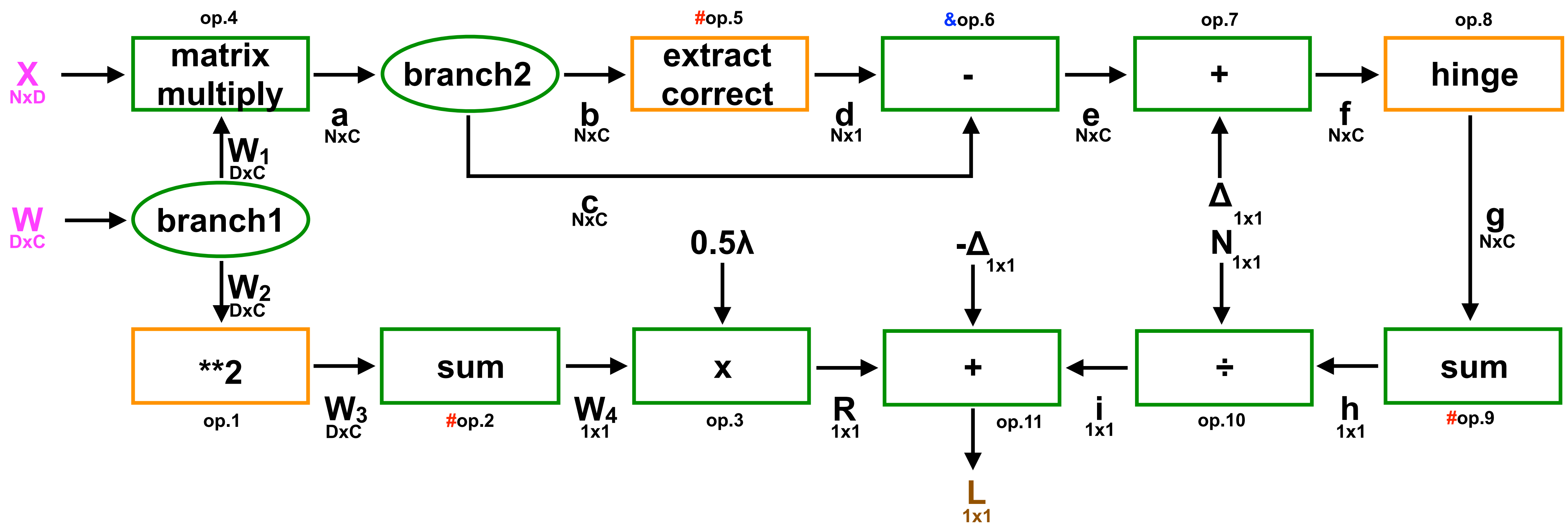
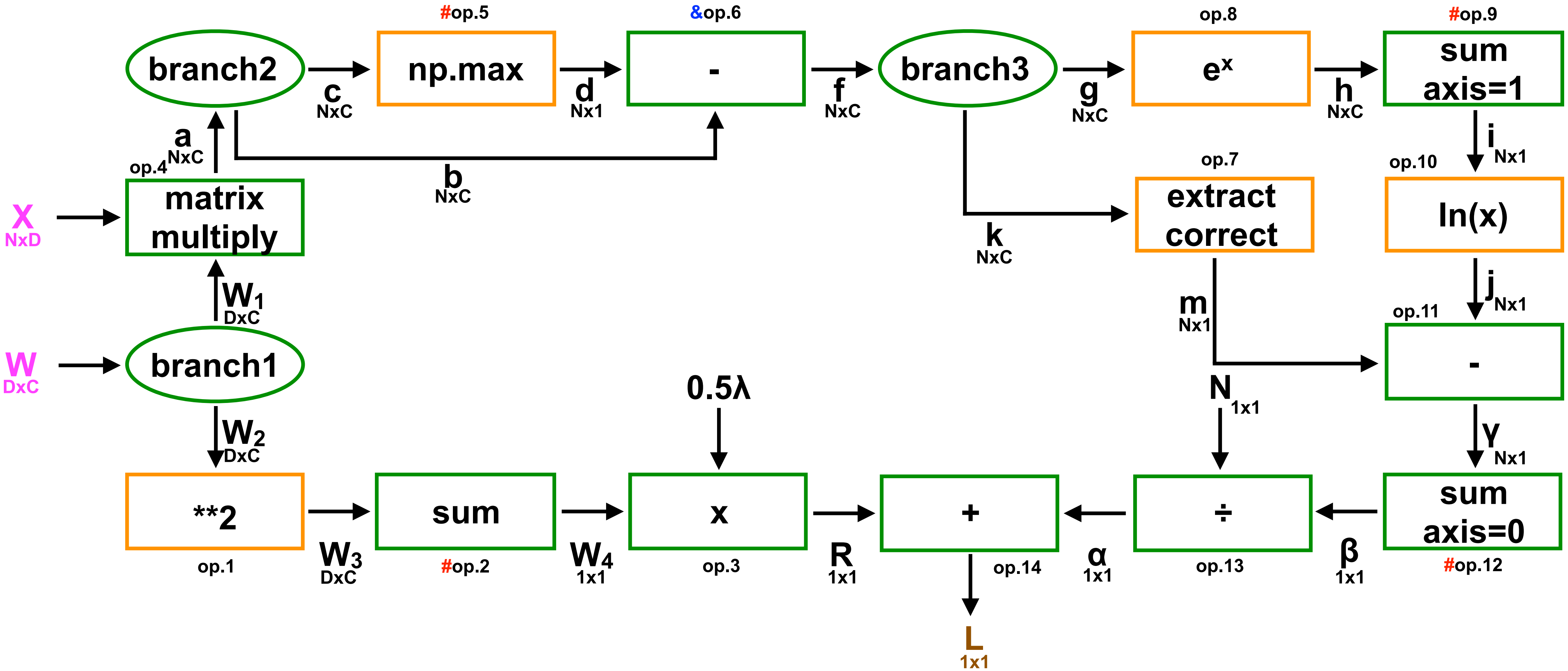
SVM



	FORWARD PROP	BACK PROP	
branch1	: $W_1 = W_2 = W$	$dW = dW_1+dW_2$	
op. 1	: $W_3 = W_2$**2	$dW_2 = dW_3$*$2W_2$	
# op. 2	: W_4 = np.sum(W_3)	$dW_3 = dW_4$*$1_{(D,C)}$	
op. 3	: $R = 0.5\lambda W_4$	dW_4 = dR*0.5λ	
op. 4	: $a = X.W_1$	$dX = da.W_1^T$	$dW_1 = X^T.da$
branch2	: b = c = a	da = db+dc	
# op. 5	: d = b[np.arange(N),y] -> d = d.reshape((N,-1))	dddb = $0_{(N,C)}$ -> dddb[np.arange(N),y] = 1.0 -> db = dd*dddb	
& op. 6	: e = c-d	dc = de	dd = -de.$1_{(C,1)}$
op. 7	: f = e+Δ	de = df	
op. 8	: g = np.maximum(0,f)	dgdf = np.greater(f,0) -> df = dg*dgdf	
# op. 9	: h = np.sum(g)	dg = dh*$1_{(N,C)}$	
op. 10	: i = h/N	dh = di/N	
op. 11	: L = i+R+Δ	dR = 1	di = 1

- # ... dimensional reduction
- & ... broadcasting
- green ... linear operations
- orange ... non-linear operations
- pink ... inputs needing gradients
- brown ... outputs

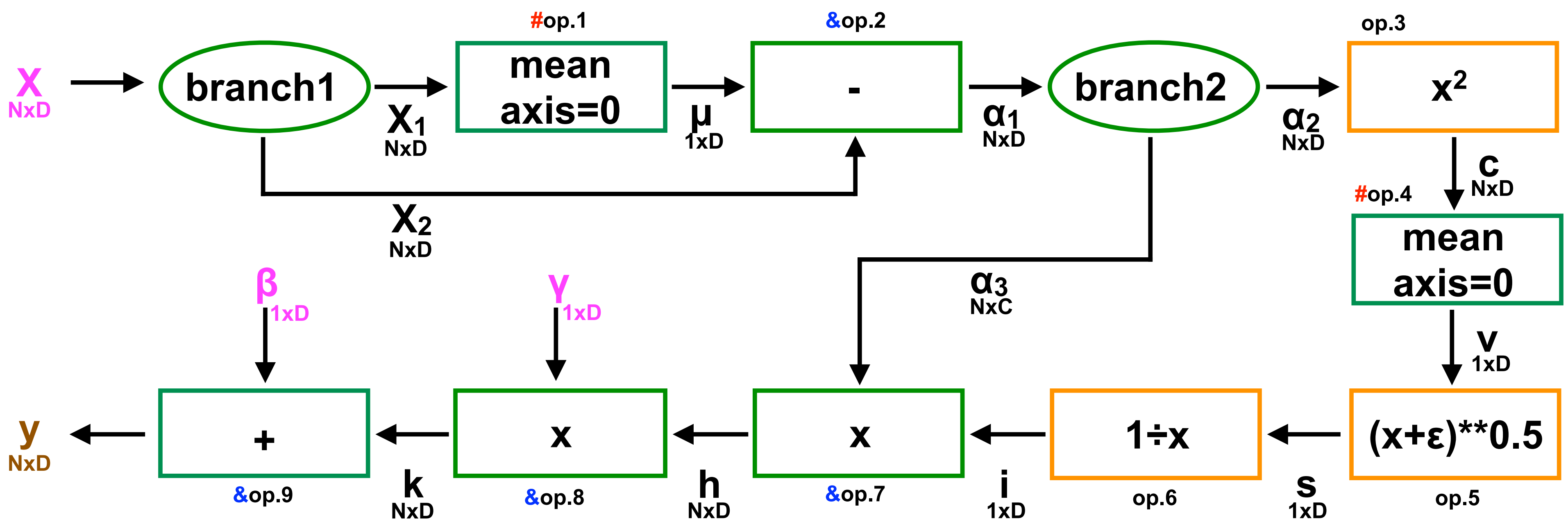
softmax



FORWARD PROP		BACK PROP	
branch1	: $W_1 = W_2 = W$	$dW = dW_1+dW_2$	
op. 1	: $W_3 = W_2$**2	$dW_2 = dW_3*2W_2$	
# op. 2	: W_4 = np.sum(W_3)	$dW_3 = dW_4*1_{(D,C)}$	
op. 3	: $R = 0.5\lambda W_4$	$dW_4 = dR*0.5\lambda$	
op. 4	: $a = X.W_1$	$dX = da.W_1^T$	$dW_1=X^T.da$
branch2	: b = c = a	da = db+dc	
# op. 5	: d = np.max(c)	dddc = $0_{(N*C)}$ -> dddc[np.argmax(c)] = 1.0 -> dddc=dddc.reshape((N,-1)) -> dc=dd*dddc	
& op. 6	: f = b-d	dd = $-1_{(1,N)}$.df.$1_{(C,1)}$	db=df
branch3	: k = g = f	df = dk+dg	
op. 7	: m = k[np.arange(N),y] -> m = m.reshape((N,-1))	dmdk = $0_{(N,C)}$ -> dmdk[np.arange(N),y]=1.0 -> dk=dm*dmdk	
op. 8	: h = exp(g)	dg = dh*exp(g)	
# op. 9	: i = np.sum(h, axis=1)	dh = di*$1_{(N,C)}$	
op. 10	: j = ln(i)	di = dj/i	
op. 11	: γ = j-m	dm = -dγ	dj=dγ
# op. 12	: β = np.sum(γ)	dγ = dβ*$1_{(N,1)}$	
op. 13	: $\alpha = \beta/N$	dβ = dα/N	
op. 14	: $L = R+\alpha$	dR = 1	dα=1

... dimensional reduction
& ... broadcasting
green ... linear operations
orange ... non-linear operations
pink ... inputs needing gradients
brown ... outputs

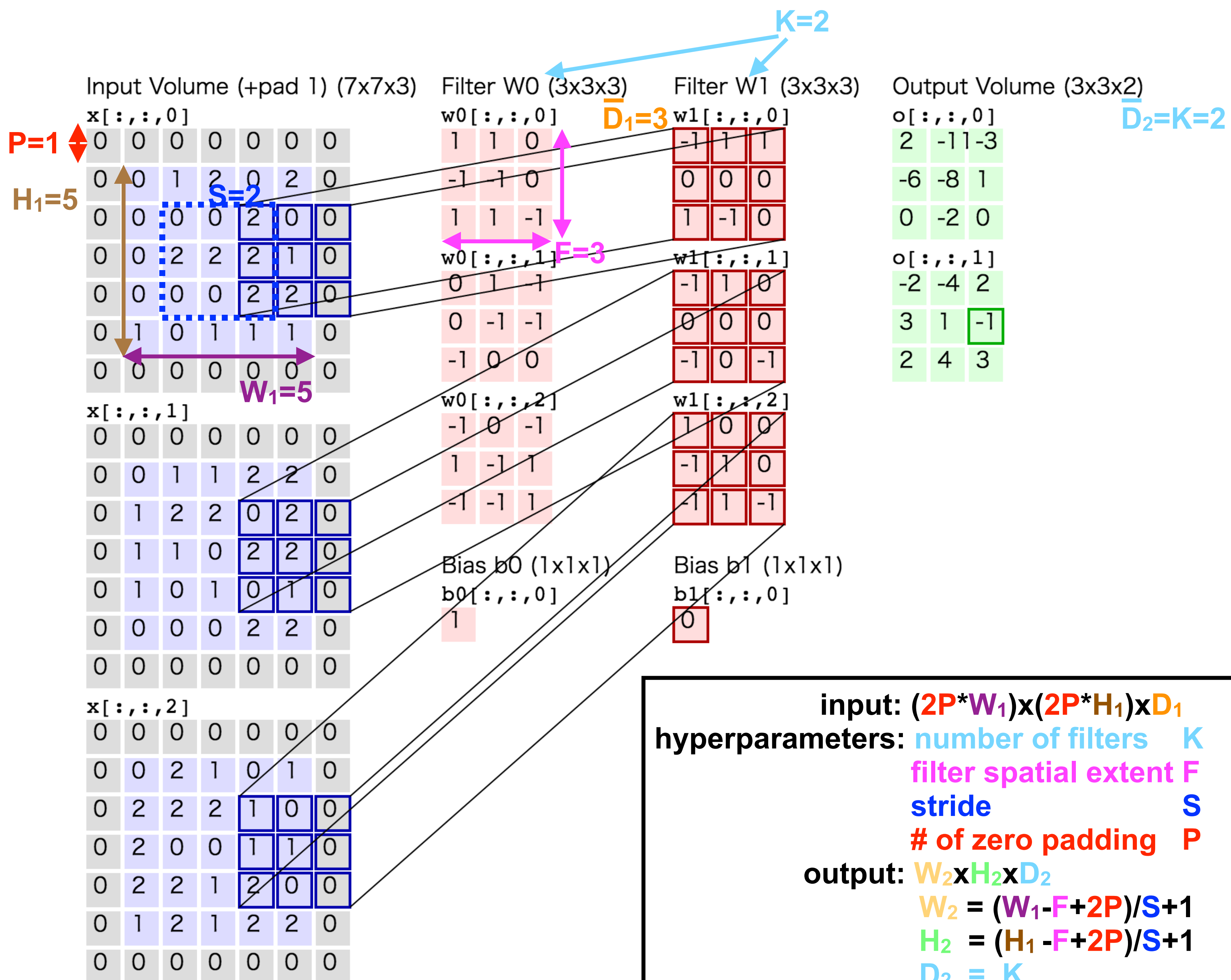
batch norm



		FORWARD PROP	BACK PROP	
	branch1:	$X_1 = X_2 = X$	$dX = dX_1+dX_2$	
#	op. 1	: μ = np.mean(X_1, axis=0)	$dX_1 = d\mu*1_{(N,D)}/N$	
&	op. 2	: $\alpha_1 = X_2-\mu$	$d\mu = 1_{(1,N)}.(-d\alpha_1)$	$dX_2 = d\alpha_1$
	branch2:	$\alpha_2 = \alpha_3 = \alpha_1$	$d\alpha_1 = d\alpha_2 + d\alpha_3$	
	op. 3	: $c = \alpha_2**2$	$d\alpha_2 = dc*2*\alpha_2$	
#	op. 4	: v = np.mean(c, axis=0)	$dc = dv*(1_{(N,D)}/N)$	
	op. 5	: s = np.sqrt(v+ε)	$dv = ds*0.5*(v+\varepsilon)**(-0.5)$	
	op. 6	: i = 1/s	$ds = di*(-1/s**2)$	
&	op. 7	: $h = \alpha_3*i$	$di = 1_{(1,N)}.dh*\alpha_3$	$\alpha_3 = dh*i$
&	op. 8	: $k = h*\gamma$	$dh = dk*\gamma$	$d\gamma = 1_{(1,N)}.dk*h$
&	op. 9	: $y = k+\beta$	$dk = dy$	$d\beta = 1_{(1,N)}.dy$

- # ... dimensional reduction
- & ... broadcasting
- green ... linear operations
- orange ... non-linear operations
- pink ... inputs needing gradients
- brown ... outputs

convolutional layer overview



input: $(2P*W_1)x(2P*H_1)xD_1$

hyperparameters:

number of filters	K
filter spatial extent	F
stride	S
# of zero padding	P

output: $W_2 x H_2 x D_2$

$W_2 = (W_1 - F + 2P)/S + 1$

$H_2 = (H_1 - F + 2P)/S + 1$

$D_2 = K$

OUTPUT CALCULATION:

```
        [[ 2  0  0     [[ -1  1  1      [[ 0  2  0     [[-1  1  0      [[ 1  0  0     [[ 1  0  0
np.sum(    2  1  0  *      0  0  0   +     2  2  0  *     0  0  0   +     1  1  0  *    -1  1  0  ) + 0 = -1
           2  2  0]]       1 -1  0]]       0  1  0]]     -1  0 -1]]       2  0  0]]     -1  1 -1]]
```

figure adapted from … http://cs231n.github.io/convolutional-networks/