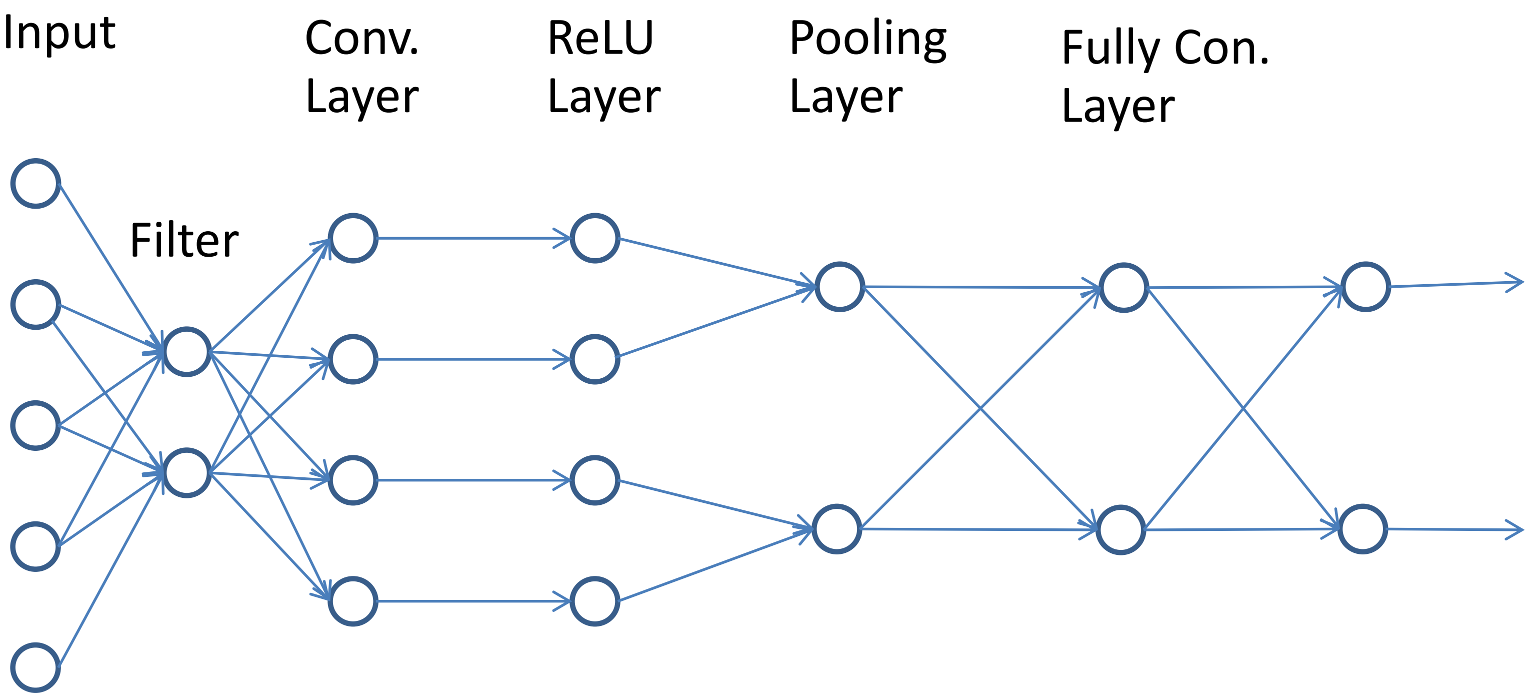
A Simple Example of Backpropagation in CNN

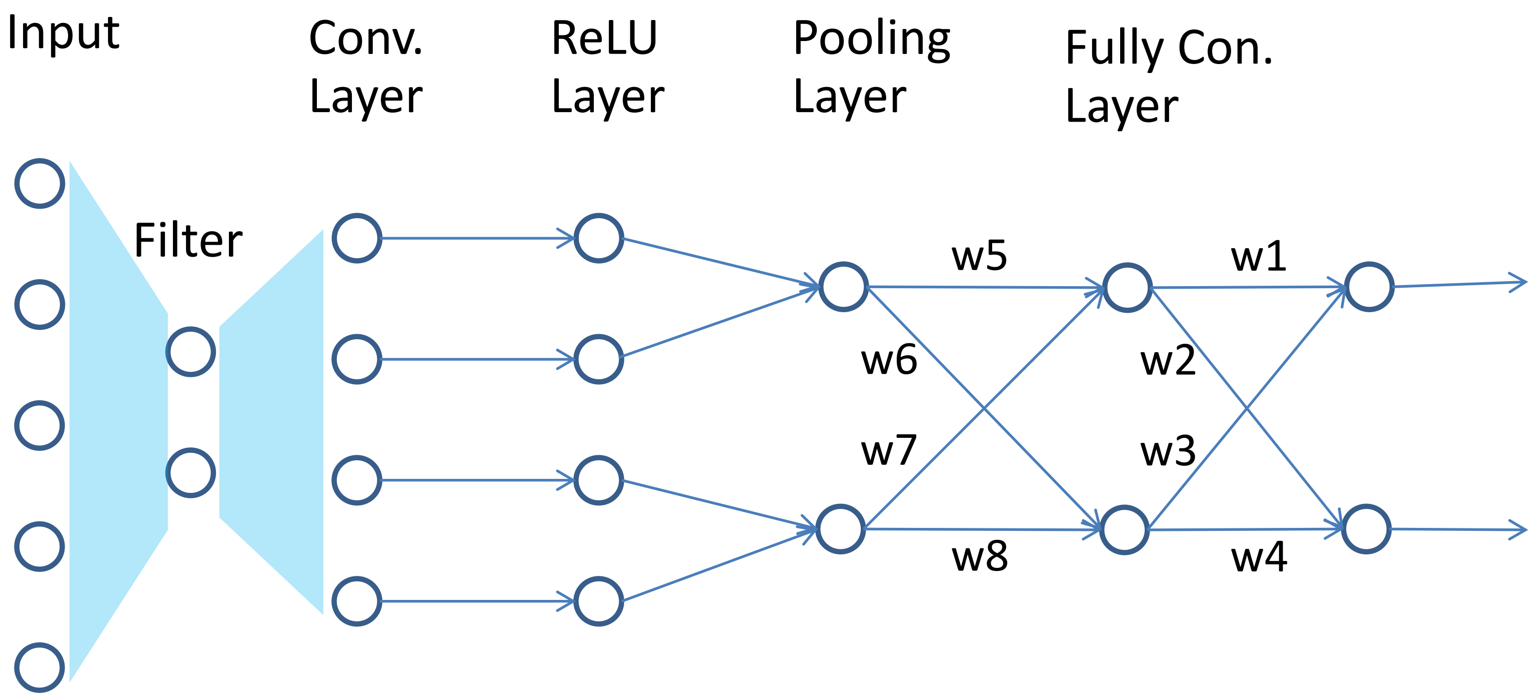
Kietikul Jearanaitanakij

Department of Computer Engineering, KMITL

Backpropagation in CNN



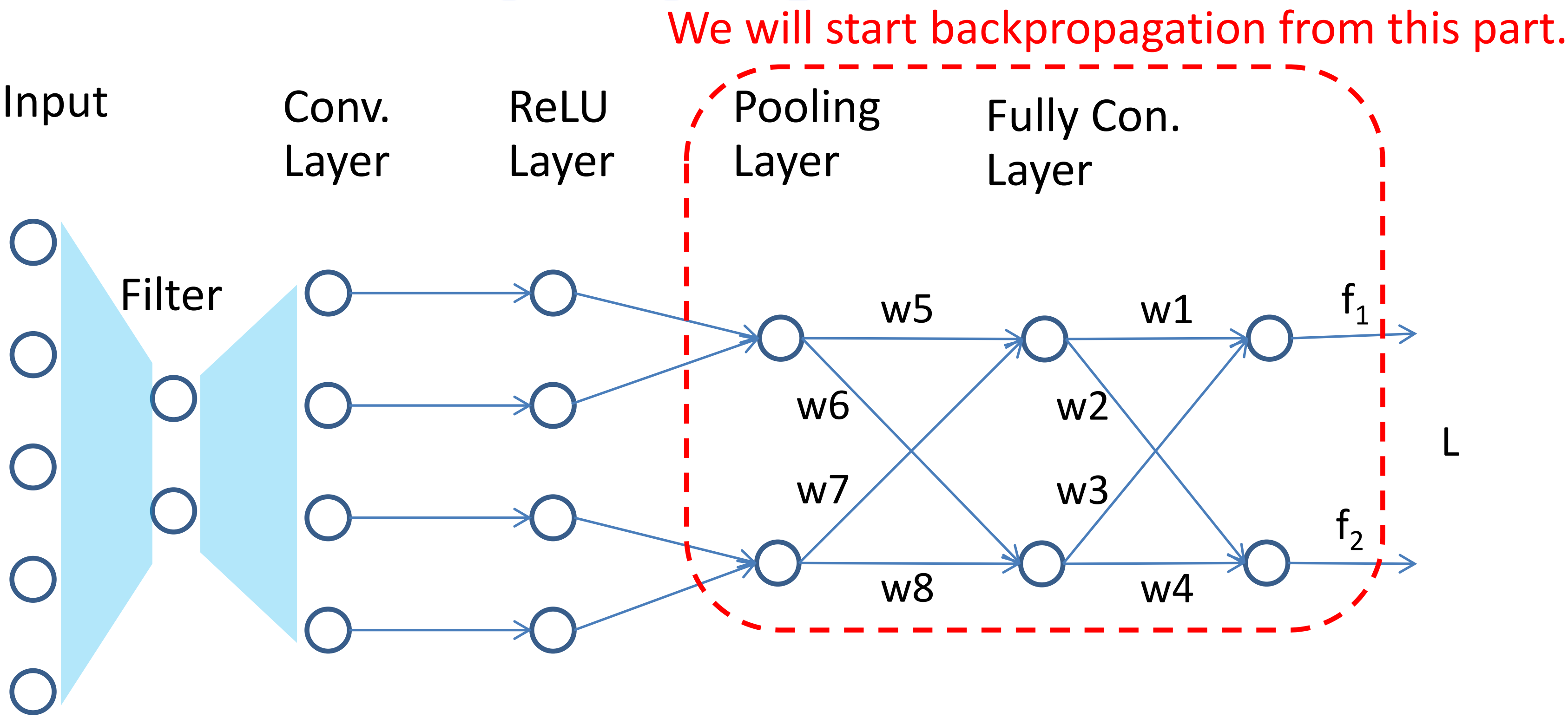
Backpropagation in CNN



Recall from lecture 4
Loss function = Data Loss + Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\textbf{Data loss}\text{: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\textbf{Regularization}\text{: Model should be "simple", so it works on test data}}$$

Backpropagation in CNN

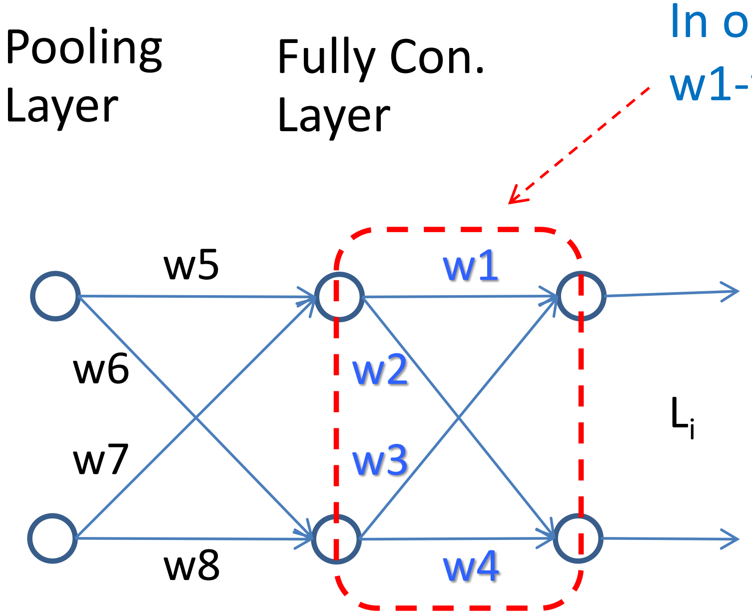


Loss Function (L) = $L_i + \frac{1}{2}\lambda \sum_k \sum_l w_{k,l}{}^2$; λ is regularization strength

$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

; where $\mathbf{L_i}$ is the data loss of the training pattern i. Here we use a cross-entropy function (softmax).

Backpropagation in CNN



In order to do backpropagation for updating weights w1-w4, we need to find the gradient of L wrt w.

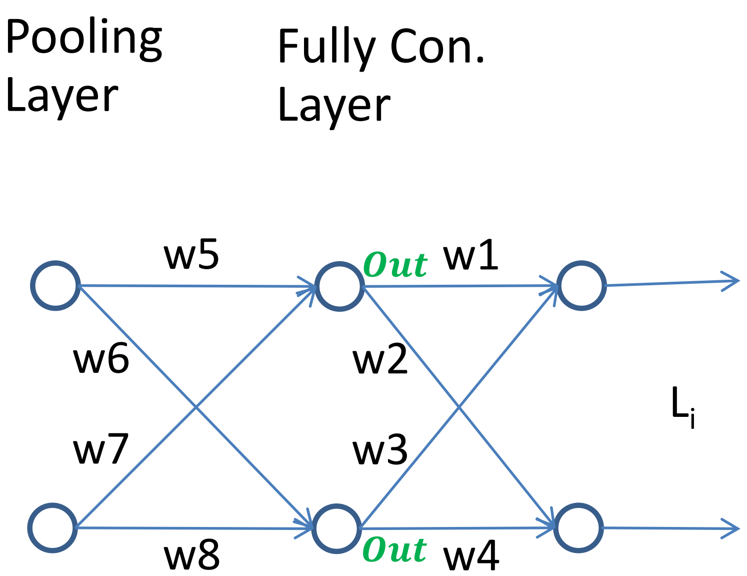
$$\text{Loss Function (L)} = \underbrace{L_i}_{\text{Data loss}} + \underbrace{\frac{1}{2}\lambda \sum_k \sum_l w_{k,l}^2}_{\text{Regularization loss}}$$

$$\frac{\partial \mathrm{L}}{\partial w} = \frac{\partial L_i}{\partial w} + \frac{\partial \frac{1}{2}\lambda \sum_k \sum_l w_{k,l}^2}{\partial w}$$

$$\frac{\partial \mathrm{L}}{\partial w} = \frac{\partial L_i}{\partial w} + \lambda w$$

We need a chain rule to find $\frac{\partial L_i}{\partial w}$.

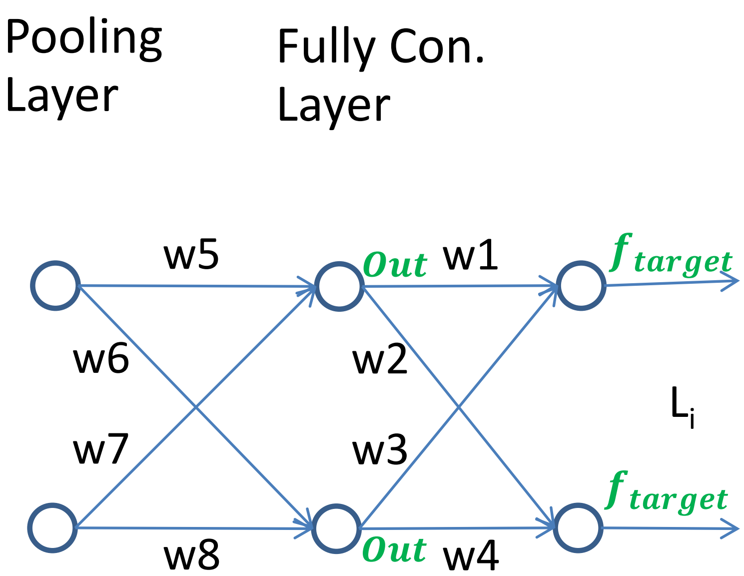
Backpropagation in CNN



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$; Chain rule

Backpropagation in CNN



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

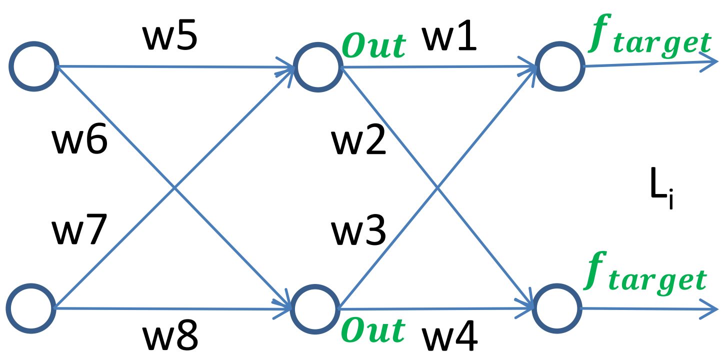
$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

Backpropagation in CNN

Pooling Layer

Fully Con. Layer



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

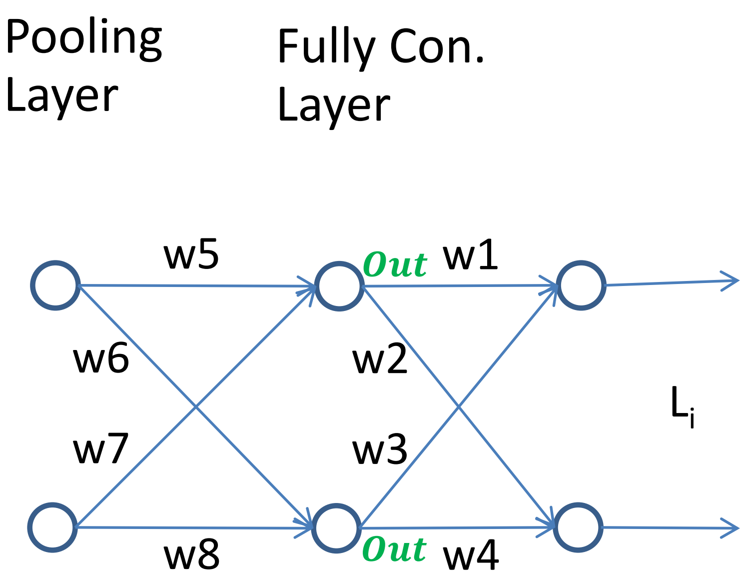
$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

$$\frac{\partial L_i}{\partial f_{target}} = \frac{\partial(-\log p)}{\partial p} \cdot \frac{\partial p}{\partial f_{target}}$$

$$Let\ p = \frac{e^{f_{target}}}{\sum_j e^{f_j}}$$

Backpropagation in CNN



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

$$\frac{\partial L_i}{\partial f_{target}} = \frac{\partial(-\log p)}{\partial p} \cdot \frac{\partial p}{\partial f_{target}}$$

$$Let\ p = \frac{e^{f_{target}}}{\sum_j e^{f_j}}$$

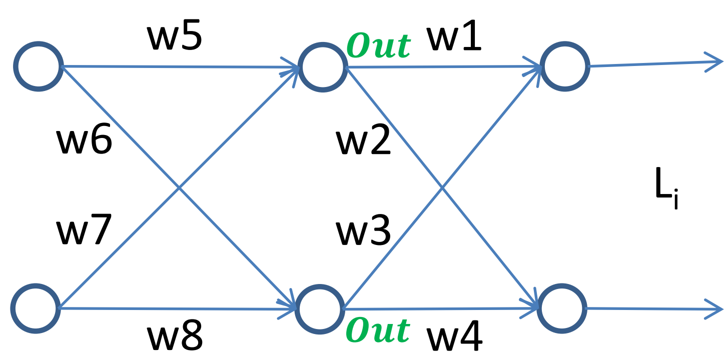
$$= -\frac{1}{p} \cdot \frac{\partial p}{\partial f_{target}} \qquad = -\frac{\sum_j e^{f_j}}{e^{f_{target}}} \cdot \frac{\partial \frac{e^{f_{target}}}{\sum_j e^{f_j}}}{\partial f_{target}}$$

Formula: $\boxed{\frac{d}{dp}(\log(p)) = \frac{1}{p}}$

Backpropagation in CNN

Pooling Layer

Fully Con. Layer



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

$$\frac{\partial L_i}{\partial f_{target}} = \frac{\partial(-\log p)}{\partial p} \cdot \frac{\partial p}{\partial f_{target}}$$

$$Let\ p = \frac{e^{f_{target}}}{\sum_j e^{f_j}}$$

$$= -\frac{1}{p} \cdot \frac{\partial p}{\partial f_{target}} \qquad = -\frac{\sum_j e^{f_j}}{e^{f_{target}}} \cdot \frac{\partial \dfrac{\overbrace{e^{f_{target}}}^{u}}{\underbrace{\sum_j e^{f_j}}_{v}}}{\partial f_{target}}$$

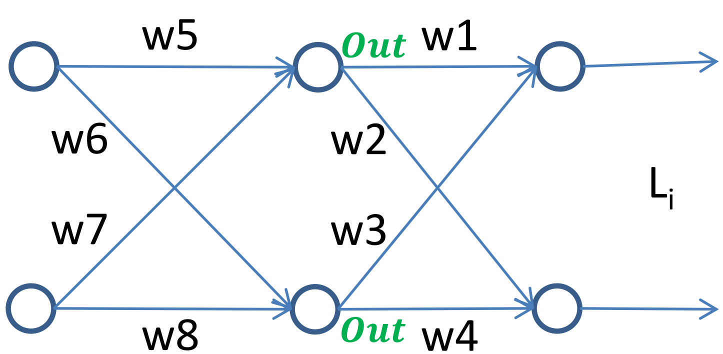
$$\frac{\sum_j e^{f_j} \cdot \dfrac{\partial e^{f_{target}}}{\partial f_{target}} - e^{f_{target}} \dfrac{\partial \sum_j e^{f_j}}{\partial f_{target}}}{\left(\sum_j e^{f_j}\right)^2}$$

$$\boxed{\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{v\dfrac{du}{dx} - u\dfrac{dv}{dx}}{v^2}}$$

Backpropagation in CNN

Pooling Layer

Fully Con. Layer



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

$$\frac{\partial L_i}{\partial f_{target}} = \frac{\partial(-\log p)}{\partial p} \cdot \frac{\partial p}{\partial f_{target}}$$

$$Let\ p = \frac{e^{f_{target}}}{\sum_j e^{f_j}}$$

$$= -\frac{1}{p} \cdot \frac{\partial p}{\partial f_{target}} \qquad = -\frac{\sum_j e^{f_j}}{e^{f_{target}}} \cdot \frac{\partial \frac{e^{f_{target}}}{\sum_j e^{f_j}}}{\partial f_{target}}$$

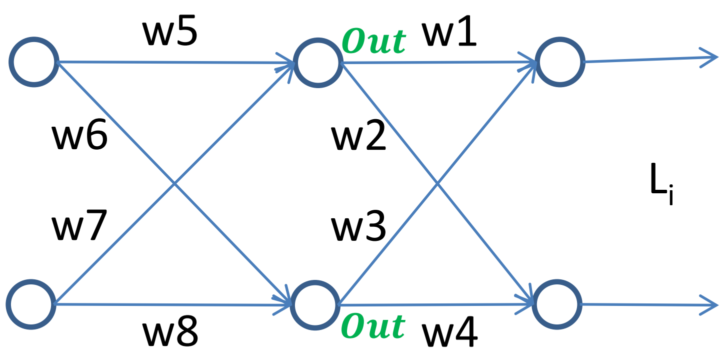
$$\frac{\sum_j e^{f_j} \cdot \frac{\partial e^{f_{target}}}{\partial f_{target}} - e^{f_{target}} \frac{\partial \sum_j e^{f_j}}{\partial f_{target}}}{\left(\sum_j e^{f_j}\right)^2}$$

$$\frac{\sum_j e^{f_j} \cdot e^{f_{target}} - e^{f_{target}} \cdot e^{f_{target}}}{\left(\sum_j e^{f_j}\right)^2}$$

Backpropagation in CNN

Pooling Layer

Fully Con. Layer



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

$$\frac{\partial L_i}{\partial f_{target}} = \frac{\partial(-\log p)}{\partial p} \cdot \frac{\partial p}{\partial f_{target}}$$

$$Let\ p = \frac{e^{f_{target}}}{\sum_j e^{f_j}}$$

$$= -\frac{1}{p} \cdot \frac{\partial p}{\partial f_{target}}$$

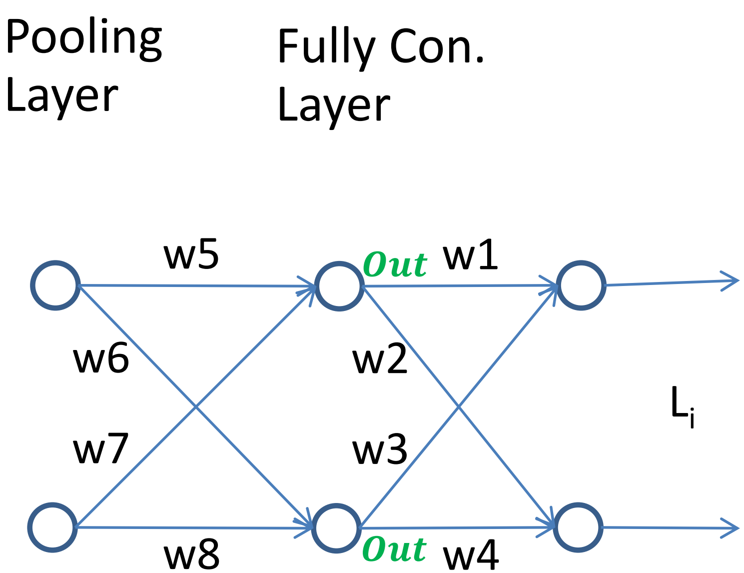
$$= -\frac{\sum_j e^{f_j}}{e^{f_{target}}} \cdot \frac{\partial \frac{e^{f_{target}}}{\sum_j e^{f_j}}}{\partial f_{target}}$$

$$\frac{\sum_j e^{f_j} \cdot \frac{\partial e^{f_{target}}}{\partial f_{target}} - e^{f_{target}} \frac{\partial \sum_j e^{f_j}}{\partial f_{target}}}{\left(\sum_j e^{f_j}\right)^2}$$

$$\frac{\sum_j e^{f_j} . e^{f_{target}} - e^{f_{target}} . e^{f_{target}}}{\left(\sum_j e^{f_j}\right)^2}$$

$$= -\frac{\sum_j e^{f_j}}{e^{f_{target}}} \cdot \frac{e^{f_{target}} . \left[\left(\sum_j e^{f_j}\right) - e^{f_{target}}\right]}{\left(\sum_j e^{f_j}\right)^2}$$

Backpropagation in CNN



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{target}} \cdot \frac{\partial f_{target}}{\partial w}$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

$$\frac{\partial L_i}{\partial f_{target}} = \frac{\partial(-\log p)}{\partial p} \cdot \frac{\partial p}{\partial f_{target}} \qquad Let\ p = \frac{e^{f_{target}}}{\sum_j e^{f_j}}$$

$$= -\frac{1}{p} \cdot \frac{\partial p}{\partial f_{target}} \qquad = -\frac{\sum_j e^{f_j}}{e^{f_{target}}} \cdot \frac{\partial \frac{e^{f_{target}}}{\sum_j e^{f_j}}}{\partial f_{target}}$$

$$= -\frac{\sum_j e^{f_j}}{e^{f_{target}}} \cdot \frac{e^{f_{target}} . \left[\left(\sum_j e^{f_j}\right) - e^{f_{target}}\right]}{\left(\sum_j e^{f_j}\right)^2}$$

$$= -\frac{\left(\sum_j e^{f_j} - e^{f_{target}}\right)}{\sum_j e^{f_j}} = -(1 - p_{target})$$

$$= (p_{target} - 1)$$

Note that we only minus 1 from p_{target}. Other p's remain unchanged. (see explanation on the next page)

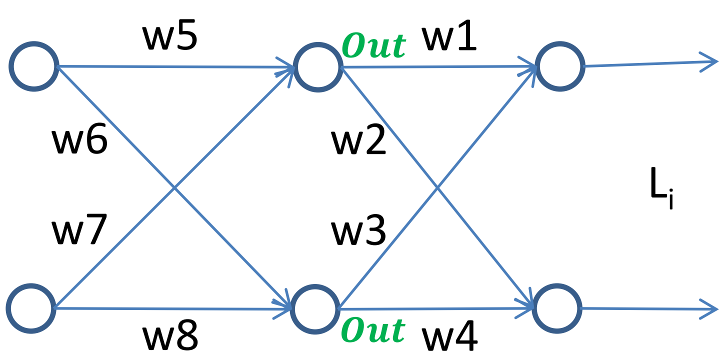
Backpropagation in CNN

For $\dfrac{\partial L_i}{\partial f_{non\_target}}$

Li is defined only on the target.
Therefore, Li formula is unchanged.

Pooling Layer

Fully Con. Layer



$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial \sum w.Out}{\partial w} = Out$$

$$\frac{\partial L_i}{\partial w} = \frac{\partial L_i}{\partial f_{non\_target}} . \frac{\partial f_{non\_target}}{\partial w}$$

$$Let\ p = \frac{e^{f_{target}}}{\sum_j e^{f_j}}$$

$$\frac{\partial L_i}{\partial f_{non\_target}} = \frac{\partial(-\log p)}{\partial p} . \frac{\partial p}{\partial f_{non\_target}}$$

$$= -\frac{1}{p} . \frac{\partial p}{\partial f_{non\_target}} = -\frac{\sum_j e^{f_j}}{e^{f_{target}}} . \frac{\partial \frac{e^{f_{target}}}{\sum_j e^{f_j}}}{\partial f_{non\_target}}$$

$$\frac{\sum_j e^{f_j} . \frac{\partial e^{f_{target}}}{\partial f_{non\_target}} - e^{f_{target}} \frac{\partial \sum_j e^{f_j}}{\partial f_{non\_target}}}{\left(\sum_j e^{f_j}\right)^2}$$

$$= -\frac{\sum_j e^{f_j}}{e^{f_{target}}} . \left(-\frac{e^{f_{target}} . e^{f_{non\_target}}}{\left(\sum_j e^{f_j}\right)^2}\right)$$

$$\frac{\sum_j e^{f_j} . 0 - e^{f_{target}} . e^{f_{non\_target}}}{\left(\sum_j e^{f_j}\right)^2}$$

$$= \frac{e^{f_{non\_target}}}{\sum_j e^{f_j}} = p_{non\_target}$$

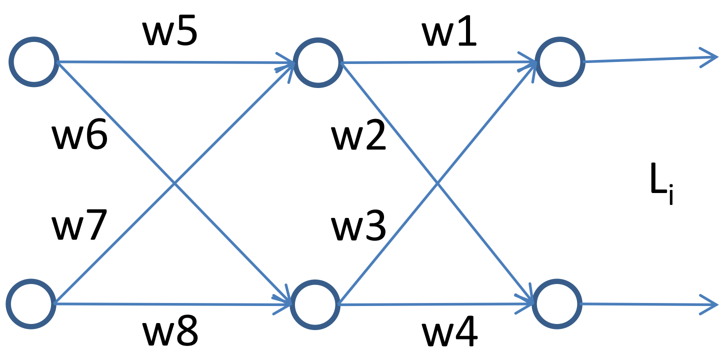
Backpropagation in CNN

Example: To calculate $\frac{\partial L_i}{\partial f_{target}}$, suppose our target is p2

$$\frac{\partial L_i}{\partial f_{target}} = (p_{target} - 1)$$

Pooling Layer

Fully Con. Layer



p1=0.3

p2=0.7
target

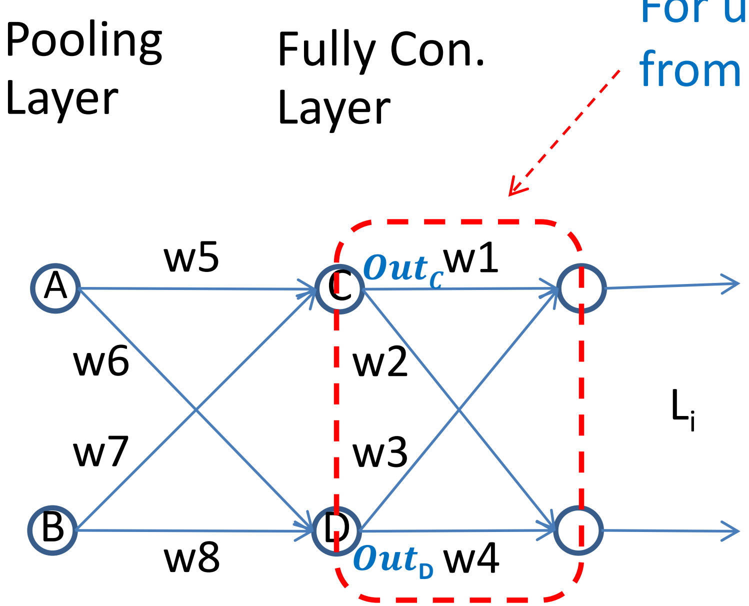
$\frac{\partial L_i}{\partial fp1}$ =0.3
(unchanged)

$\frac{\partial L_i}{\partial fp2}$ = 0.7-1
= -0.3

Backpropagation in CNN

Weights updating (w1-w4):

For updating weights w1-w4, we substitute the results from the chain rule.



$$w = w - \propto \frac{\partial L}{\partial w}$$

$$w = w - \propto \frac{\partial\left(\frac{1}{N}\sum_i^N L_i + \frac{1}{2}\lambda \sum_k \sum_l w_{k,l}{}^2\right)}{\partial w}$$

$$w = w - \propto \left(\frac{\partial L_i}{\partial f} . \frac{\partial f}{\partial w} + \lambda w\right)$$

$(p_{target} - 1)$ Out

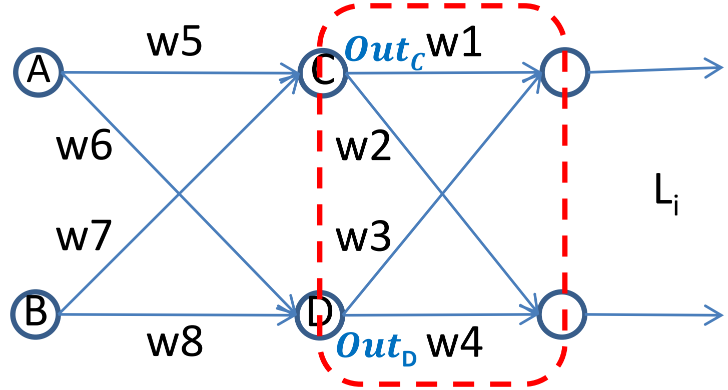
Backpropagation in CNN

Weights updating (w1-w4):

Example of weight updating (w1-w4)

Pooling Layer

Fully Con. Layer



$\frac{\partial L_i}{\partial f}$

$\frac{\partial L_i}{\partial fp1}$ =0.3

$\frac{\partial L_i}{\partial fp2}$ = 0.7-1 = - 0.3

$$\boxed{w = w - \propto (\frac{\partial L}{\partial f} . \frac{\partial f}{\partial w} + \lambda w)}$$

$(p_{target} - 1)$ Out

$$w_1 = w_1 - \propto (0.3 * OutC + \lambda w_1)$$

$$w_2 = w_2 - \propto (-0.3 * OutC + \lambda w_2)$$

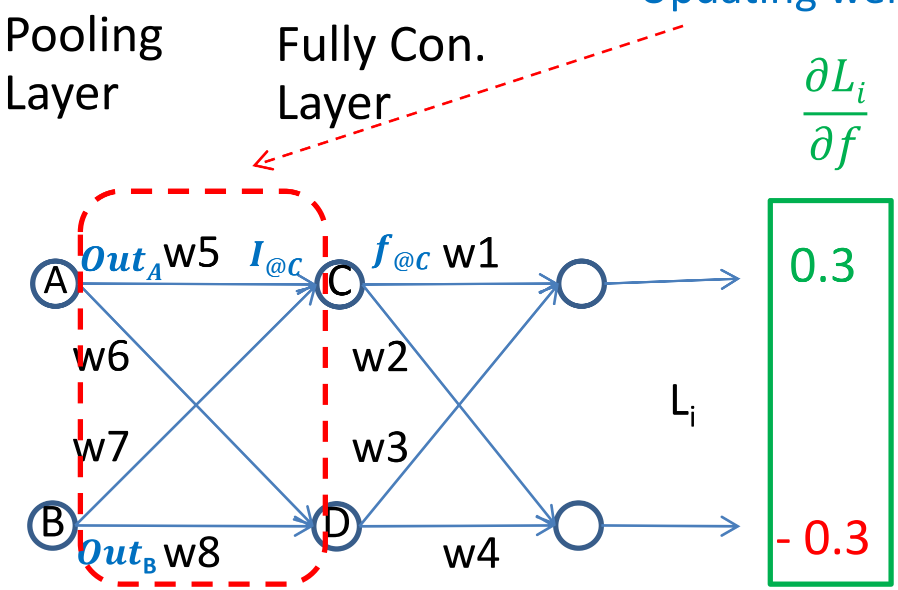
$$w_3 = w_3 - \propto (0.3 * OutD + \lambda w_3)$$

$$w_4 = w_4 - \propto (-0.3 * OutD + \lambda w_4)$$

Backpropagation in CNN

Weights updating (w5-w8):

Updating weights in the inner layer (w5-w8)



$$w_5 = w_5 - \propto (\frac{\partial L}{\partial w_5})$$

$$w_5 = w_5 - \propto (\frac{\partial Li}{\partial w_5} + \lambda w_5)$$

Backpropagation in CNN

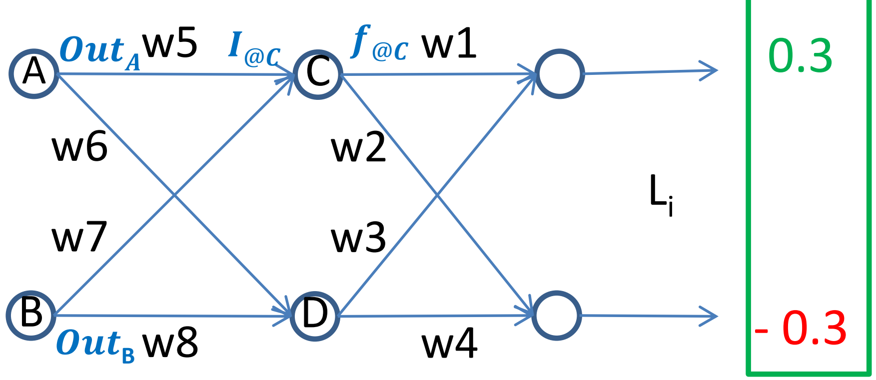
Weights updating (w5-w8):

Updating weights in the inner layer (w5-w8)

Pooling Layer

Fully Con. Layer

$$\frac{\partial L_i}{\partial f}$$



$$w_5 = w_5 - \propto (\frac{\partial L}{\partial w_5})$$

$$w_5 = w_5 - \propto (\frac{\partial Li}{\partial w_5} + \lambda w_5)$$

$$\frac{\partial Li}{\partial w_5} = \frac{\partial Li}{\partial f_{@c}} . \frac{\partial f_{@c}}{\partial I_{@c}} . \frac{\partial I_{@c}}{\partial w_5}$$

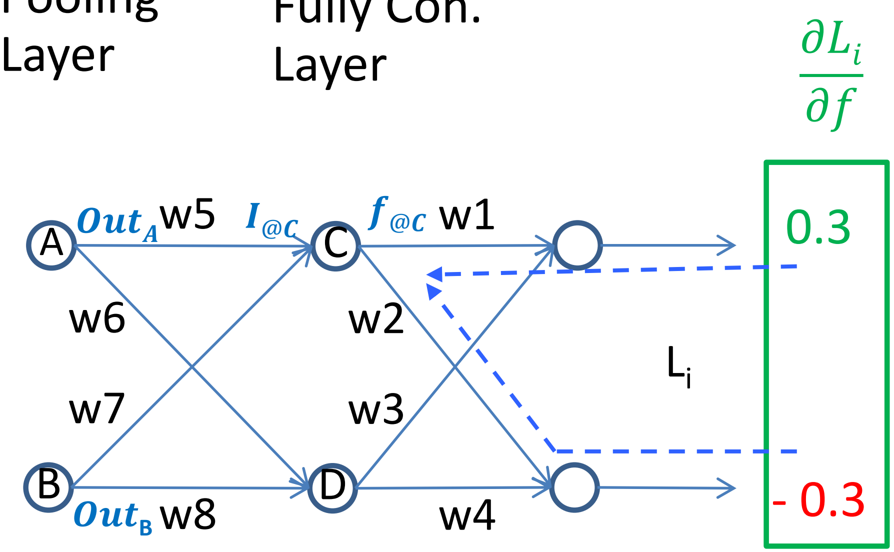
Chain rule 3 times!

Backpropagation in CNN

Updating weights in the inner layer (w5-w8)

Pooling Layer

Fully Con. Layer



$$w_5 = w_5 - \propto (\frac{\partial L}{\partial w_5})$$

$$w_5 = w_5 - \propto (\frac{\partial Li}{\partial w_5} + \lambda w_5)$$

$$\frac{\partial Li}{\partial w_5} = \frac{\partial Li}{\partial f_{@c}} . \frac{\partial f_{@c}}{\partial I_{@c}} . \frac{\partial I_{@c}}{\partial w_5}$$

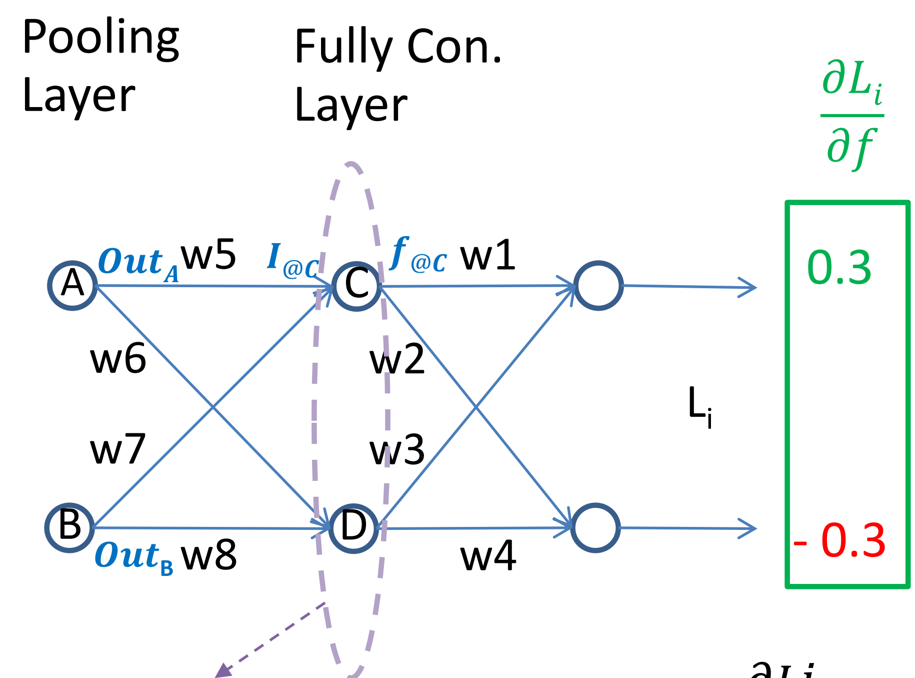
$$\frac{\partial Li}{\partial w_5} = \underbrace{\sum(\frac{\partial L_i}{\partial f} . w_{@c})}_{\text{Collect } \frac{\partial L_i}{\partial f} \text{ from the output layer that backpropagate to node c.}} . f@c.* (1 - f@c). Out_A$$

Collect $\frac{\partial L_i}{\partial f}$ from the output layer

that backpropagate to node c.

Backpropagation in CNN

Updating weights in the inner layer (w5-w8)



Not Softmax function.
(Sigmoid, tanh, ReLu)

$$w_5 = w_5 - \propto (\frac{\partial L}{\partial w_5})$$

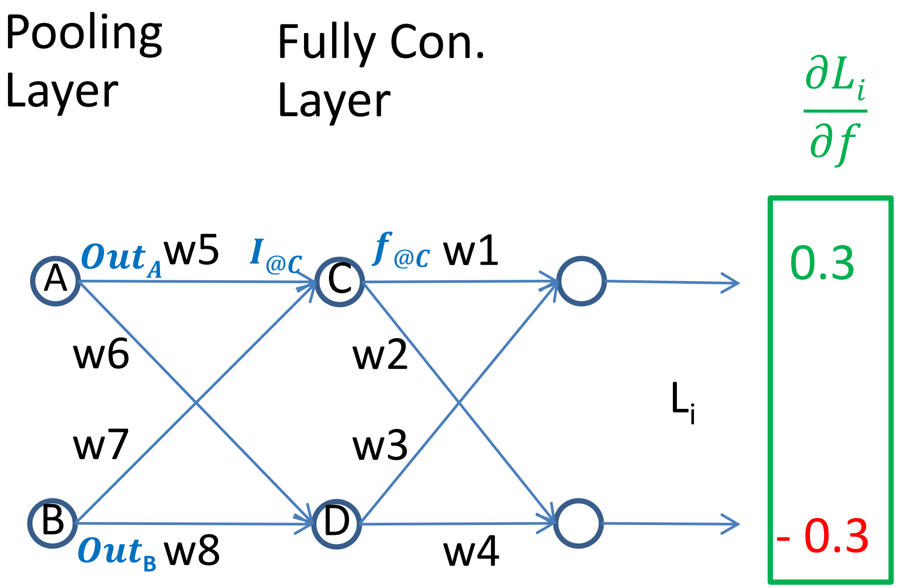
$$w_5 = w_5 - \propto (\frac{\partial Li}{\partial w_5} + \lambda w_5)$$

$$\frac{\partial Li}{\partial w_5} = \frac{\partial Li}{\partial f_{@c}} . \frac{\partial f_{@c}}{\partial I_{@c}} . \frac{\partial I_{@c}}{\partial w_5}$$

$$\frac{\partial Li}{\partial w_5} = \sum(\frac{\partial L_i}{\partial f} . w_{@c}) . \underbrace{f@c .* (1 - f@c)}_{\text{Derivative of sigmoid function}} . Out_A$$

Backpropagation in CNN

Updating weights in the inner layer (w5-w8)



$$w_5 = w_5 - \propto (\frac{\partial L}{\partial w_5})$$

$$w_5 = w_5 - \propto (\frac{\partial Li}{\partial w_5} + \lambda w_5)$$

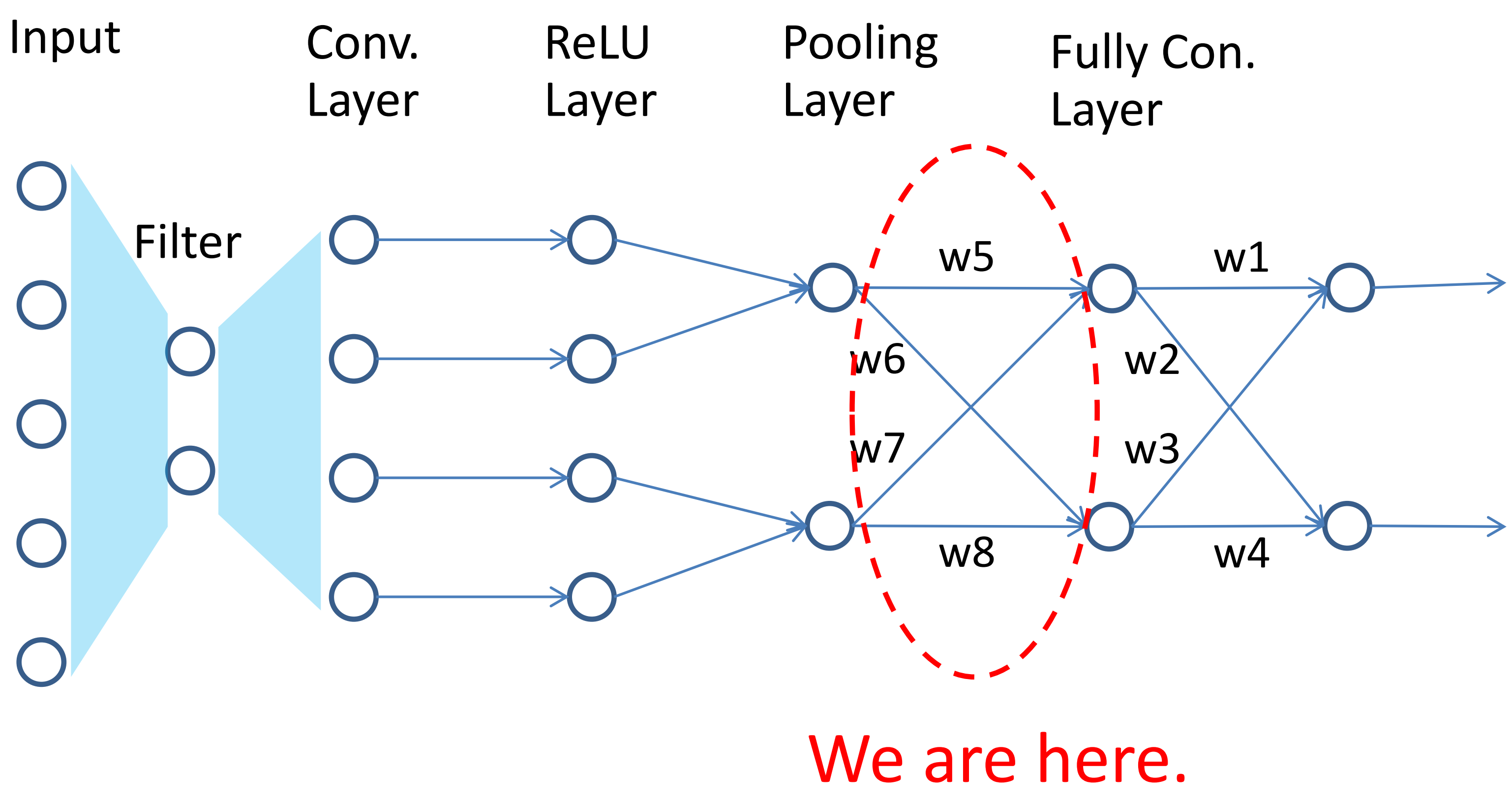
$$\frac{\partial Li}{\partial w_5} = \frac{\partial Li}{\partial f_{@c}} . \frac{\partial f_{@c}}{\partial I_{@c}} . \frac{\partial I_{@c}}{\partial w_5}$$

$$\frac{\partial Li}{\partial w_5} = \sum(\frac{\partial L_i}{\partial f} . w_{@c}). f@c.* (1 - f@c). Out_A$$

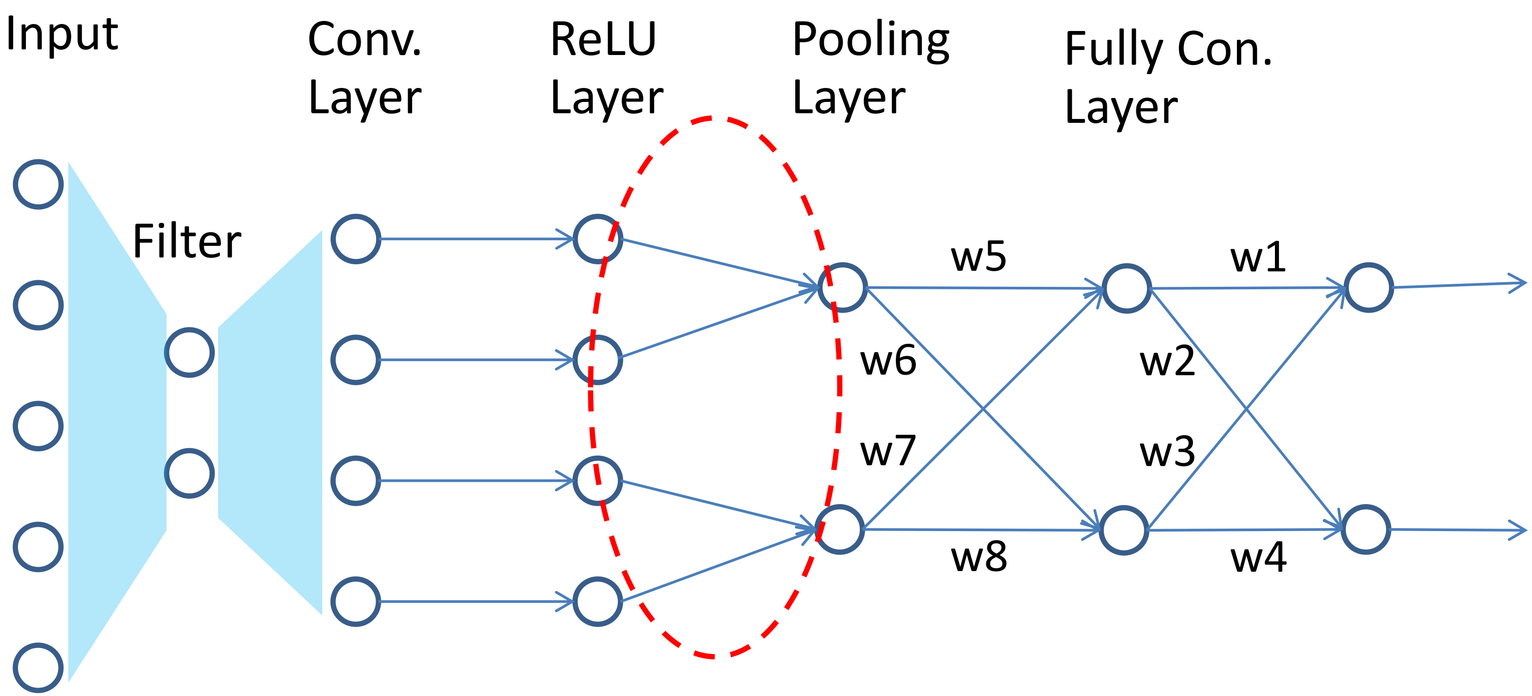
$$\frac{\partial Li}{\partial w_5} = ((0.3 * w1) + (-0.3 * w2)). f@c.* (1 - f@c). Out_A$$

Then, updating w6-w8 in the same manner as w5.

Backpropagation in CNN



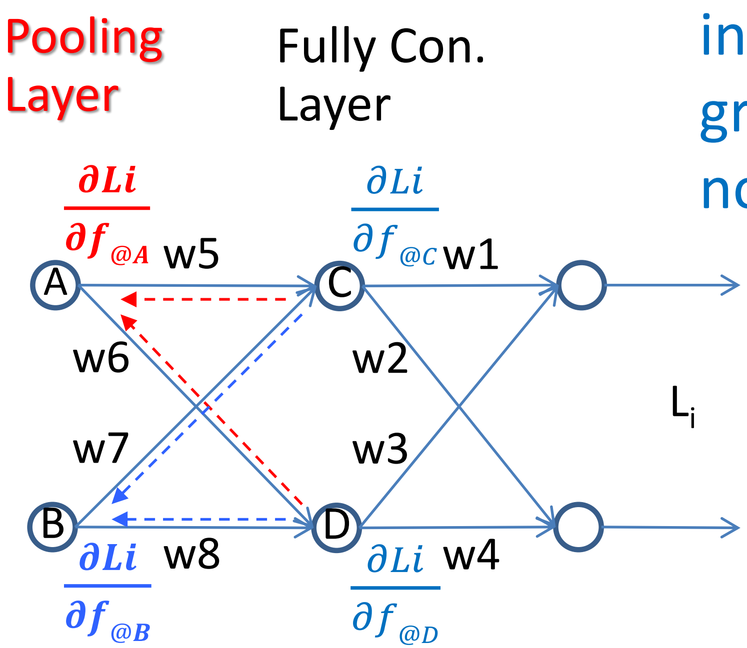
Backpropagation in CNN



Next, backpropagate gradient through pooling layer.

Backpropagation in CNN

Prepare gradients in pooling layer:

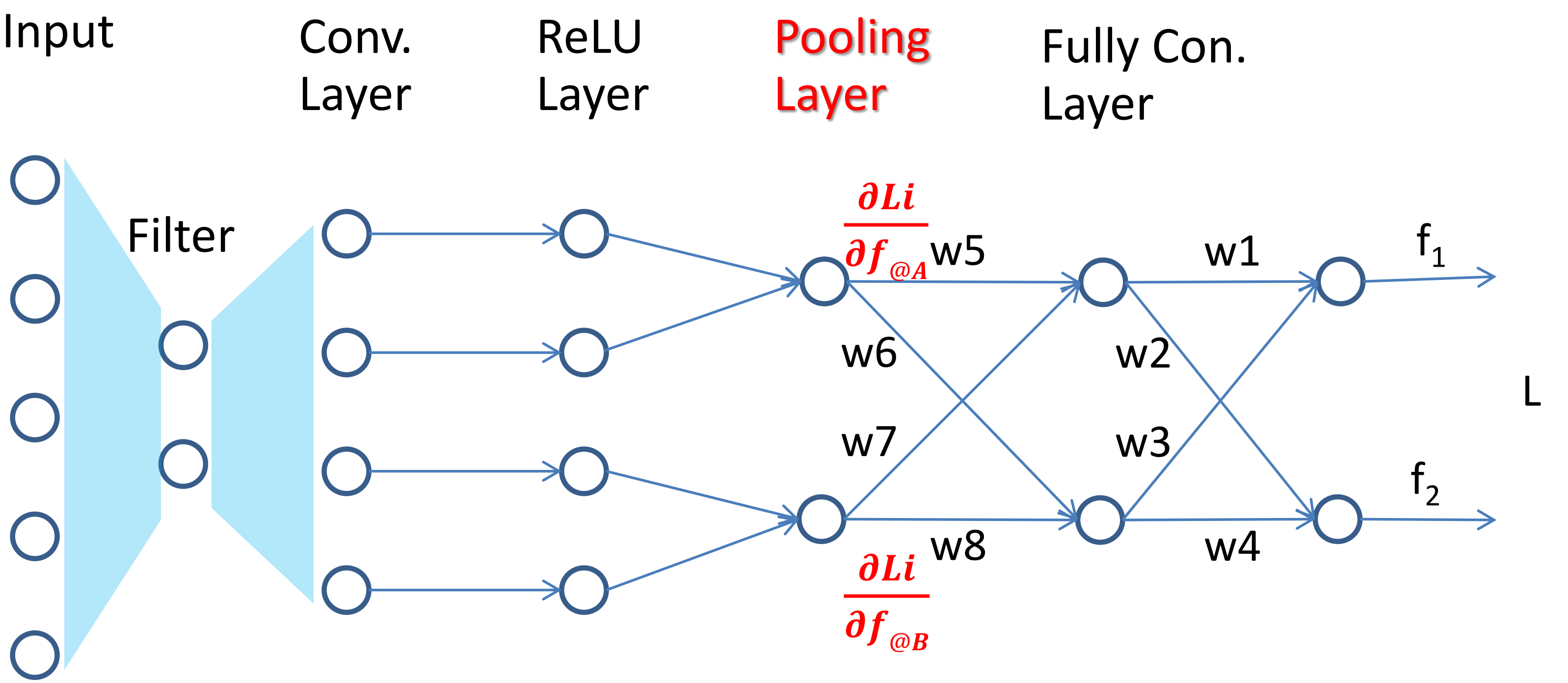


In order to update weights of the filter in Conv. Layer, we need to flow the gradients of *Li* calculated at node A and node B back through the network.

$$\frac{\partial Li}{\partial f_{@A}} = \left(\left(\frac{\partial Li}{\partial f_{@C}} * w5\right) + \left(\frac{\partial Li}{\partial f_{@D}} * w6\right)\right)$$

$$\frac{\partial Li}{\partial f_{@B}} = \left(\left(\frac{\partial Li}{\partial f_{@C}} * w7\right) + \left(\frac{\partial Li}{\partial f_{@D}} * w8\right)\right)$$

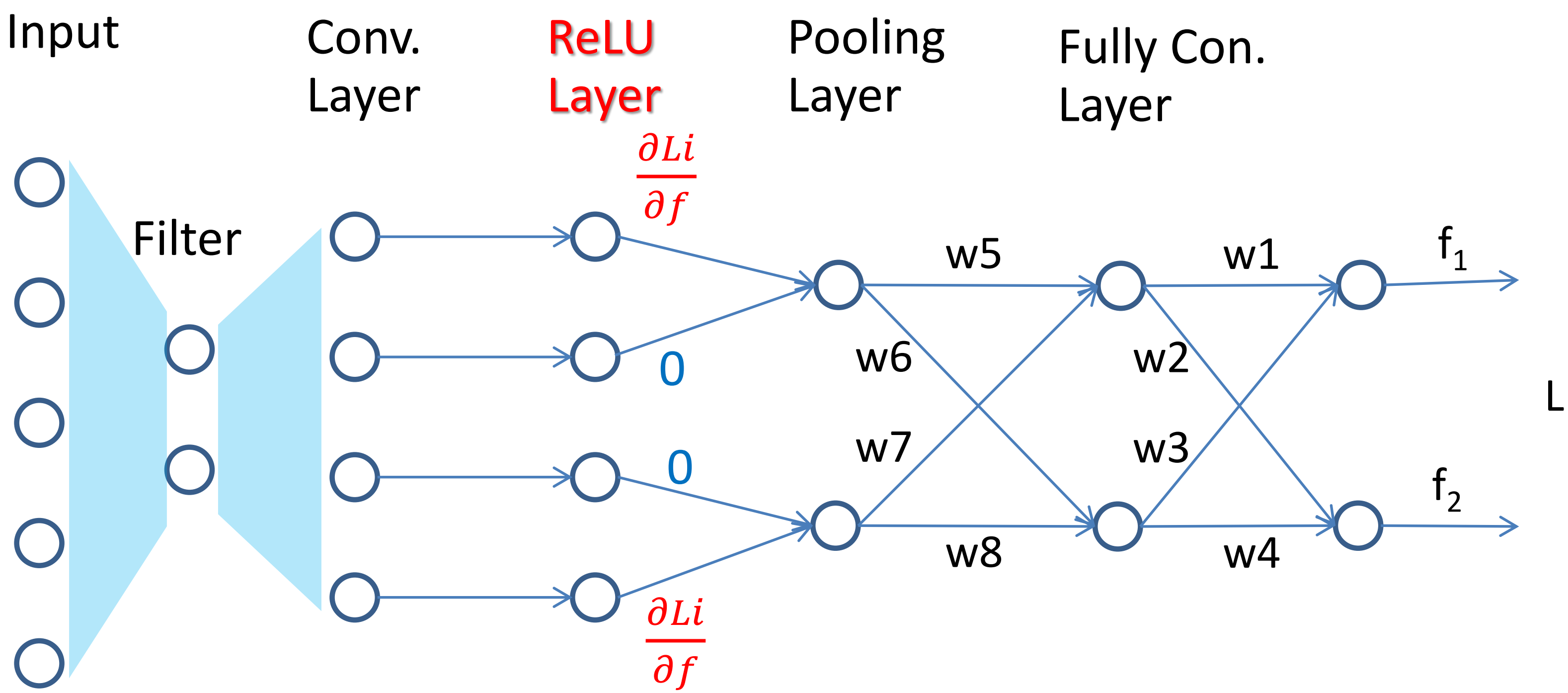
Backpropagation in CNN



- Each neuron in pooling layer just pools the maximum value among its corresponding neurons in previous layer (ReLU).
- Therefore, the gradient of the neuron in pooling layer will flow through the neuron which has the largest activation value in previous layer. Other neurons will have zero gradients.

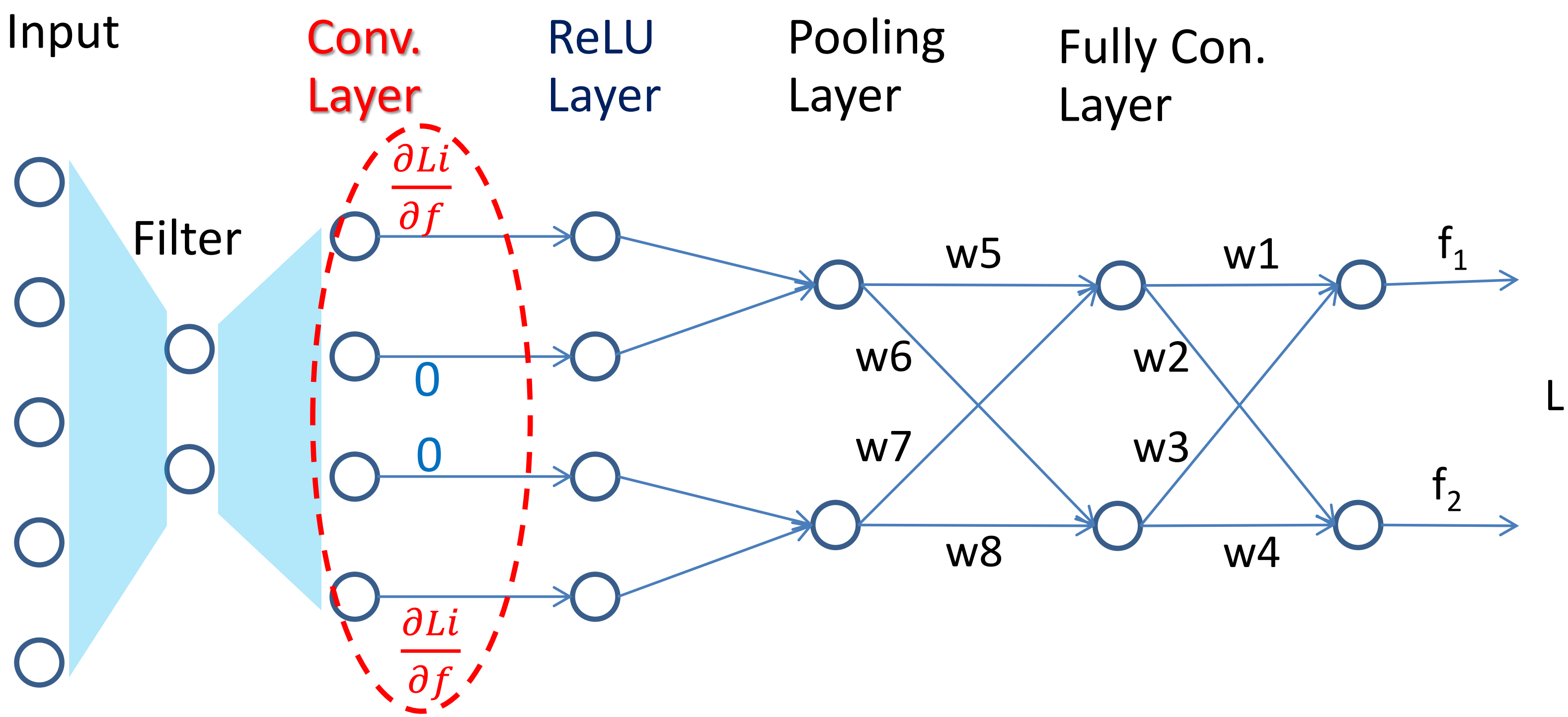
Backpropagation in CNN

Recall: ReLU(x) = max(0,x)



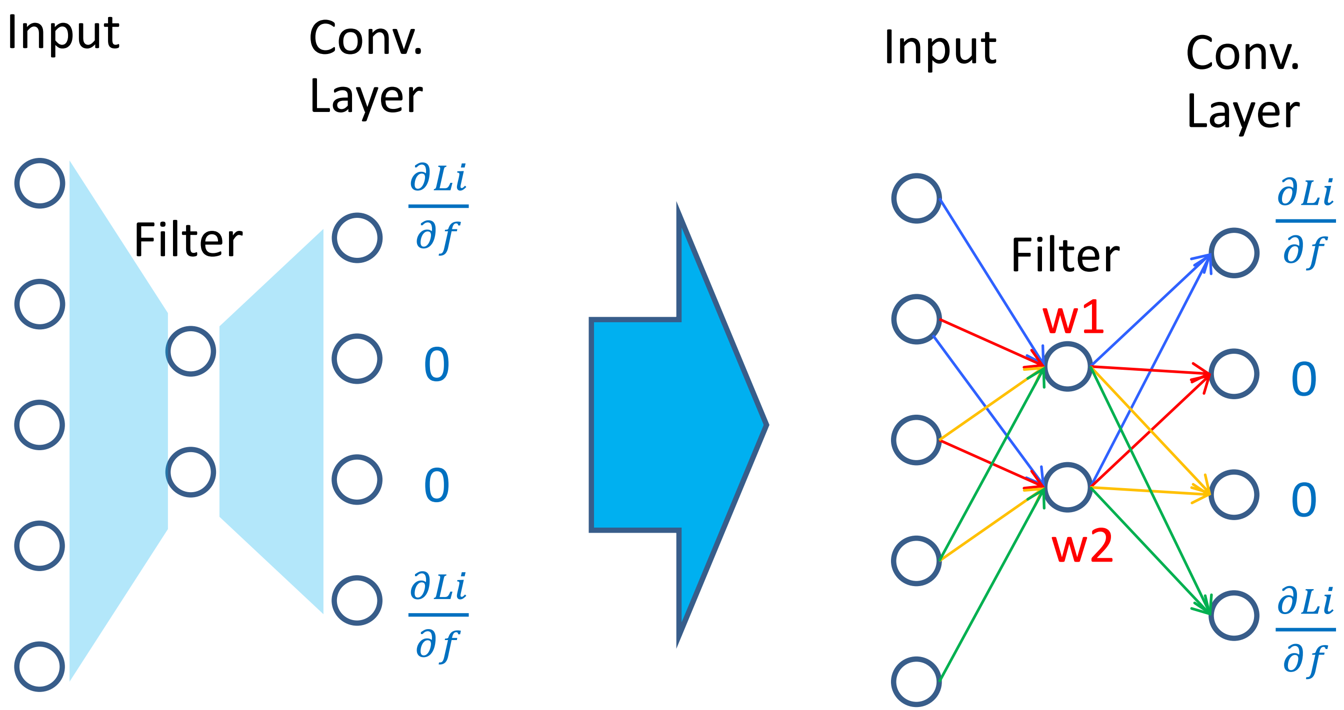
- Each neuron in ReLU layer filter the negative value from its corresponding neuron in previous layer (Conv.).
- Therefore, the gradient of the neuron in ReLU layer will flow through the neuron X in previous layer if the activation value of neuron X is positive or zero. Otherwise, neuron X with negative activation value will have zero gradients.

Backpropagation in CNN



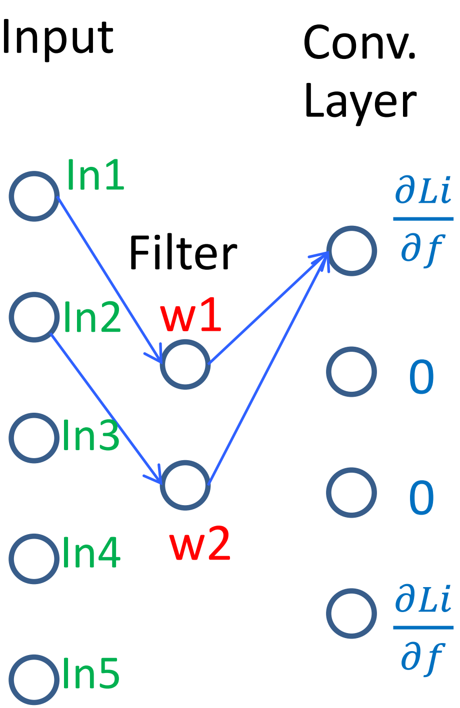
We are here.

Backpropagation in CNN



Recall that same Conv. layer shares the same set filter weights.

Backpropagation in CNN



For the first neuron of Conv. Layer.

$$\frac{\partial Li}{\partial w1} = \overset{\text{known}}{\frac{\partial Li}{\partial f}} . \frac{\partial f}{\partial w1}$$

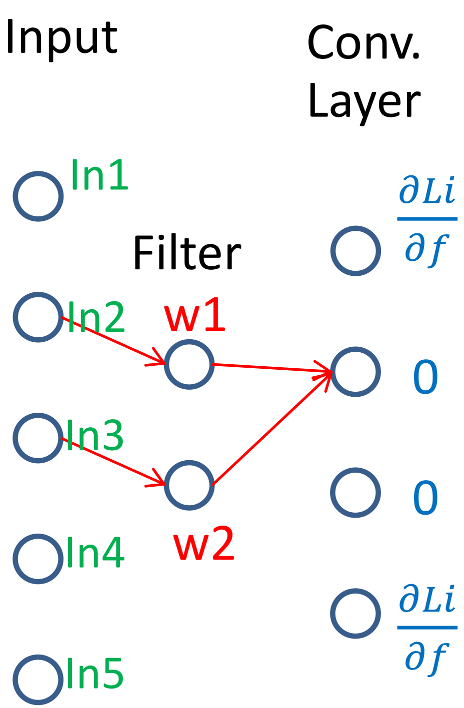
$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} . \frac{\partial (In1.w1 + In2.w2)}{\partial w1}$$

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} . In1$$

Similarly,

$$\frac{\partial Li}{\partial w2} = \frac{\partial Li}{\partial f} . In2$$

Backpropagation in CNN



Next, the second neuron of Conv. Layer.

$$\frac{\partial Li}{\partial w1} = \frac{\overset{0}{\overset{\downarrow}{\partial Li}}}{\partial f} . \frac{\partial f}{\partial w1}$$

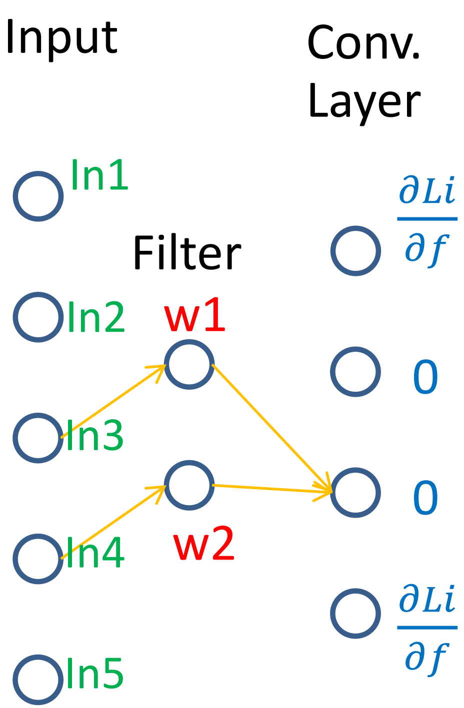
$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} . \frac{\partial (In2.w1+In3.w2)}{\partial w1}$$

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} . In2$$

Similarly,

$$\frac{\partial Li}{\partial w2} = \frac{\partial Li}{\partial f} . In3$$

Backpropagation in CNN



Repeat process for the rest neurons.

$$\frac{\partial Li}{\partial w1} = \overset{\overset{0}{\downarrow}}{\frac{\partial Li}{\partial f}} . \frac{\partial f}{\partial w1}$$

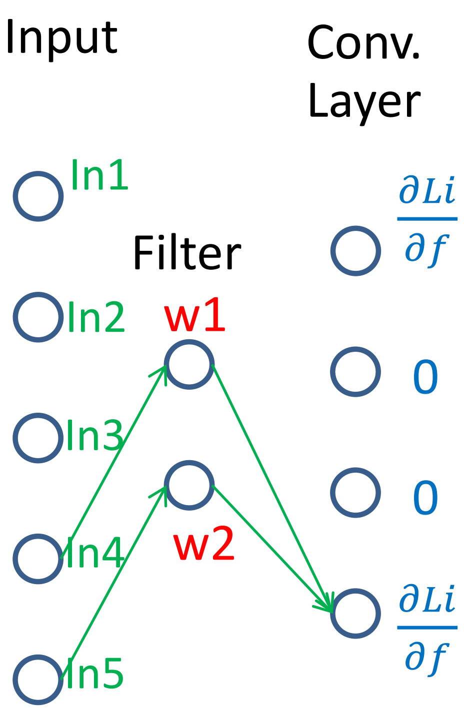
$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} . \frac{\partial (In3.w1+In4.w2)}{\partial w1}$$

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} . In3$$

Similarly,

$$\frac{\partial Li}{\partial w2} = \frac{\partial Li}{\partial f} . In4$$

Backpropagation in CNN



Repeat process for the rest neurons.

known

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} \cdot \frac{\partial f}{\partial w1}$$

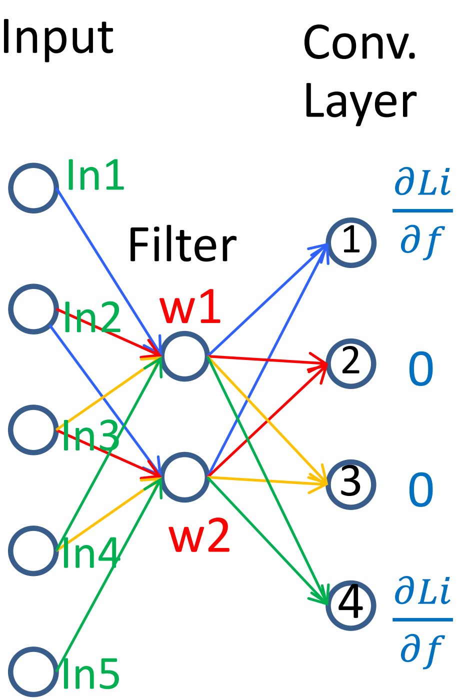
$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} \cdot \frac{\partial (In4.w1 + In5.w2)}{\partial w1}$$

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f} . In4$$

Similarly,

$$\frac{\partial Li}{\partial w2} = \frac{\partial Li}{\partial f} . In5$$

Backpropagation in CNN



Sum all gradients together.

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f_1}.In1 + \frac{\partial Li}{\partial f_2}.In2 + \frac{\partial Li}{\partial f_3}.In3 + \frac{\partial Li}{\partial f_4}.In4$$

$$\frac{\partial Li}{\partial w2} = \frac{\partial Li}{\partial f_1}.In2 + \frac{\partial Li}{\partial f_2}.In3 + \frac{\partial Li}{\partial f_3}.In4 + \frac{\partial Li}{\partial f_4}.In5$$

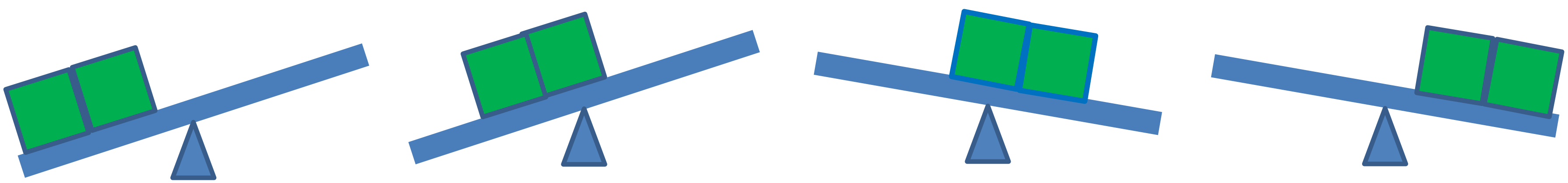
Update w1 and w2 according to the delta rule

$$w = w - \propto (\underbrace{\frac{\partial Li}{\partial w}}_{\text{Data loss}} + \underbrace{\lambda w}_{\text{Regularization loss}})$$

Done! We just updated all weights in CNN.

SIMPLE CASE STUDY

- Let’s consider a seesaw problem
  - A long, narrow board supported by a single pivot point.
  - There are two boxes, each has the same weight and size.
  - There are five position on the board which we can place two boxes.
  - Two boxes needs to be adjacent when placed on the board.
  - There are two states of the board : tilt left, tilt right



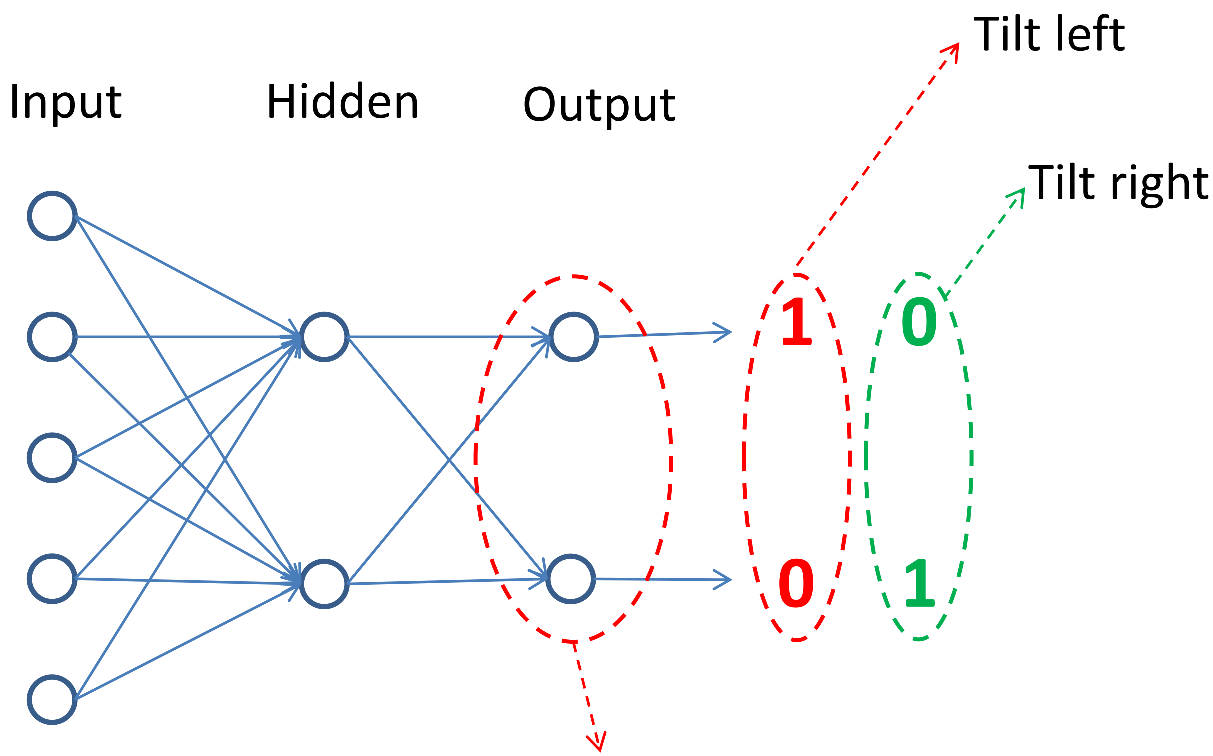
- Dataset

Position1	Position2	Position3	Position4	Position5	State/Status
1	1	0	0	0	0
0	1	1	0	0	0
0	0	1	1	0	1
0	0	0	1	1	1

0 = Tilt left
1 = Tilt right

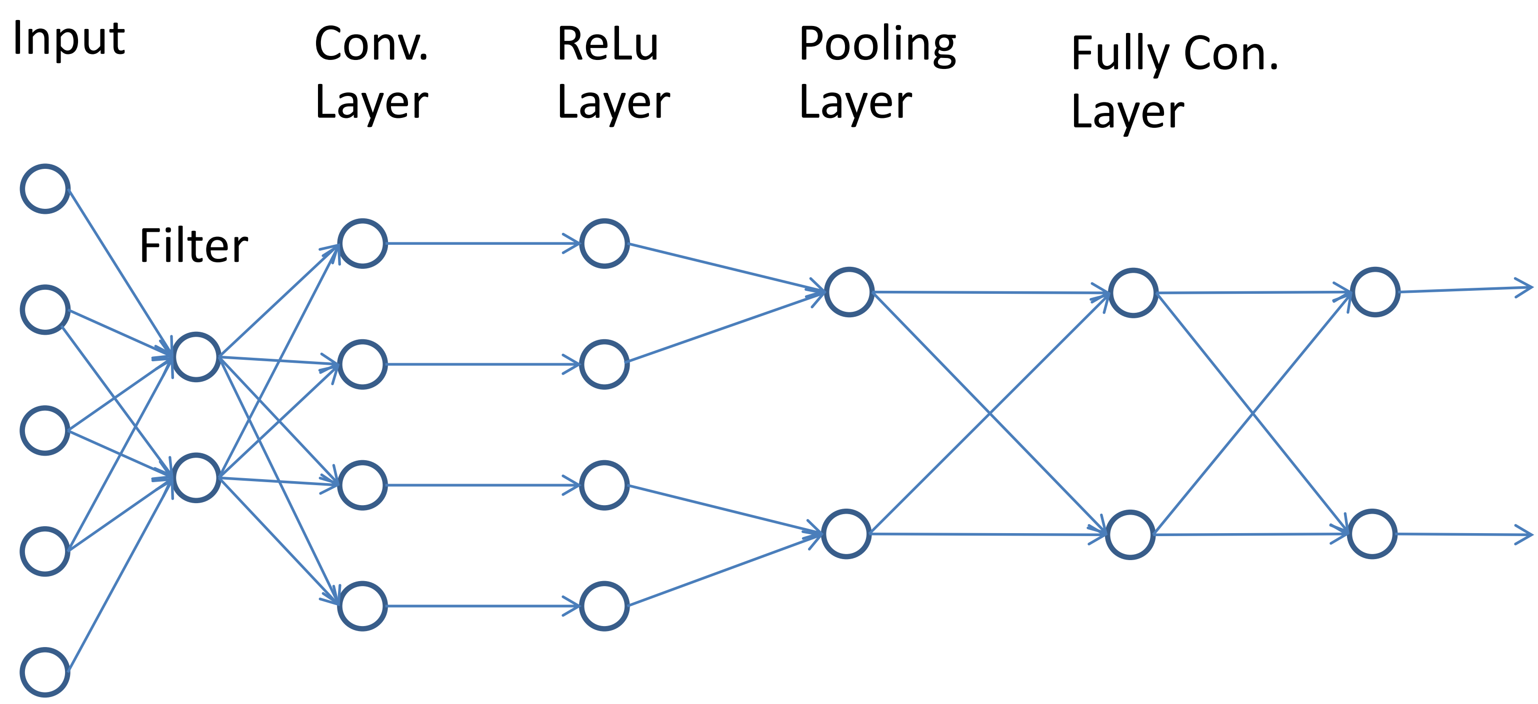
We will compare architectures of Multilayer NN and Conv. NN for the seesaw problem.

Architecture of Multilayer NN for seesaw problem



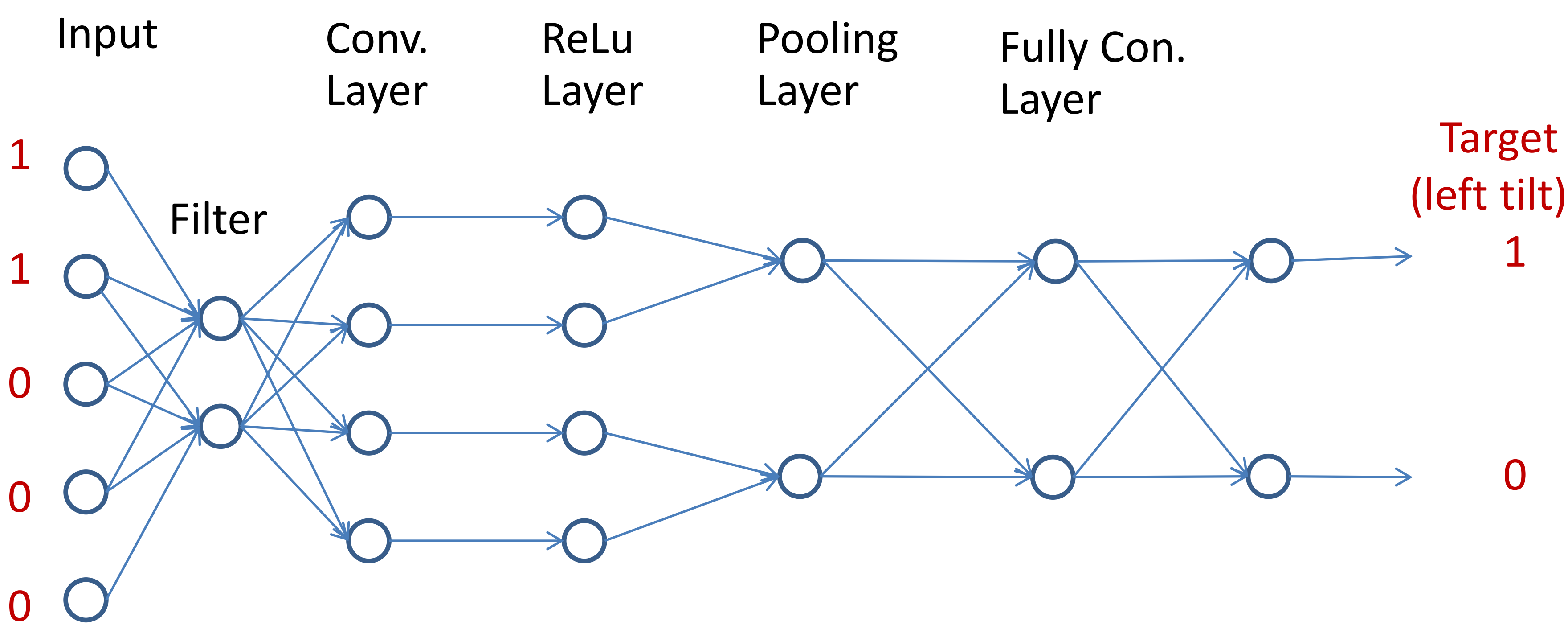
Activation function for output layer can be **sigmoid** or **softmax**.

Architecture of Convolutional NN for seesaw problem



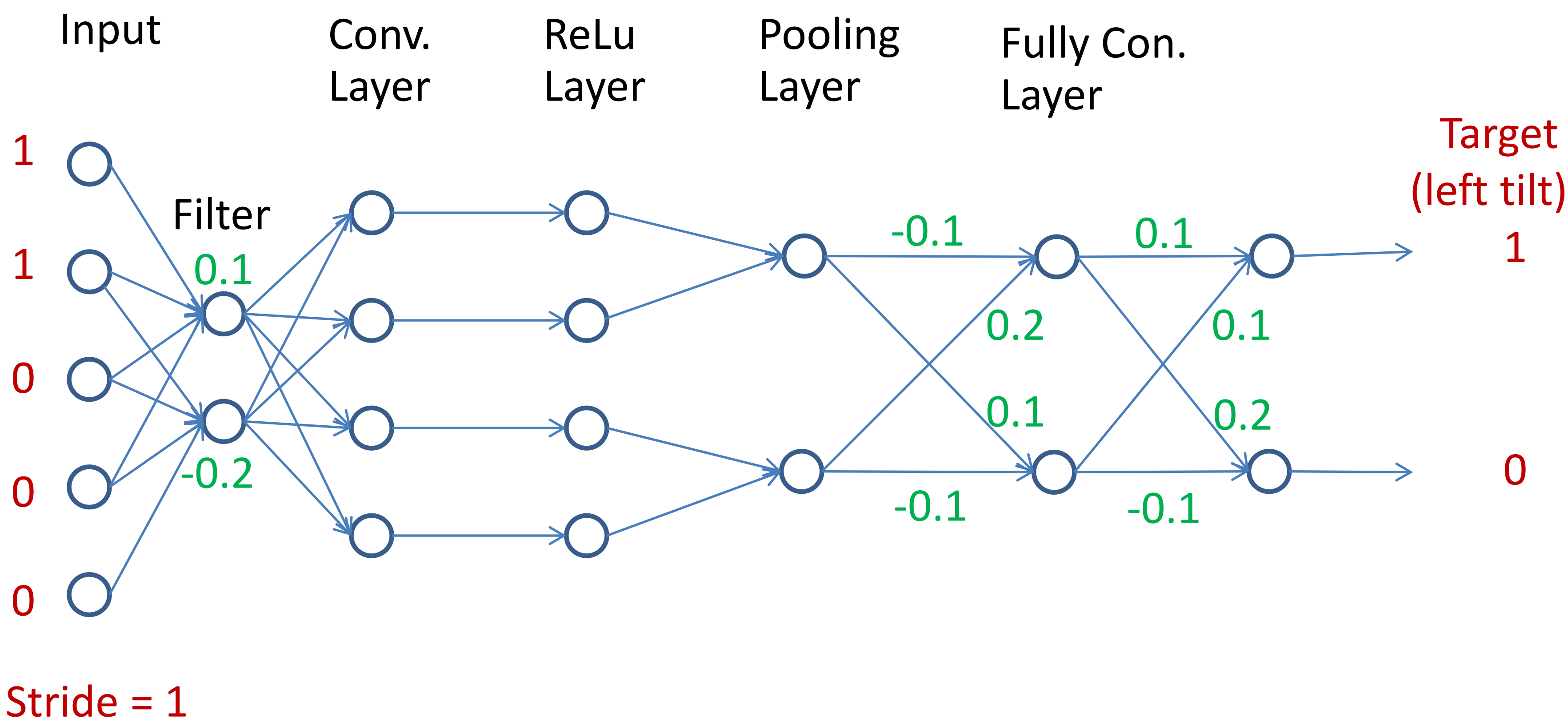
Stride = 1

Convolutional NN : Sample run

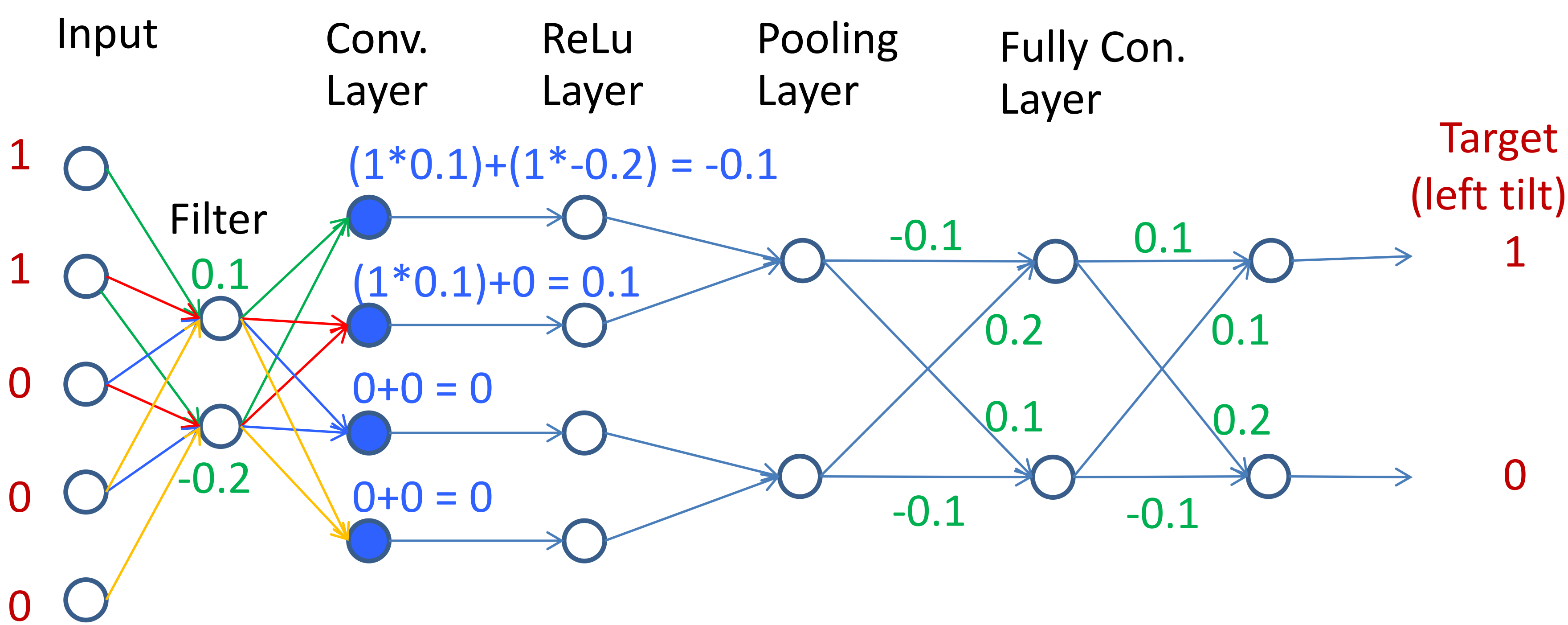


Stride = 1

Convolutional NN : Weight initialization

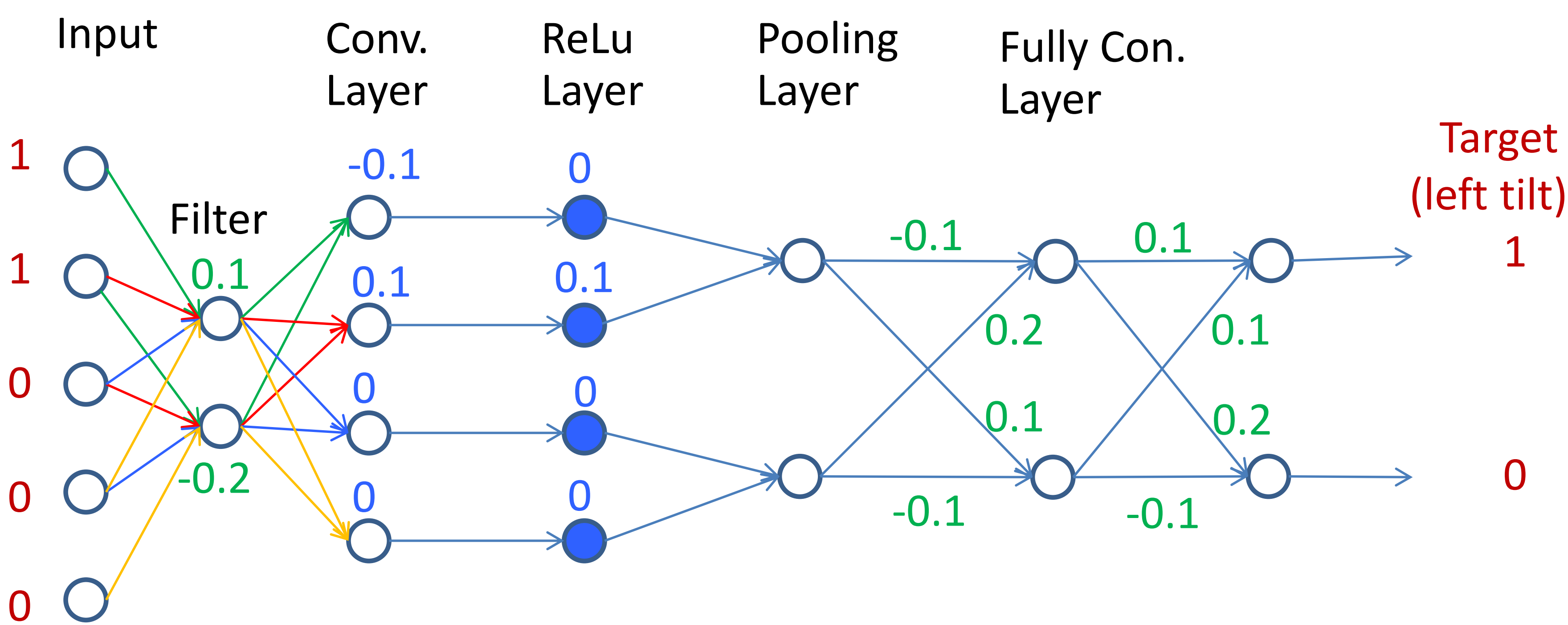


Convolutional NN : Convolution



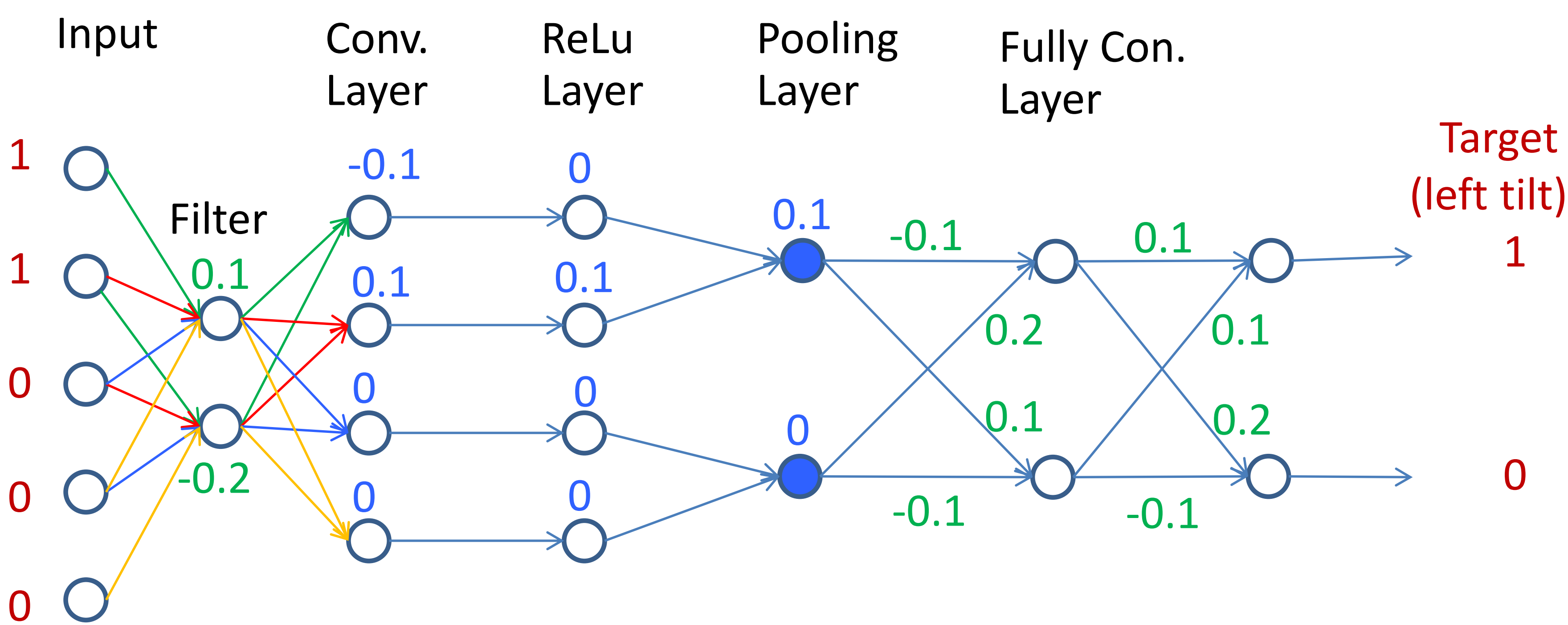
Stride = 1

Convolutional NN : ReLU



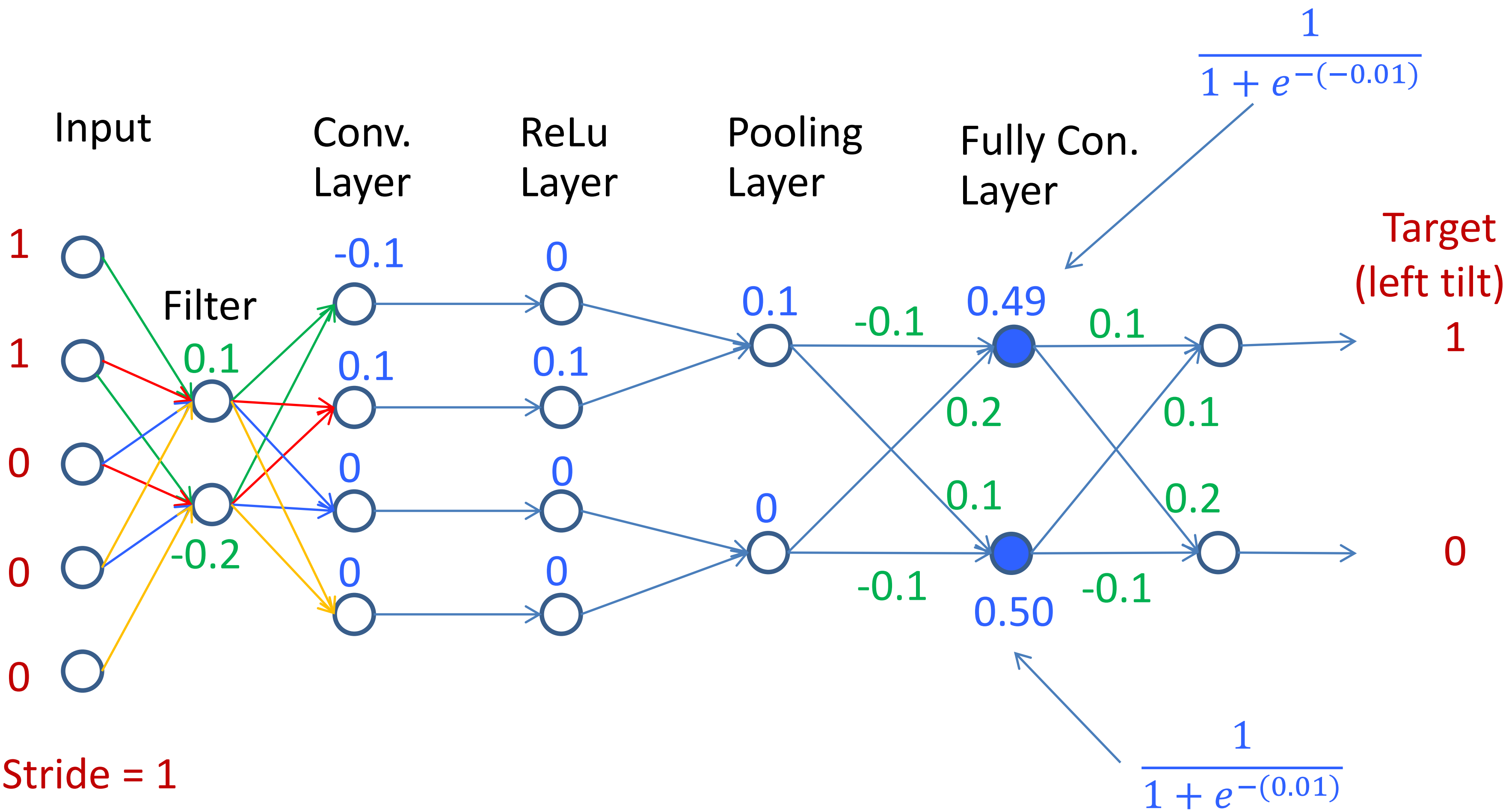
Stride = 1

Convolutional NN : Pooling (Max)



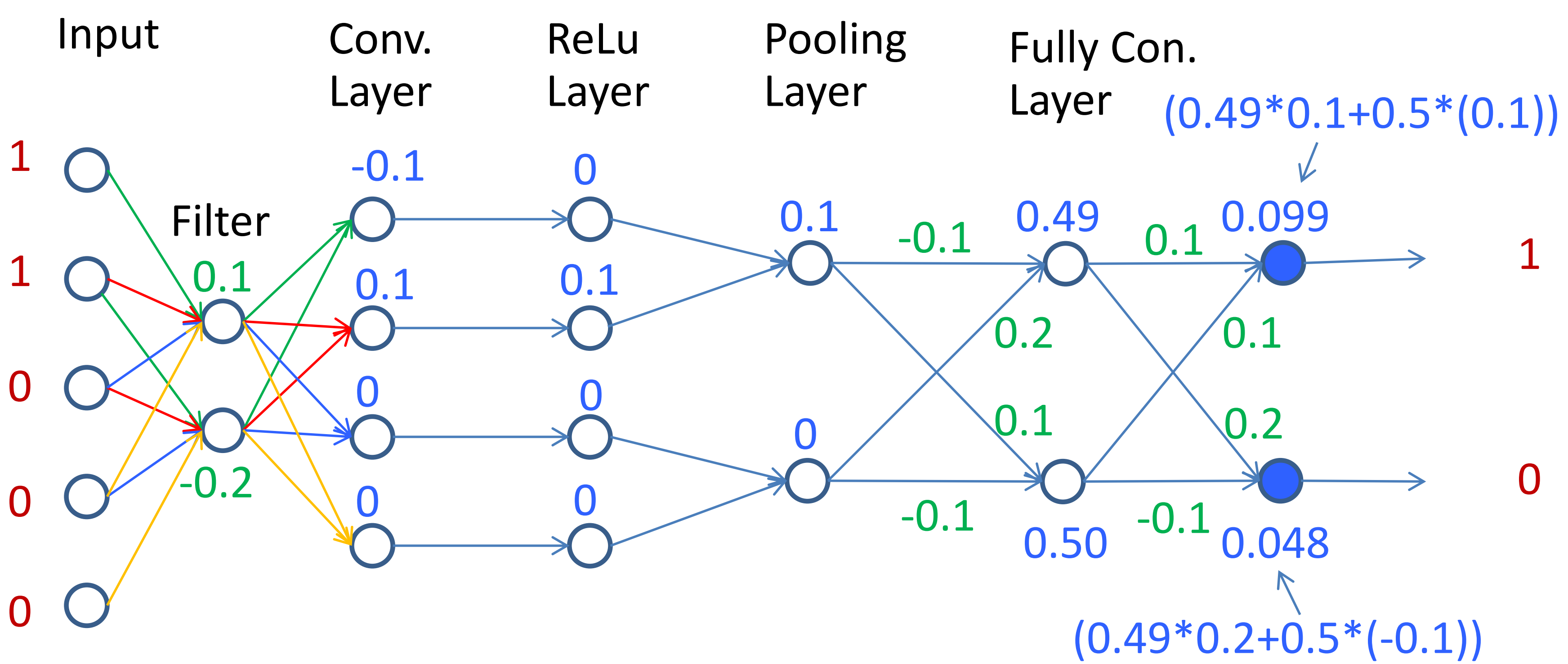
Stride = 1

Convolutional NN : Fully Con.



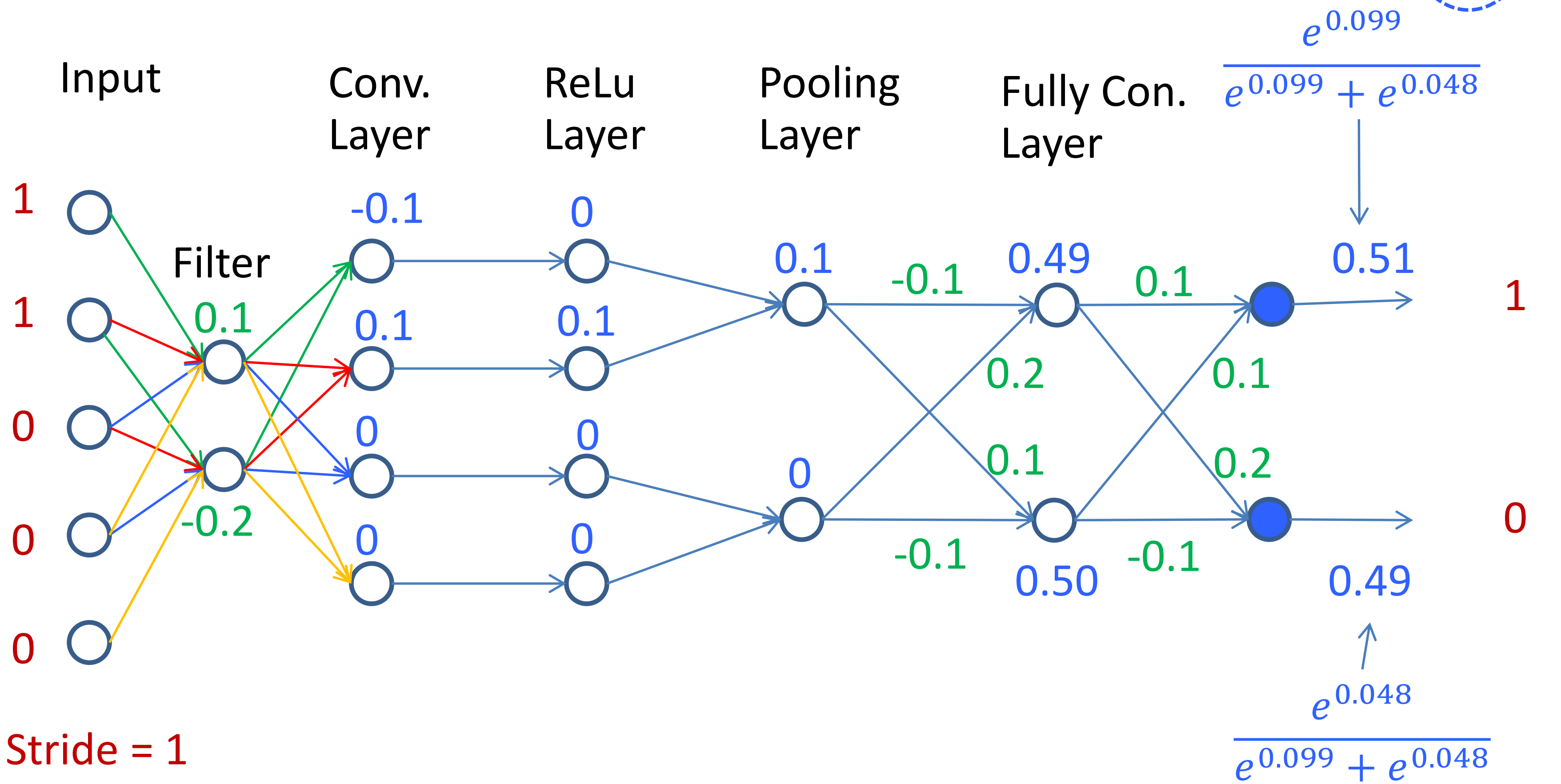
Stride = 1

Convolutional NN : Output layer (Score function)



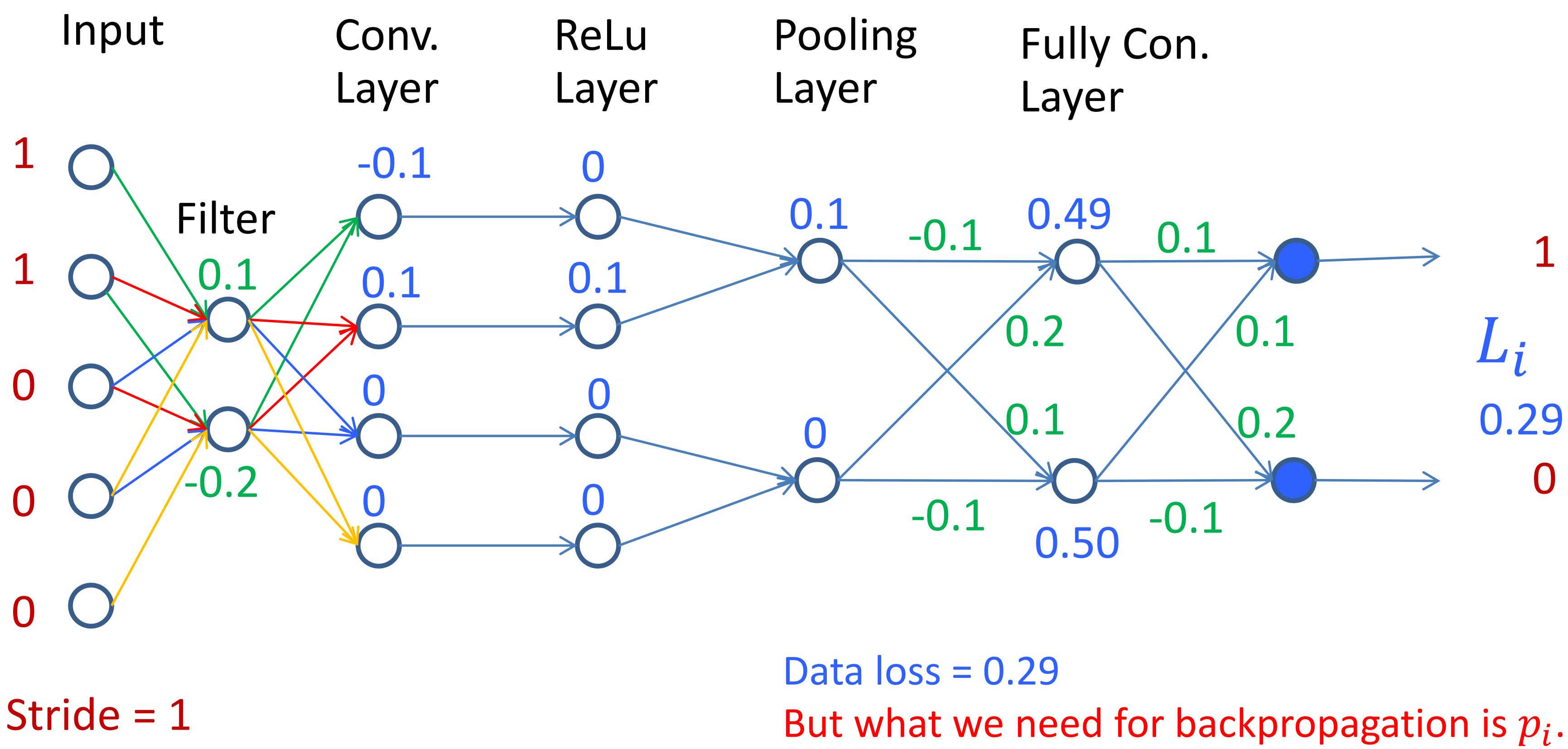
Stride = 1

Convolutional NN : Data Loss (Exponential) $L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$



Convolutional NN : Data Loss (take -log)

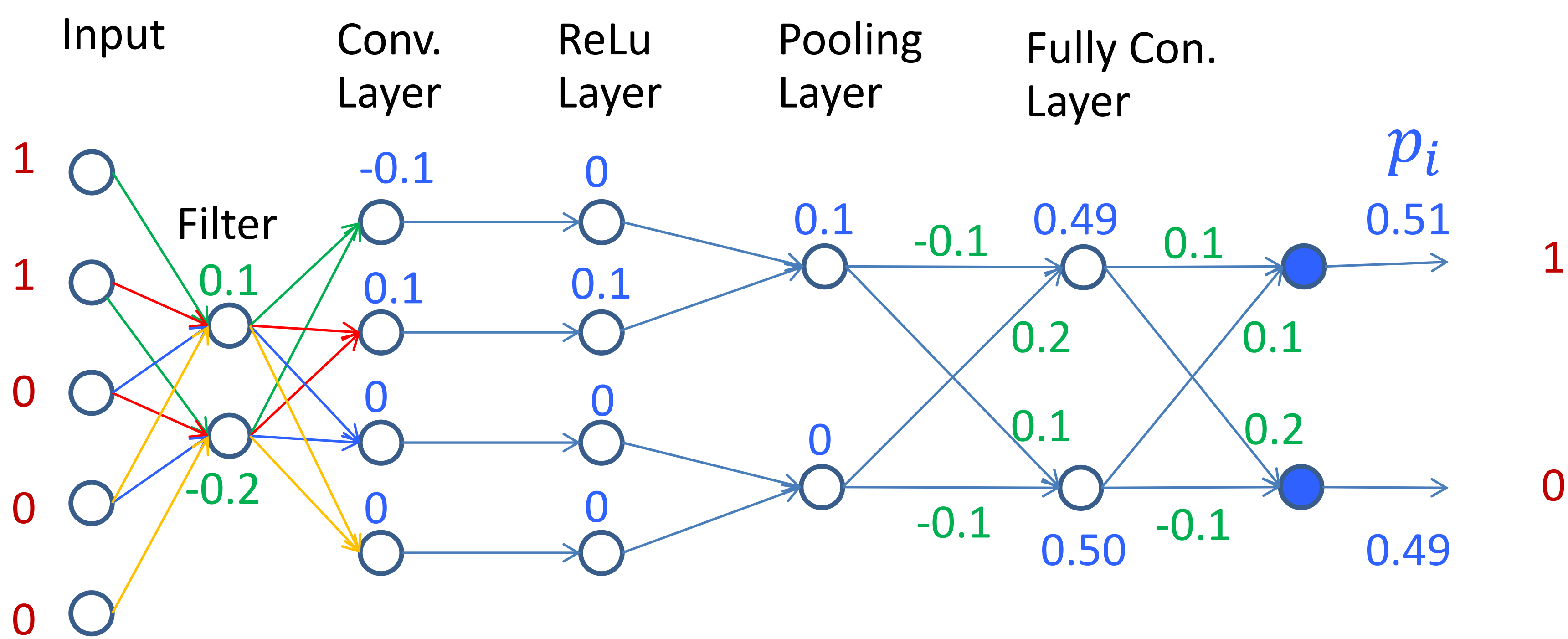
$$L_i = -\log\left(\frac{e^{f_{target}}}{\sum_j e^{f_j}}\right)$$



Data loss = 0.29

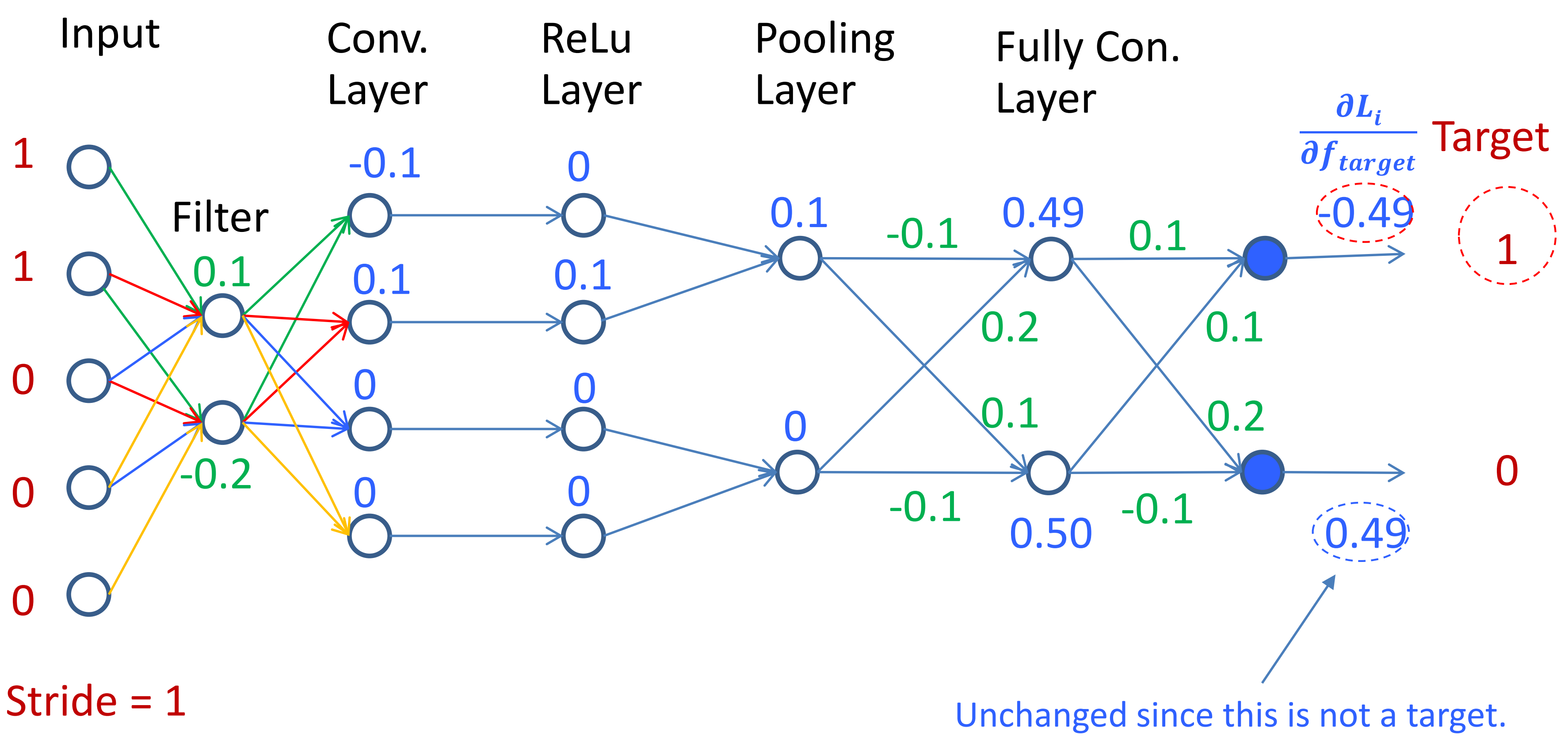
But what we need for backpropagation is p_i.

Convolutional NN : Preparation for backpropagation.



Stride = 1

Convolutional NN : Calculate $\frac{\partial L_i}{\partial f_{target}} = (p_{target} - 1)$

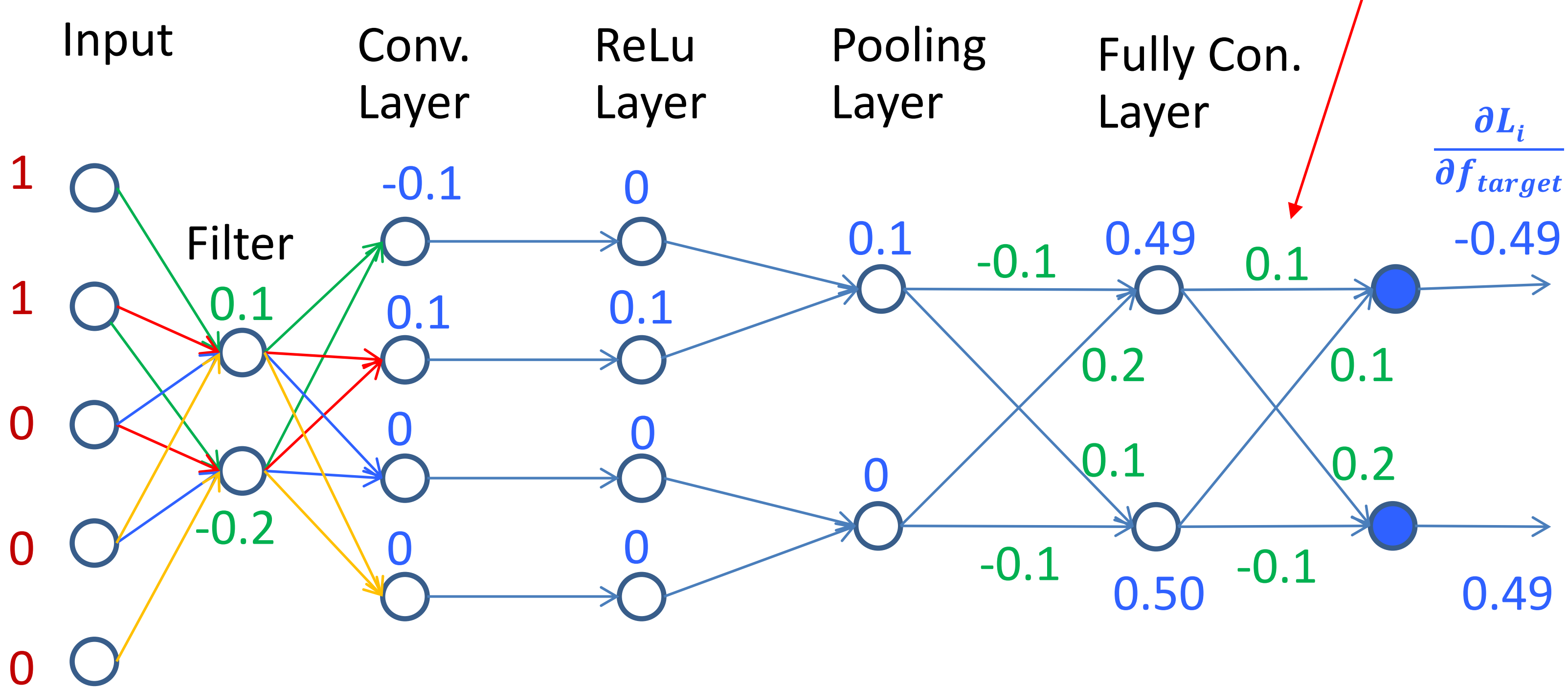


$$w = w \overbrace{- \propto (\underbrace{\frac{\partial L}{\partial f}}_{(p_{target} - 1)} . \underbrace{\frac{\partial f}{\partial w}}_{Out} + \lambda w)}^{\Delta w}$$

Assume $\propto$ *and* λ equal 0.1.

Convolutional NN :

$$\Delta w = -0.1 * (-0.49 * 0.49 + 0.1 * 0.1) = 0.023$$

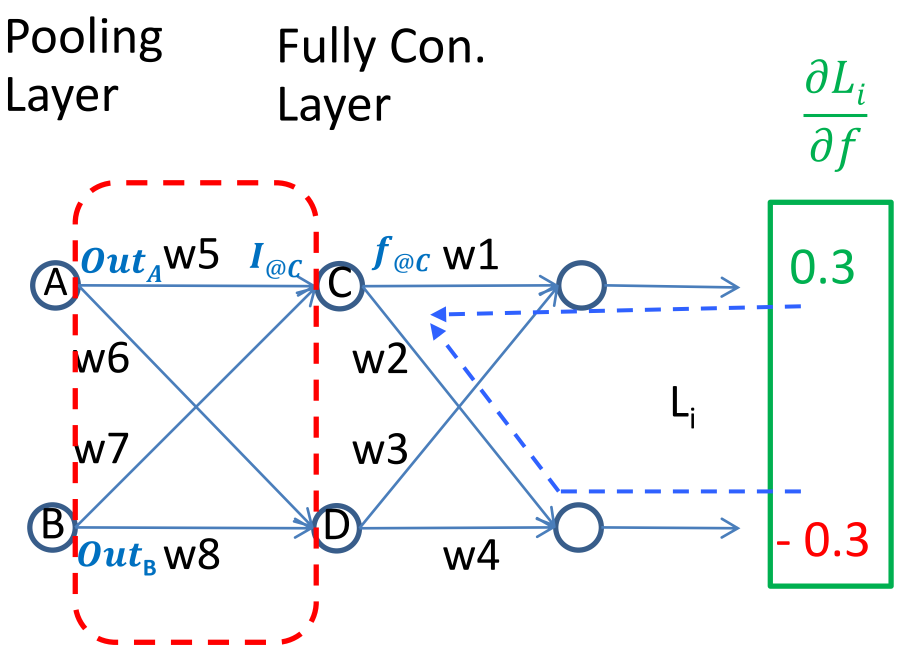


Stride = 1

Calculate Δw for other weights in the same layer, but do not update weights (hold them until the last step).

Recall: Backpropagation in CNN (page 21)

Updating weights in the inner layer (w5-w8)



$$w_5 = w_5 - \propto (\frac{\partial L}{\partial w_5})$$

$$w_5 = w_5 - \propto (\frac{\partial Li}{\partial w_5} + \lambda w_5)$$

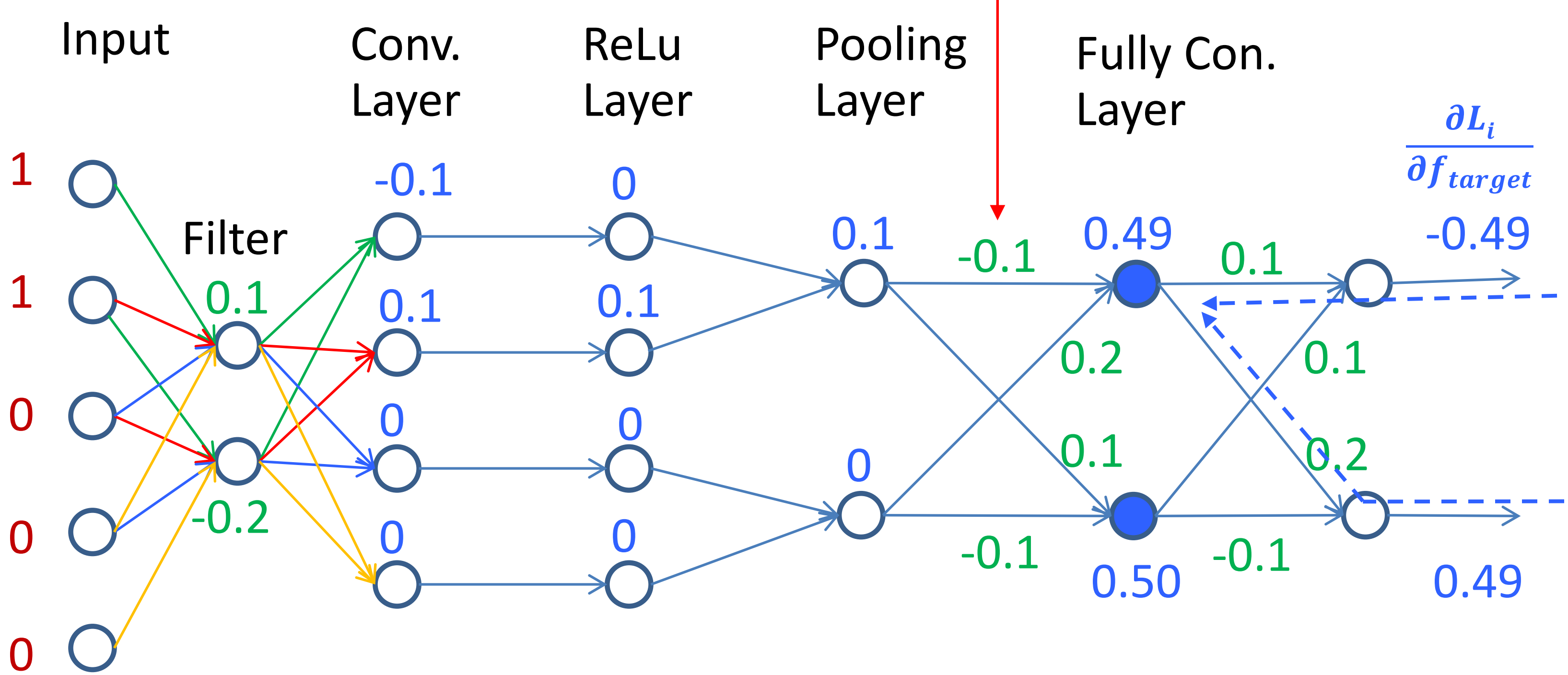
$$\frac{\partial Li}{\partial w_5} = \frac{\partial Li}{\partial f_{@c}} . \frac{\partial f_{@c}}{\partial I_{@c}} . \frac{\partial I_{@c}}{\partial w_5}$$

$$\frac{\partial Li}{\partial w_5} = \sum(\frac{\partial L_i}{\partial f} . w_{@c}) . f@c.* (1 - f@c). Out_A$$

$$\frac{\partial Li}{\partial w_5} = ((0.3 * w1) + (-0.3 * w2)). f@c.* (1 - f@c). Out_A$$

$$\Delta w = \overbrace{-0.1}^{\alpha} * (\overbrace{[-0.49 * 0.1 + 0.49 * 0.2]}^{\sum(\frac{\partial L_i}{\partial f}.w_{@c})} * \overbrace{[0.49 * (1 - 0.49)]}^{f@c.*(1-f@c)} * \overbrace{0.1}^{Out_A} + \overbrace{0.1*(-0.1)}^{\lambda w}) = 0.000878$$

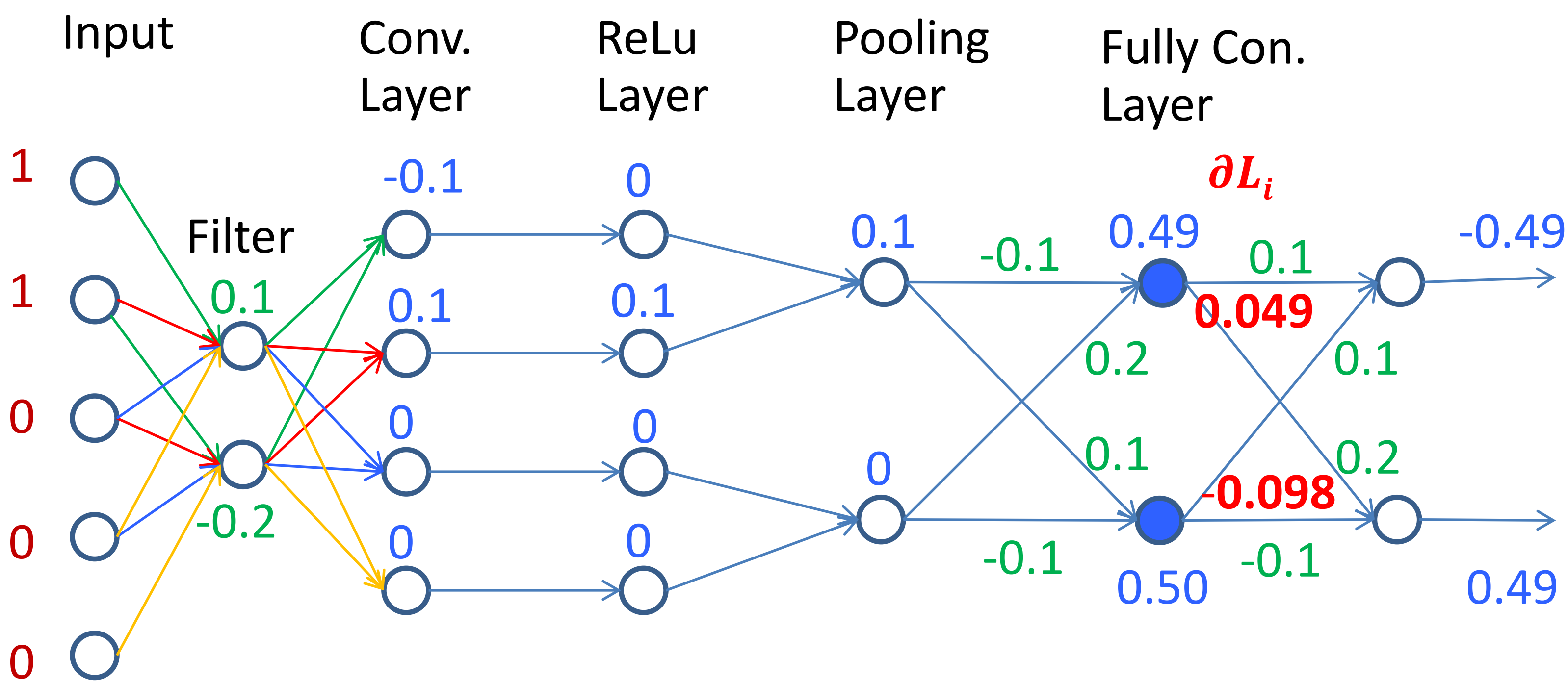
Convolutional NN :



Stride = 1

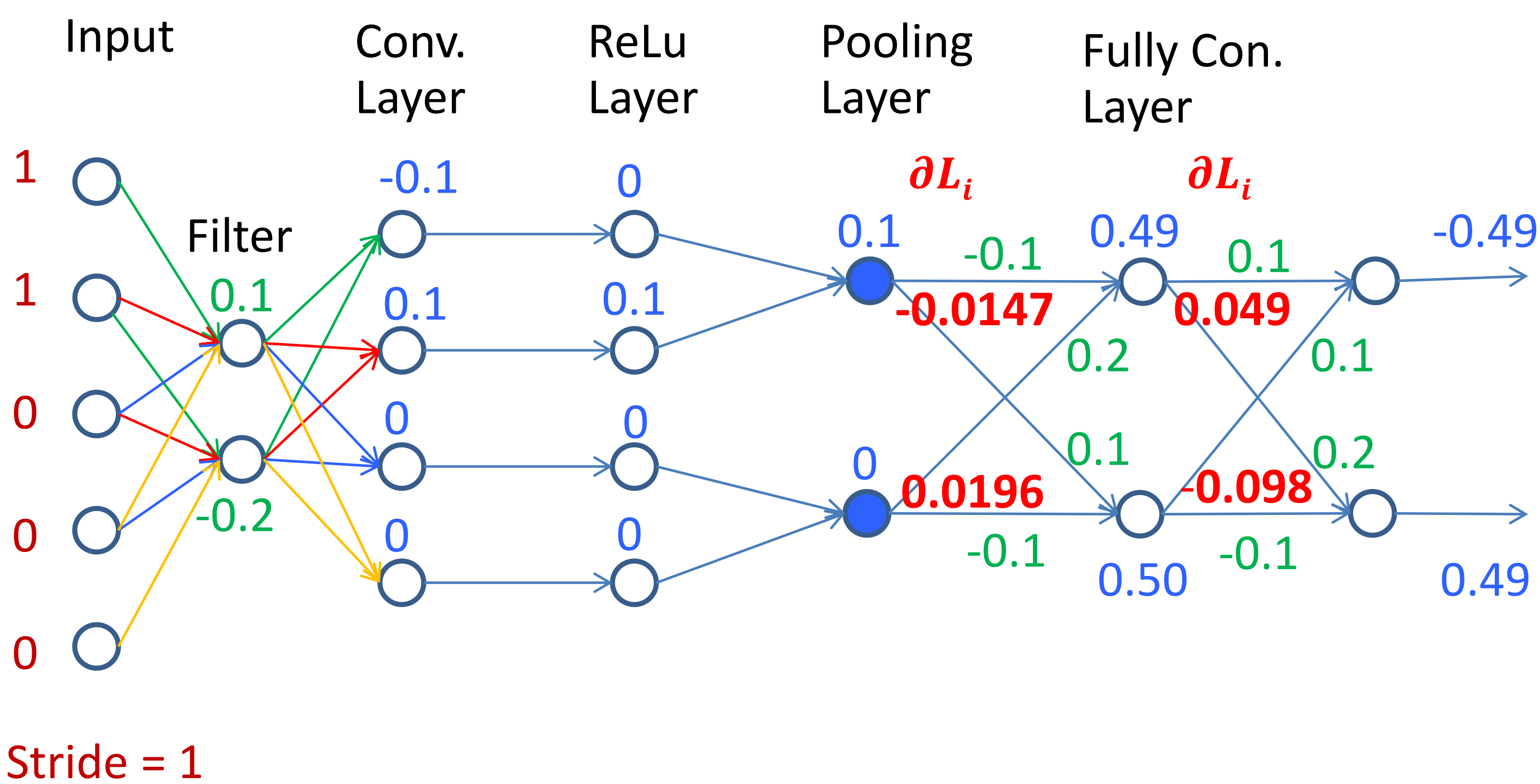
Calculate Δw for other weights in the same layer, but do not update weights (hold them until the last step).

Convolutional NN :

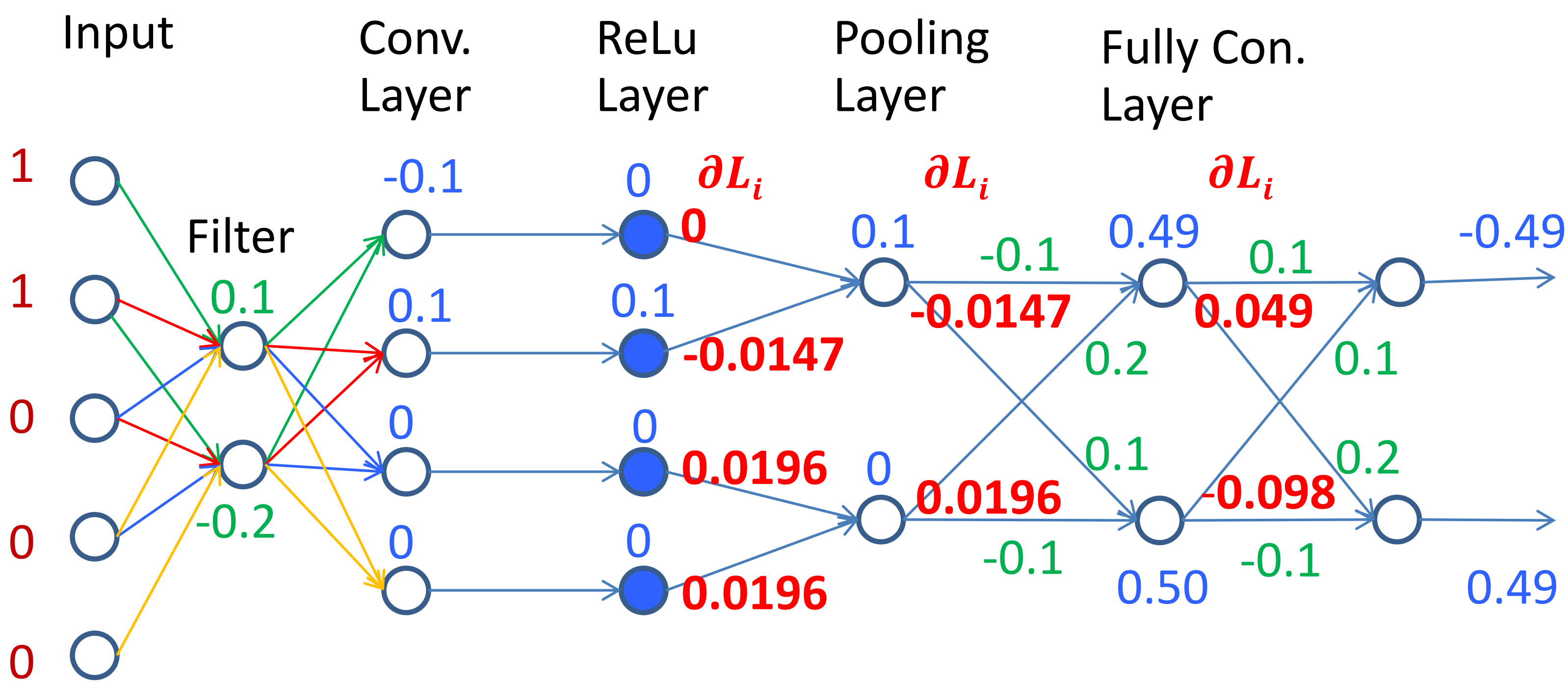


Stride = 1

Convolutional NN : Gradient of Pooling layer

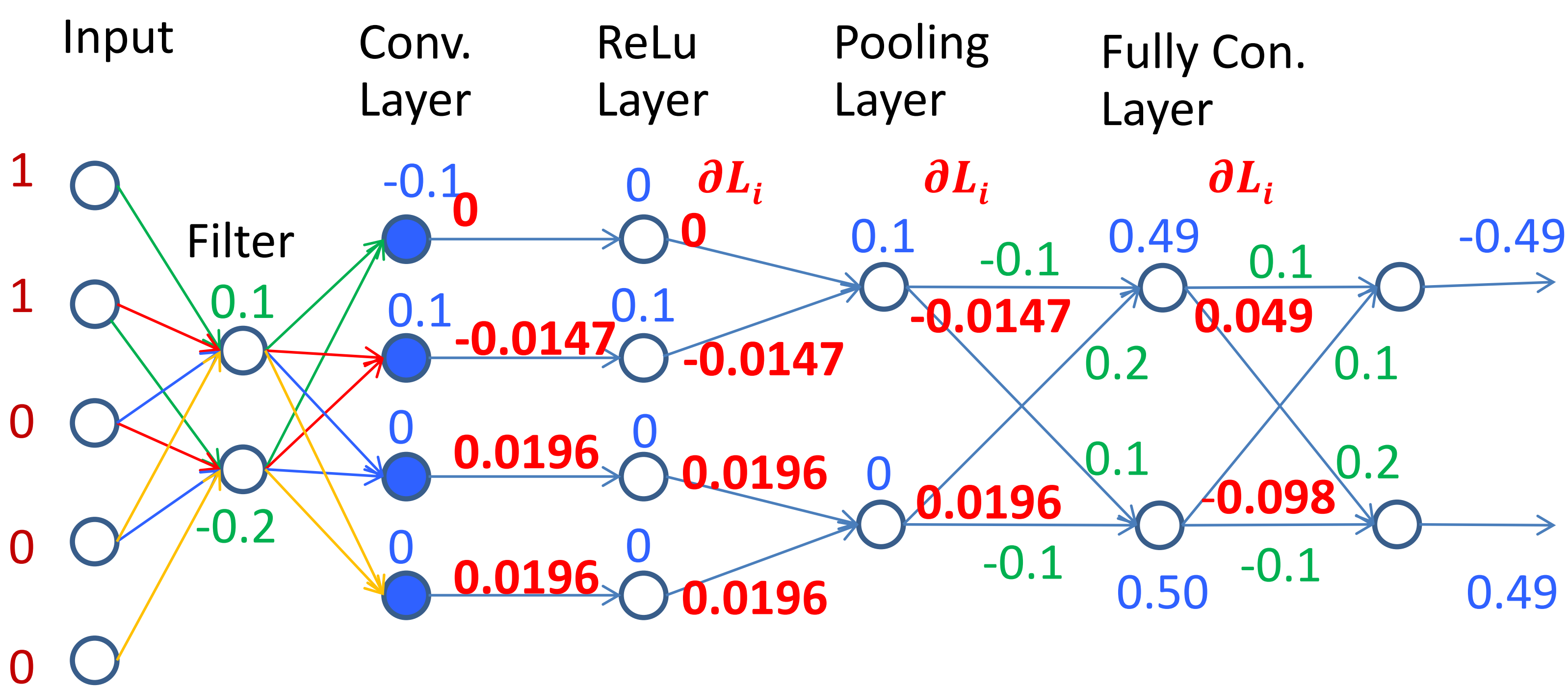


Convolutional NN : Gradient of ReLU layer



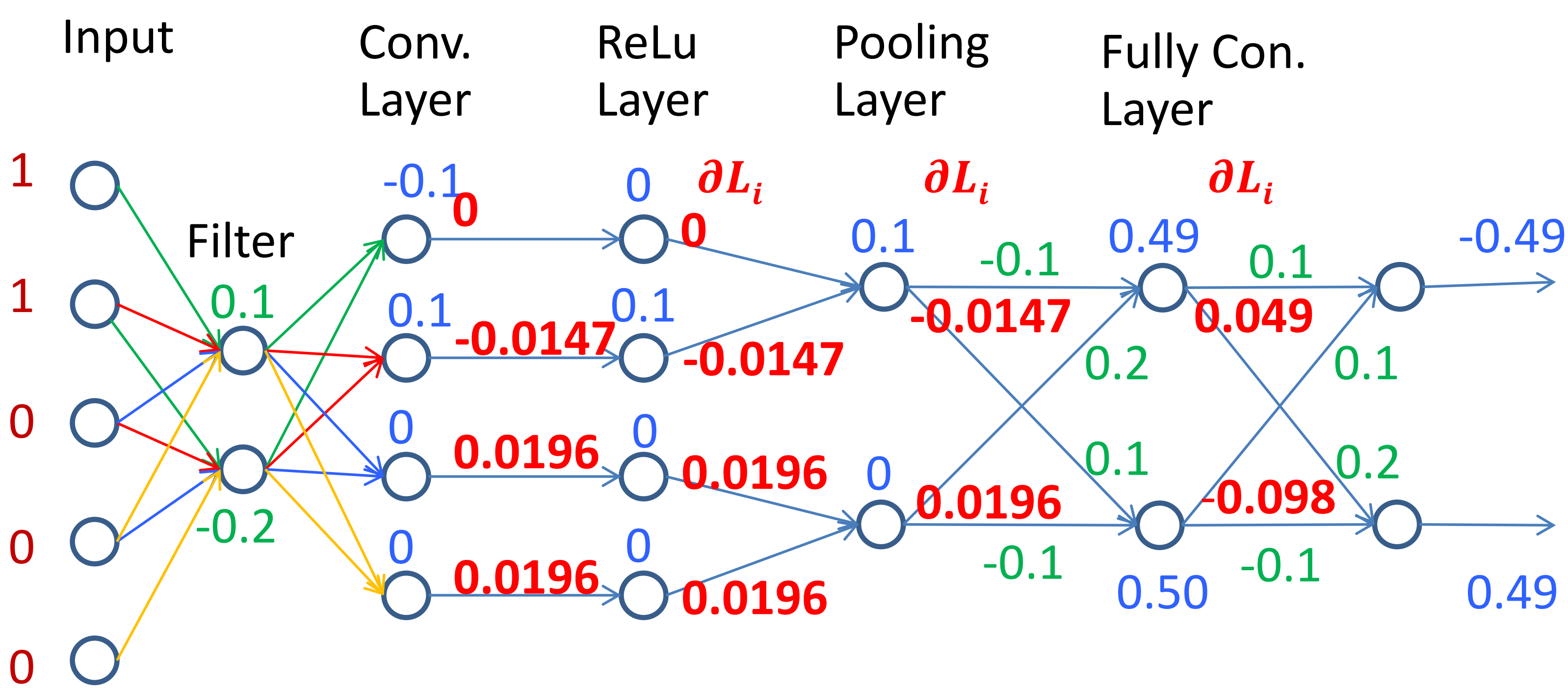
Stride = 1

Convolutional NN : Gradient of Conv. layer



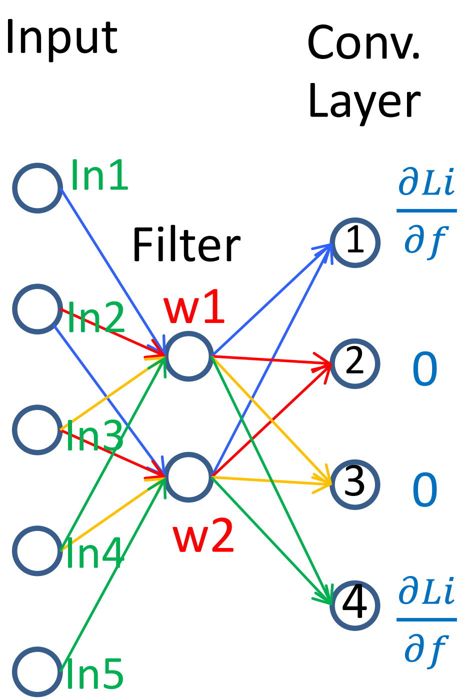
Stride = 1

Convolutional NN : Update weights in the filter.



Stride = 1

Recall: page 33



Sum all gradients together.

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f_1}.In1 + \frac{\partial Li}{\partial f_2}.In2 + \frac{\partial Li}{\partial f_3}.In3 + \frac{\partial Li}{\partial f_4}.In4$$

$$\frac{\partial Li}{\partial w2} = \frac{\partial Li}{\partial f_1}.In2 + \frac{\partial Li}{\partial f_2}.In3 + \frac{\partial Li}{\partial f_3}.In4 + \frac{\partial Li}{\partial f_4}.In5$$

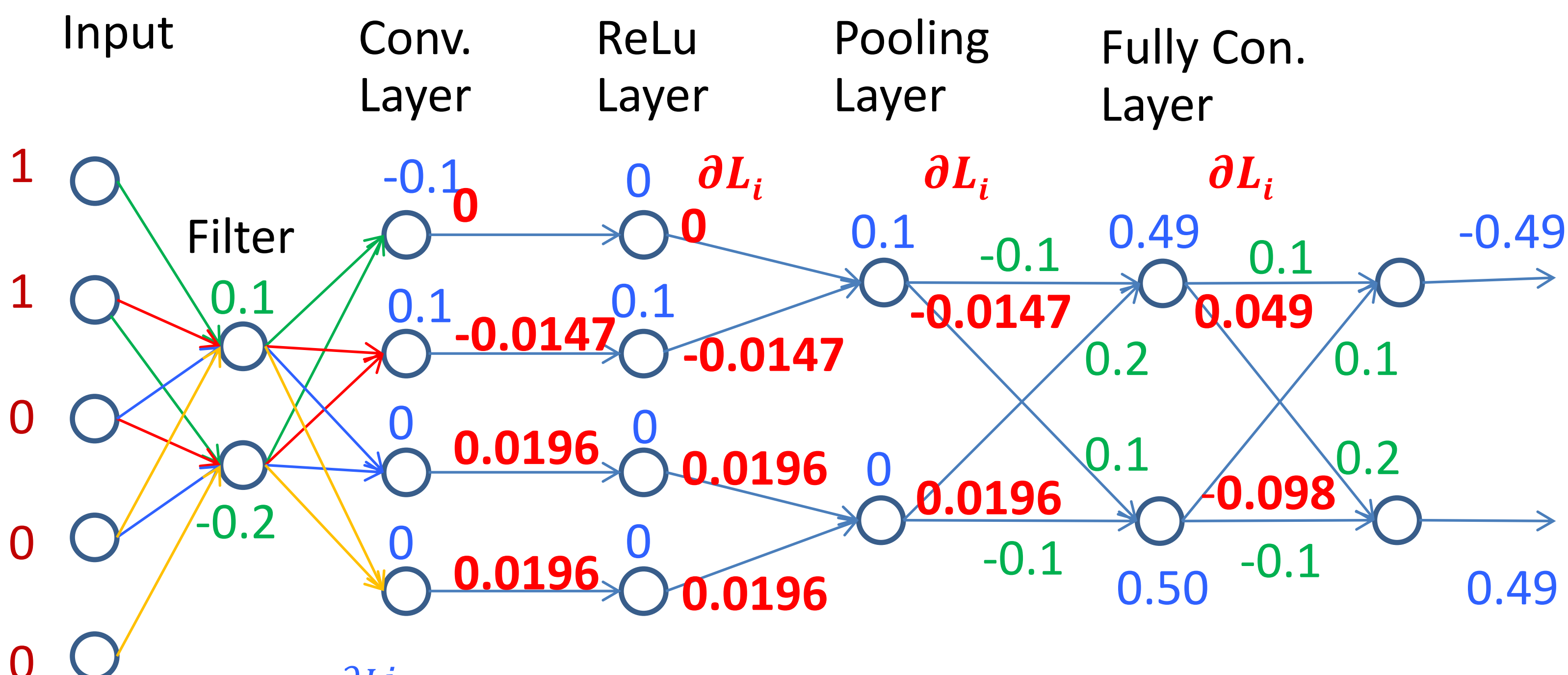
Update w1 and w2 according to the delta rule

$$w = w - \propto (\underbrace{\frac{\partial Li}{\partial w}}_{\text{Data loss}} + \underbrace{\lambda w}_{\text{Regularization loss}})$$

$$\frac{\partial Li}{\partial w1} = \frac{\partial Li}{\partial f_1}.In1 + \frac{\partial Li}{\partial f_2}.In2 + \frac{\partial Li}{\partial f_3}.In3 + \frac{\partial Li}{\partial f_4}.In4$$

$$\frac{\partial Li}{\partial w2} = \frac{\partial Li}{\partial f_1}.In2 + \frac{\partial Li}{\partial f_2}.In3 + \frac{\partial Li}{\partial f_3}.In4 + \frac{\partial Li}{\partial f_4}.In5$$

Convolutional NN : Calculate Δw in the filter.

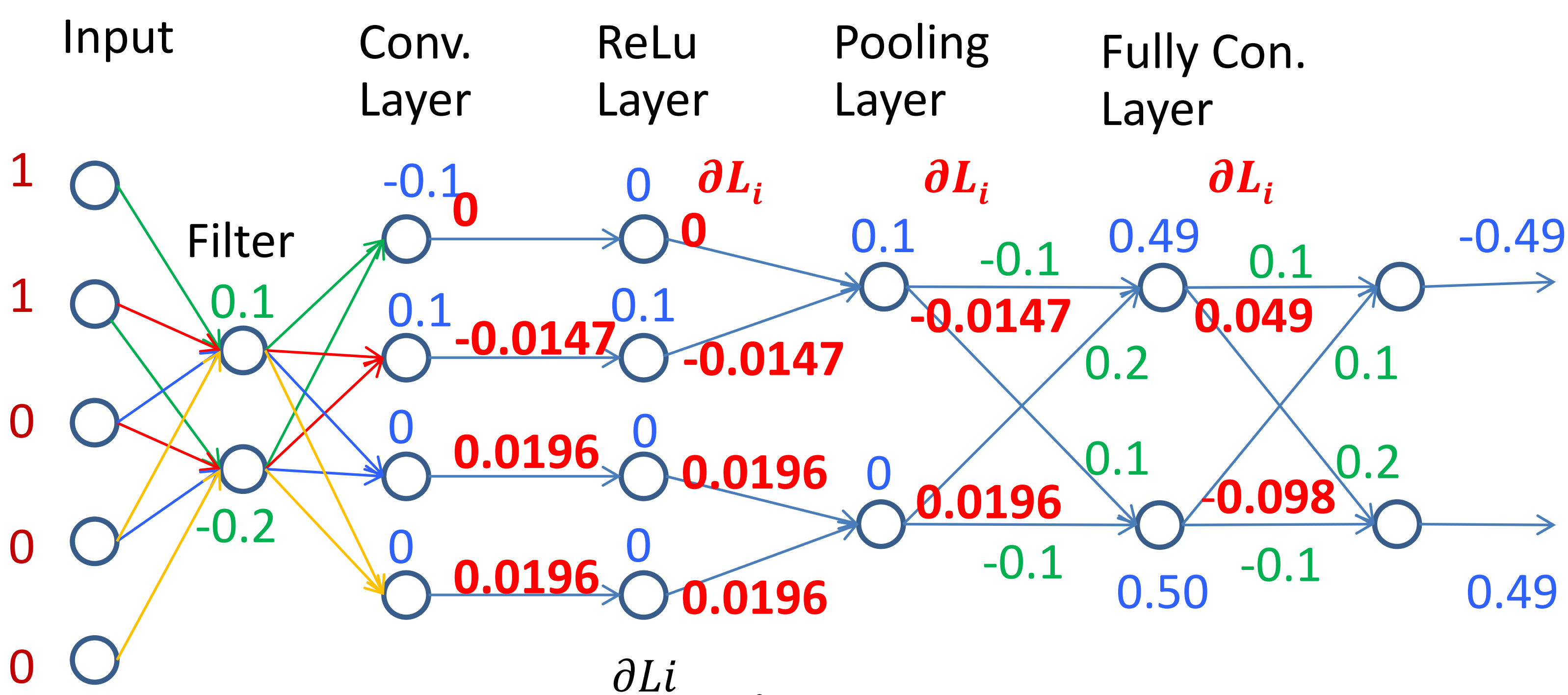


Stride = 1

$$\frac{\partial Li}{\partial w1} = (0*1)+(-0.0147*1)+(0.0196*0)+(0.0196*0) = -0.0147$$

$$\frac{\partial Li}{\partial w2} = (0*1)+(-0.0147*0)+(0.0196*0)+(0.0196*0) = 0$$

Convolutional NN : Calculate Δw in the filter.



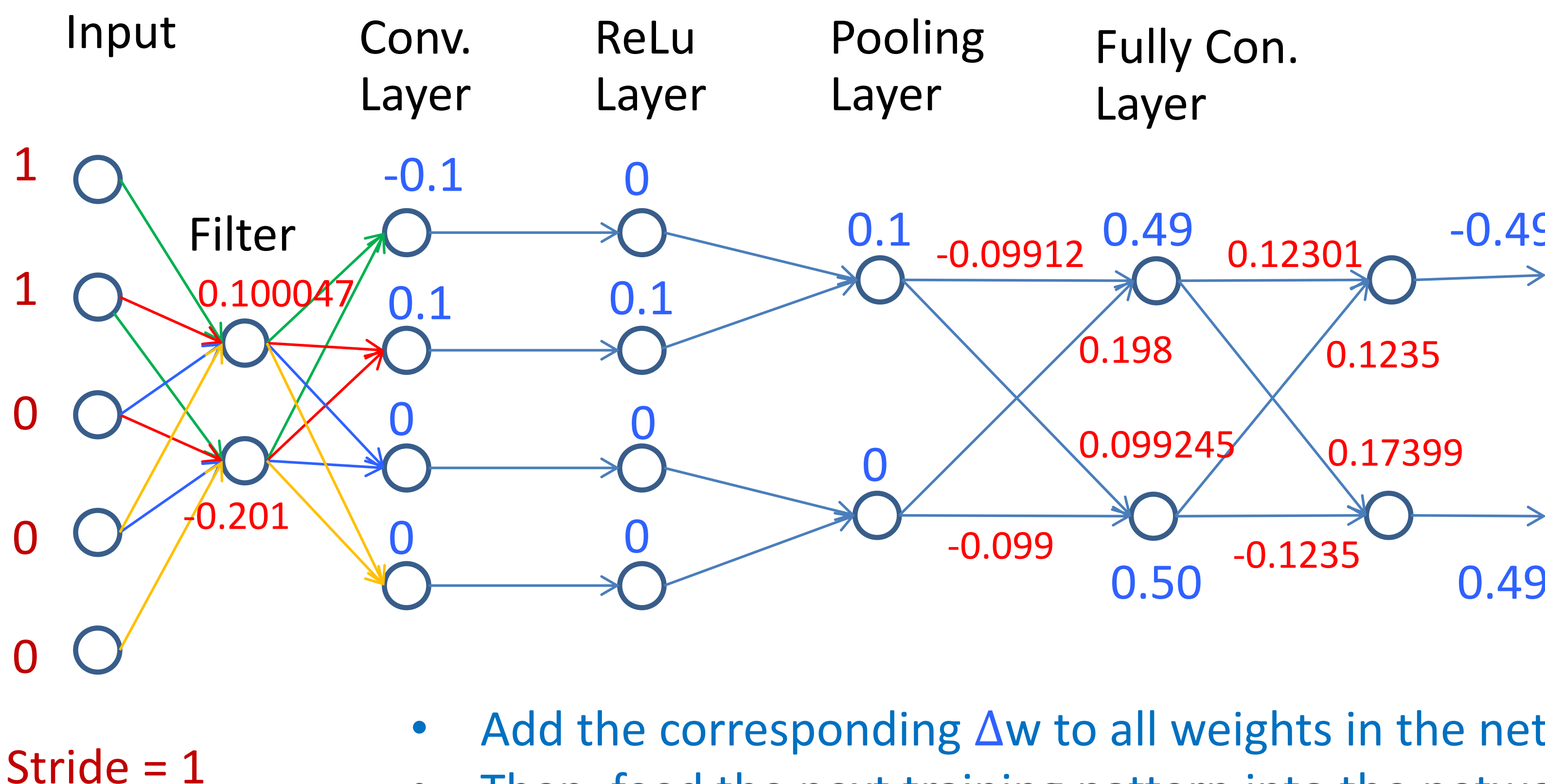
Stride = 1

$$\Delta w = -\propto \left(\frac{\partial Li}{\partial w} + \lambda w\right)$$

Δw1 = – 0.1(-0.0147+0.1*0.1) = 0.00047

Δw2 = – 0.1(0+0.1*0.1) = -0.001

Convolutional NN : Update all weights in the network.



- Add the corresponding Δw to all weights in the network.
- Then, feed the next training pattern into the network.

	Multilayer NN with Sigmoid Output	**Multilayer NN with Softmax Output**	**Convolutional NN with Softmax Output**
SSE	0.023	**0.00**	**0.00**
#epoch	1000	324	**261**

- Next class
  - Implementing Convolutional Neural Network by Keras.