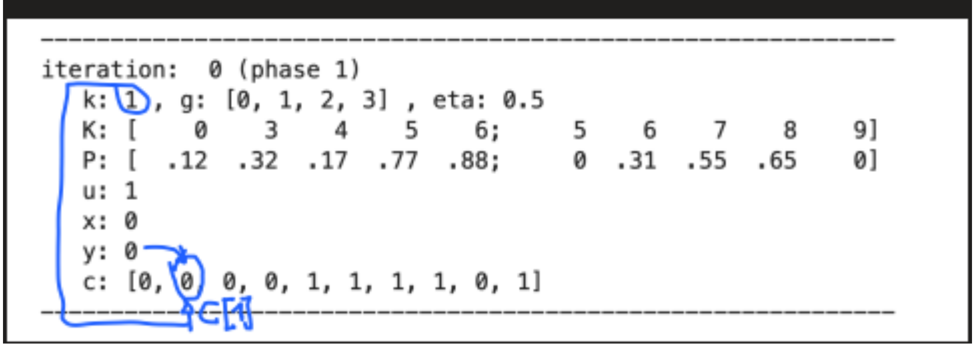






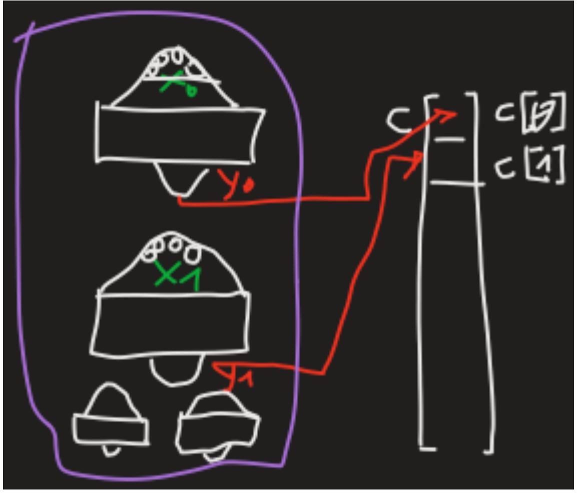
```
-----------------------------------------------------------------
iteration:  0 (phase 1)
   k: 0 , g: [0, 1, 2, 3] , eta: 0.5
   K: [   1    3    5    7    9;     1    2    3    4    5]
   P: [  .52  .62  .44  .27   0;     0  .61  .65    0    0]
   u: 1
   x: 0
   y: 0
   c: [0, 0, 0, 0, 1, 1, 1, 1, 0, 1]
-----------------------------------------------------------------
```

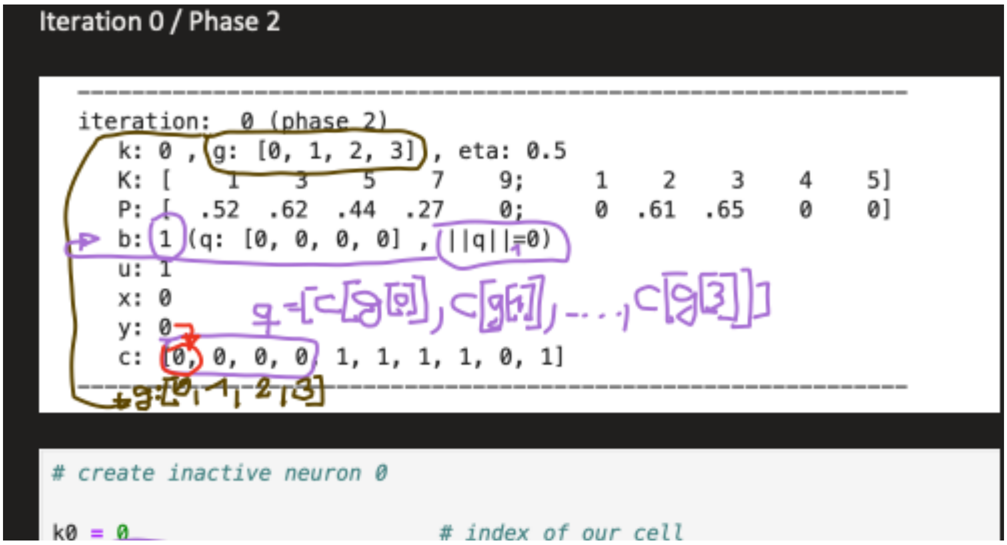
c[0]



```
-----------------------------------------------------------------
iteration:  0 (phase 1)
   k: 1 , g: [0, 1, 2, 3] , eta: 0.5
   K: [   0    3    4    5    6;     5    6    7    8    9]
   P: [  .12  .32  .17  .77  .88;    0  .31  .55  .65    0]
   u: 1
   x: 0
   y: 0
   c: [0, 0, 0, 0, 1, 1, 1, 1, 0, 1]
-----------------------------------------------------------------
```

c[1]





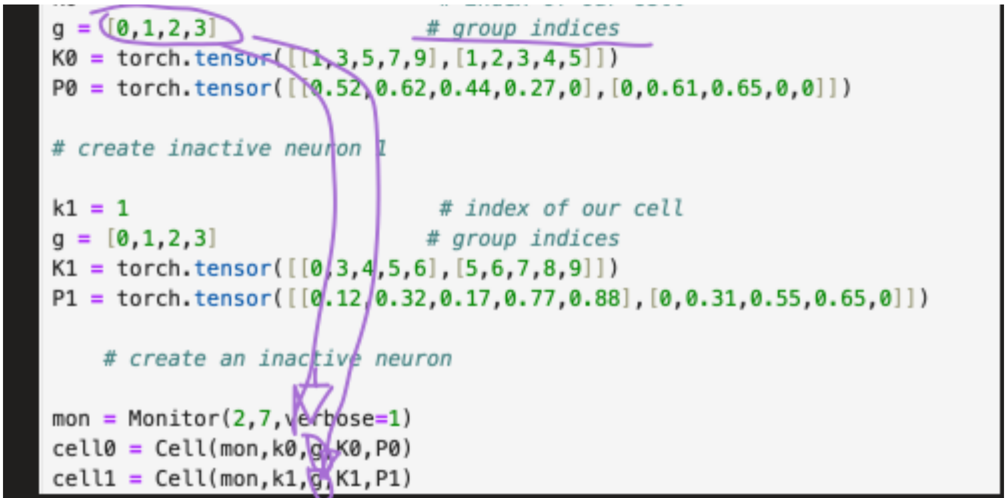
Iteration 0 / Phase 2

```
-----------------------------------------------------------------
iteration:  0 (phase 2)
   k: 0 , g: [0, 1, 2, 3] , eta: 0.5
   K: [   1    3    5    7    9;     1    2    3    4    5]
   P: [  .52  .62  .44  .27   0;     0  .61  .65    0    0]
   b: 1 (q: [0, 0, 0, 0] , ||q||=0)
   u: 1
   x: 0
   y: 0
   c: [0, 0, 0, 0, 1, 1, 1, 1, 0, 1]
-----------------------------------------------------------------
```

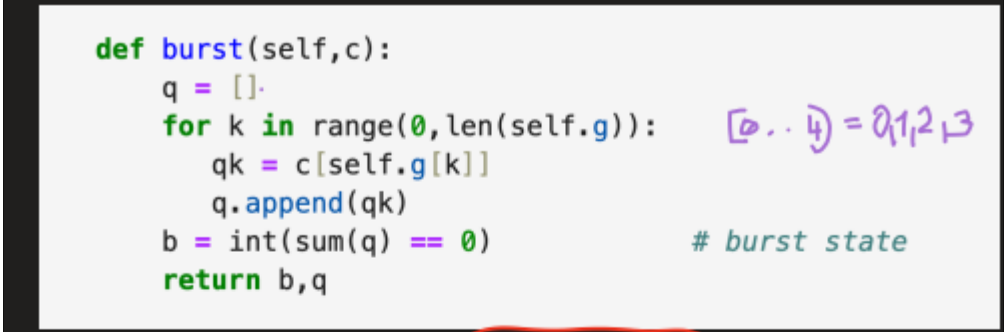
q=[c[g[0]], c[g[1]], ..., c[g[3]]]

g:[0, 1, 2, 3]

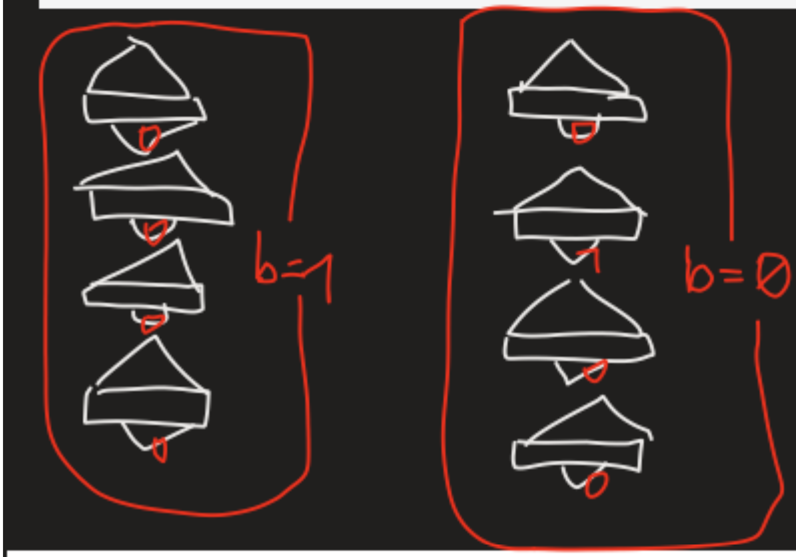
```
# create inactive neuron 0

k0 = 0                          # index of our cell
```

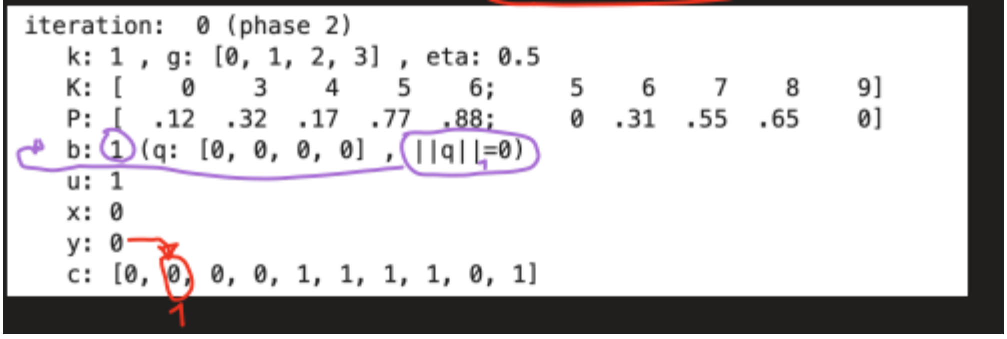


```
g = [0,1,2,3]                         # group indices
K0 = torch.tensor([[1,3,5,7,9],[1,2,3,4,5]])
P0 = torch.tensor([[0.52,0.62,0.44,0.27,0],[0,0.61,0.65,0,0]])

# create inactive neuron 1

k1 = 1                                 # index of our cell
g = [0,1,2,3]                         # group indices
K1 = torch.tensor([[0,3,4,5,6],[5,6,7,8,9]])
P1 = torch.tensor([[0.12,0.32,0.17,0.77,0.88],[0,0.31,0.55,0.65,0]])

    # create an inactive neuron

mon = Monitor(2,7,verbose=1)
cell0 = Cell(mon,k0,g,K0,P0)
cell1 = Cell(mon,k1,g,K1,P1)
```



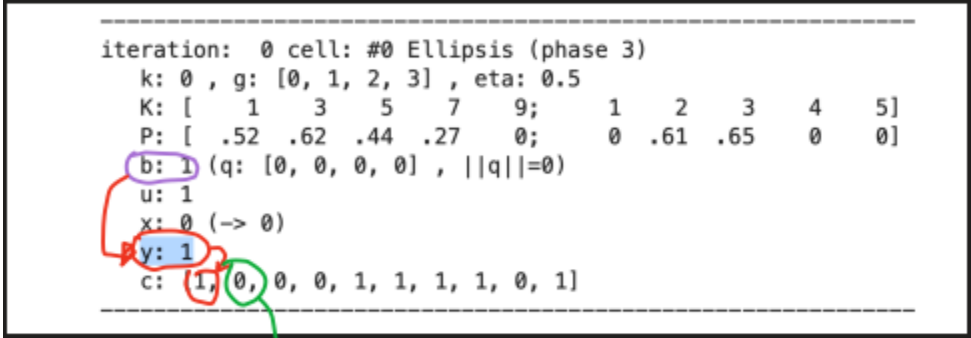
```
    def burst(self,c):
        q = []
        for k in range(0,len(self.g)):        [0..4) = 0,1,2,3
            qk = c[self.g[k]]
            q.append(qk)
        b = int(sum(q) == 0)              # burst state
        return b,q
```



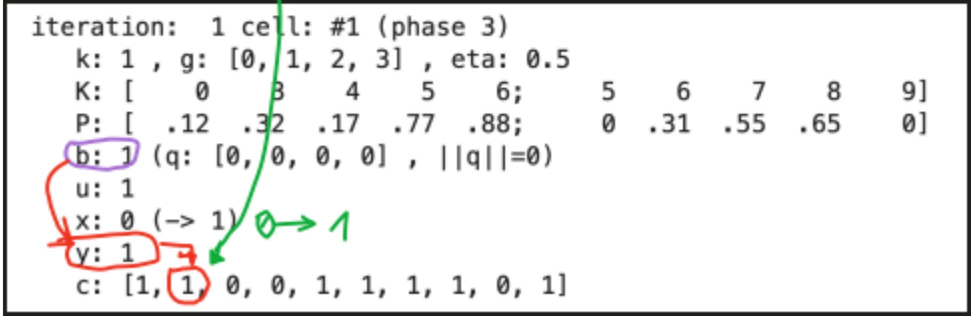


```
iteration:  0 (phase 2)
   k: 1 , g: [0, 1, 2, 3] , eta: 0.5
   K: [    0    3    4    5    6;     5    6    7    8    9]
   P: [  .12  .32  .17  .77  .88;     0  .31  .55  .65    0]
   b: 1 (q: [0, 0, 0, 0] , ||q||=0)
   u: 1
   x: 0
   y: 0
   c: [0, 0, 0, 0, 1, 1, 1, 1, 0, 1]
           1
```

iteration 0 / phase 3



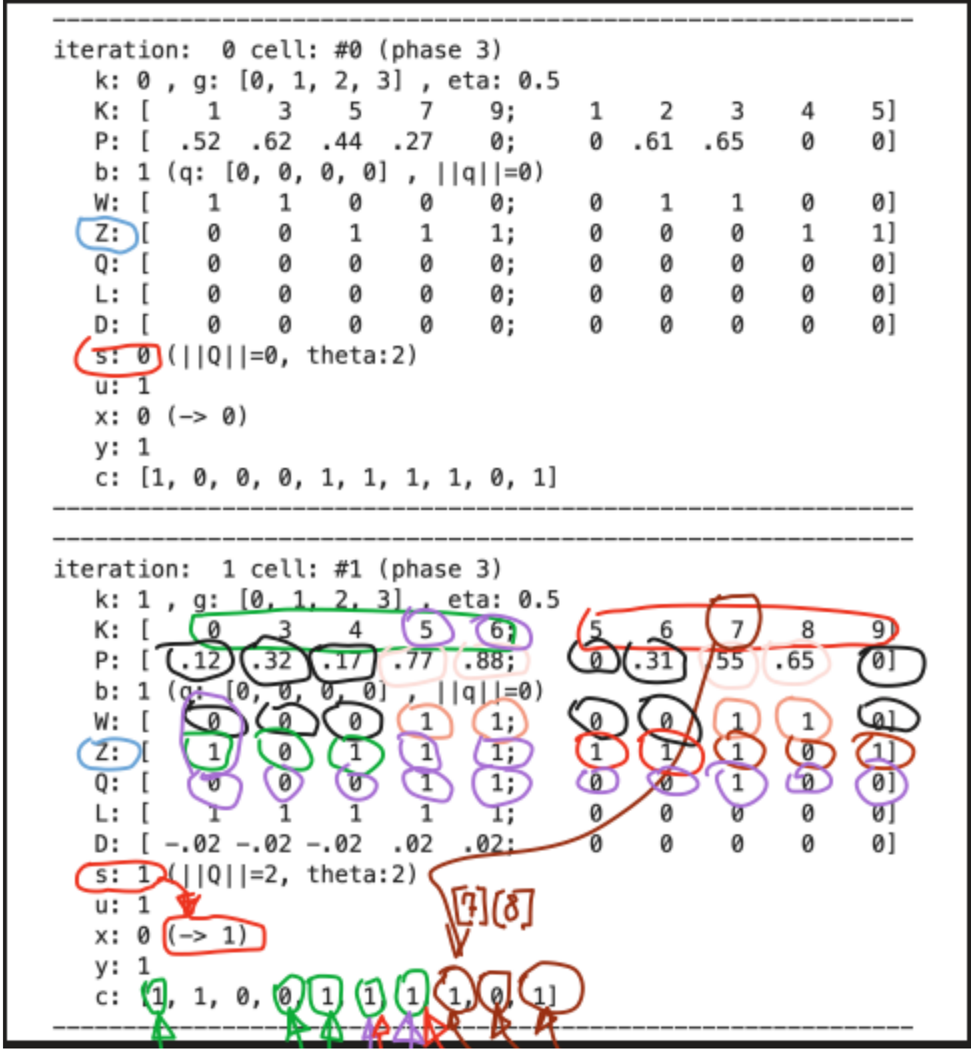
```
------------------------------------------------------------------
iteration:  0 cell: #0 Ellipsis (phase 3)
   k: 0 , g: [0, 1, 2, 3] , eta: 0.5
   K: [    1    3    5    7    9;     1    2    3    4    5]
   P: [  .52  .62  .44  .27    0;     0  .61  .65    0    0]
   b: 1 (q: [0, 0, 0, 0] , ||q||=0)
   u: 1
   x: 0 (-> 0)
   y: 1
   c: [1, 0, 0, 0, 1, 1, 1, 1, 0, 1]
------------------------------------------------------------------
```



```
iteration:  1 cell: #1 (phase 3)
   k: 1 , g: [0, 1, 2, 3] , eta: 0.5
   K: [    0    3    4    5    6;     5    6    7    8    9]
   P: [  .12  .32  .17  .77  .88;     0  .31  .55  .65    0]
   b: 1 (q: [0, 0, 0, 0] , ||q||=0)
   u: 1
   x: 0 (-> 1)
   y: 1
   c: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1]
```

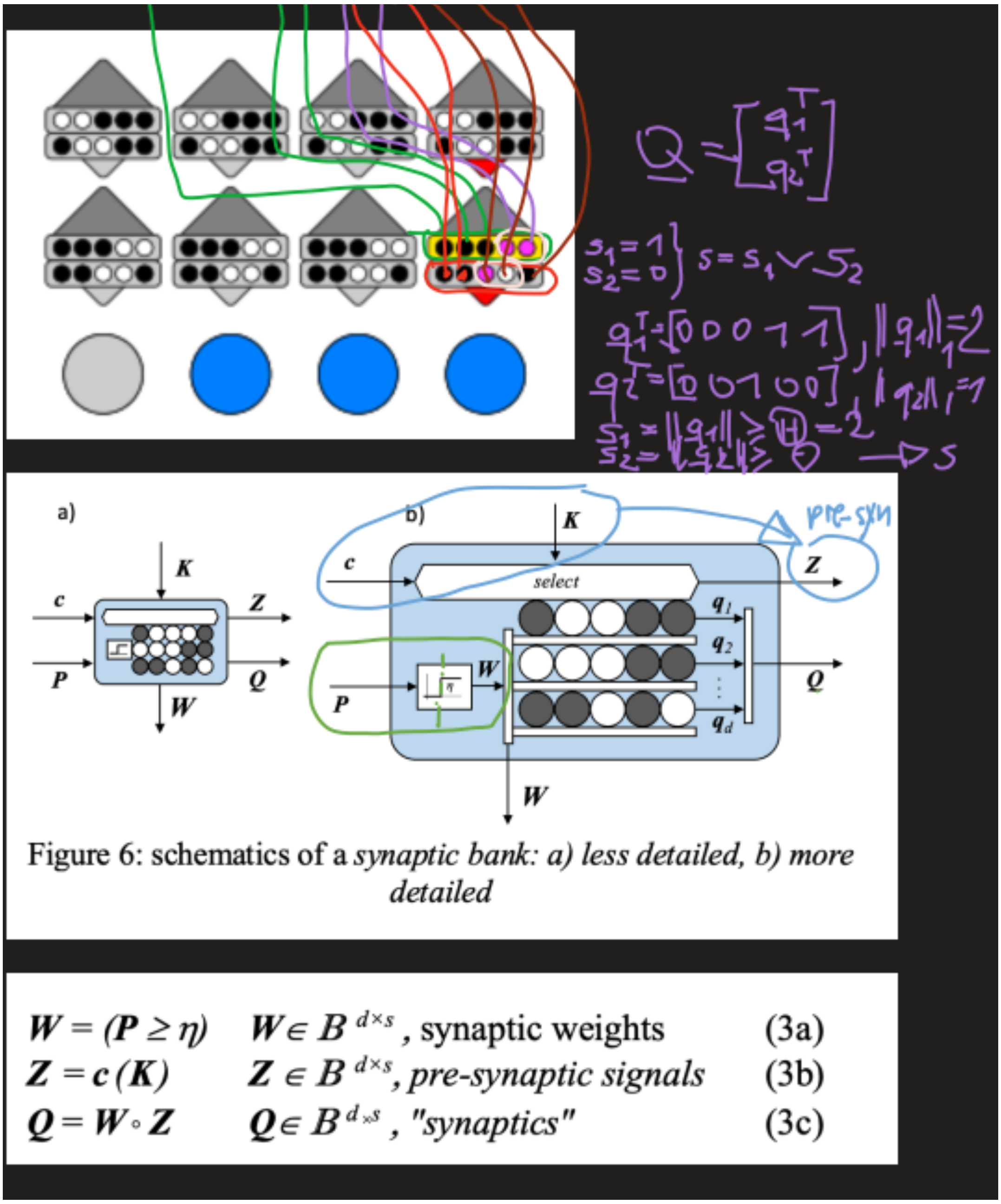
0 —> 1

process W,Q,Z,L



```
------------------------------------------------------------------
iteration:  0 cell: #0 (phase 3)
   k: 0 , g: [0, 1, 2, 3] , eta: 0.5
   K: [    1    3    5    7    9;     1    2    3    4    5]
   P: [  .52  .62  .44  .27    0;     0  .61  .65    0    0]
   b: 1 (q: [0, 0, 0, 0] , ||q||=0)
   W: [    1    1    0    0    0;     0    1    1    0    0]
   Z: [    0    0    1    1    1;     0    0    0    1    1]
   Q: [    0    0    0    0    0;     0    0    0    0    0]
   L: [    0    0    0    0    0;     0    0    0    0    0]
   D: [    0    0    0    0    0;     0    0    0    0    0]
   s: 0 (||Q||=0, theta:2)
   u: 1
   x: 0 (-> 0)
   y: 1
   c: [1, 0, 0, 0, 1, 1, 1, 1, 0, 1]
------------------------------------------------------------------
------------------------------------------------------------------
iteration:  1 cell: #1 (phase 3)
   k: 1 , g: [0, 1, 2, 3] , eta: 0.5
   K: [    0    3    4    5    6;     5    6    7    8    9]
   P: [  .12  .32  .17  .77  .88;     0  .31  .55  .65    0]
   b: 1 (q: [0, 0, 0, 0] , ||q||=0)
   W: [    0    0    0    1    1;     0    0    1    1    0]
   Z: [    1    0    1    1    1;     1    1    1    0    1]
   Q: [    0    0    0    1    1;     0    0    1    0    0]
   L: [    1    1    1    1    1;     0    0    0    0    0]
   D: [ -.02 -.02 -.02  .02  .02;     0    0    0    0    0]
   s: 1 (||Q||=2, theta:2)
   u: 1
   x: 0 (-> 1)
   y: 1
   c: [1, 1, 0, 0, 1, 1, 1, 1, 0, 1]
------------------------------------------------------------------
```

[7][8]



$$\underline{Q} = \begin{bmatrix} \underline{q}_1^T \\ \underline{q}_2^T \end{bmatrix}$$

$$\left.\begin{matrix} s_1 = 1 \\ s_2 = 0 \end{matrix}\right\} s = s_1 \vee s_2$$

$$\underline{q}_1^T = [0\ 0\ 0\ 1\ 1], \ \|\underline{q}_1\|_1 = 2$$

$$\underline{q}_2^T = [0\ 0\ 1\ 0\ 0], \ \|q_2\|_1 = 1$$

$$\begin{matrix} s_1 = \|\underline{q}_1\| \geq \boxed{H} = 2 \\ s_2 = \|\underline{q}_2\| \geq \boxed{H} \end{matrix} \longrightarrow s$$

Figure 6: schematics of a *synaptic bank: a) less detailed, b) more detailed*

$\boldsymbol{W} = (\boldsymbol{P} \geq \eta)$	$\boldsymbol{W} \in B^{d \times s}$, synaptic weights	(3a)
$\boldsymbol{Z} = \boldsymbol{c}(\boldsymbol{K})$	$\boldsymbol{Z} \in B^{d \times s}$, *pre-synaptic signals*	(3b)
$\boldsymbol{Q} = \boldsymbol{W} \circ \boldsymbol{Z}$	$\boldsymbol{Q} \in B^{d \times s}$, *"synaptics"*	(3c)

Erstellt mit OneNote.

Gruesse / servus / regards

bluenetics

Hugo Pristauz
CTO & Co-Founder Bluenetics
Mobile: +43 650 730 3003
LinkedIn: bit.ly/hugopristauz

UID: ATU 7320 4819 I FN: 489296d I FB-Gericht: Innsbruck I Sitz: A-6112 Wattens