

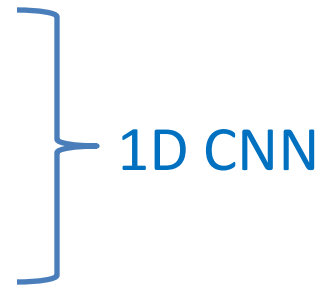
Forward & Backward Computing in Convolutional Neural Network

Minlie Huang

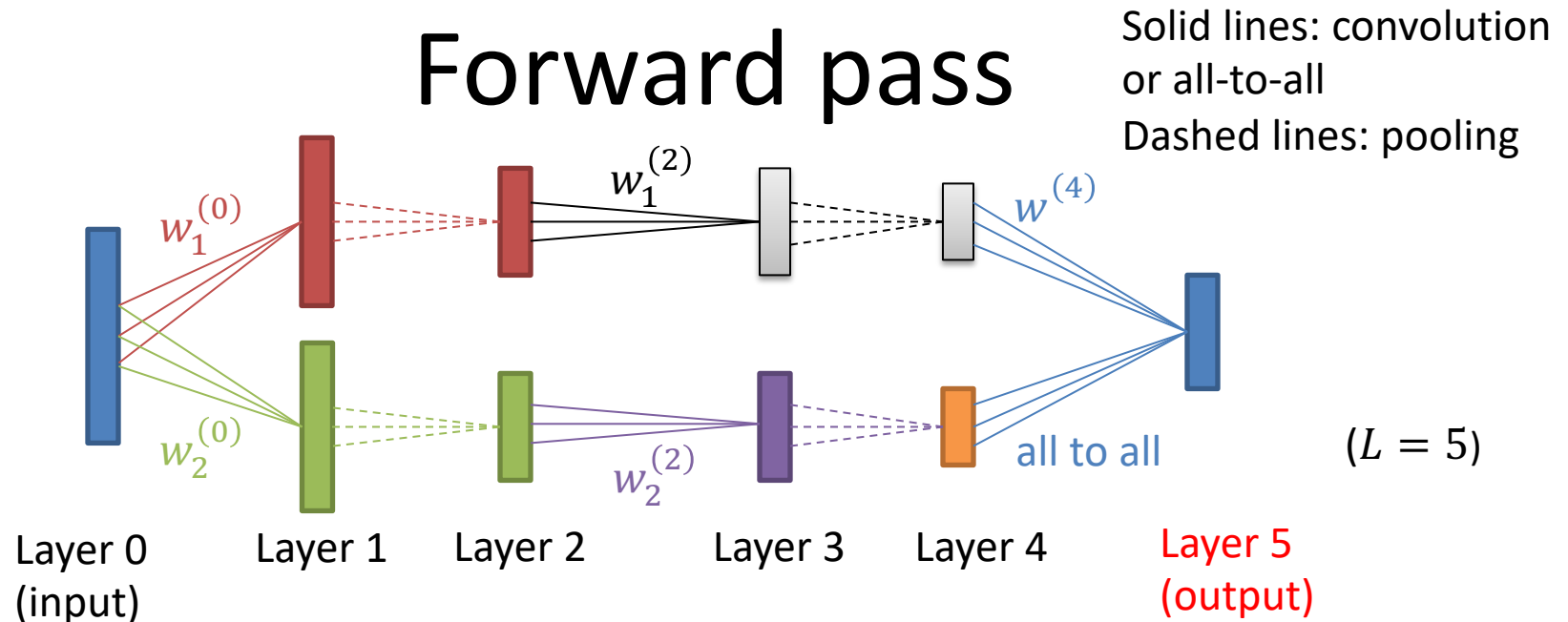
aihuang@Tsinghua.edu.cn

Dept. of Computer Science and Technology
Tsinghua University

Outline

- Forward pass
 - Backward pass
 - Feature combination
 - 2D CNN
- 
- 1D CNN

Forward pass



- For layer $l = 0: 1: L - 2$, do
 - If the l -th layer is a **convolution layer**, convolve every filter $w_p^{(l)}$ with the current **feature map(s)** and obtain **a new feature map**

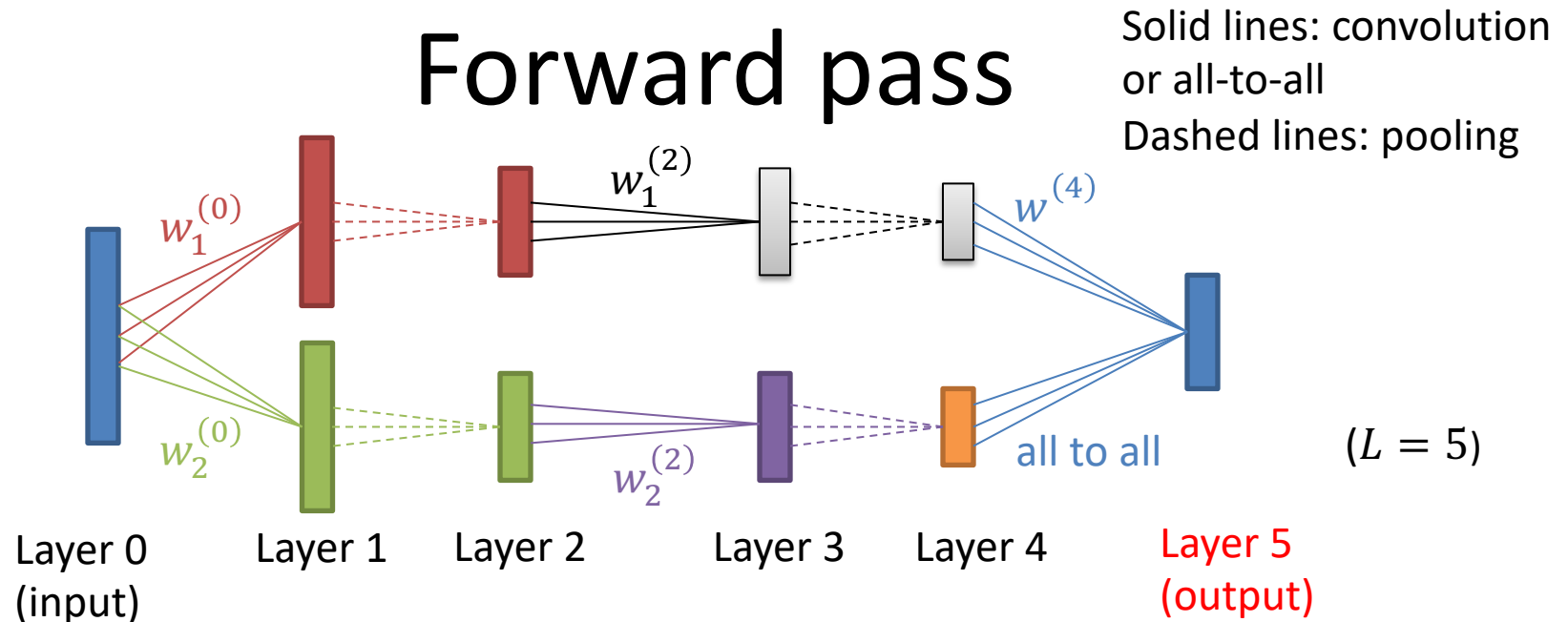
$$y_p^{(l+1)} = f \left(y_p^{(l)} *_{\text{valid}} \text{rot180} \left(w_p^{(l)} \right) + b_p \right)$$

where f is the activation function $u_p^{(l+1)}$

- If the l -th layer is a pooling layer, perform pooling

$$y_p^{(l+1)} = \text{pooling} \left(y_p^{(l)} \right)$$

Forward pass



- For layer $l = L - 1$, this is usually an all-to-all layer, the same as in the MLP
 - If least square error is used, calculate

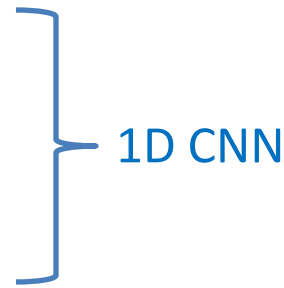
$$y^{(L)} = \text{sigmoid} \left(W^{(L-1)} y^{(L-1)} + b^{(L-1)} \right) \quad y^{(L)} \in R^K$$
 - If softmax is used, calculate

$$y^{(L)} = \text{softmax} \left(W^{(L-1)} y^{(L-1)} + b^{(L-1)} \right) \quad y^{(L)} \in R^K$$
- Do prediction with $y^{(L)}$

Relationship to MLP?

- Commonality?
 - Essentially MLPs
- Difference?
 - Pooling is different
 - Many small MLPs with shared weights
 - Dynamic connections and dynamic size

Outline

- Forward pass
 - **Backward pass**
 - Feature combination
 - 2D CNN
- 
- 1D CNN

Error functions are the same as in MLP

- Error function $E = \sum_{n=1}^N E^{(n)}$

where $E^{(n)}$ is the error function for each input sample n

- Least square error

$$E^{(n)} = \frac{1}{2} \sum_{k=1}^K (t_k - y_k^{(L)})^2, \quad y_k^{(L)} = \frac{1}{1 + \exp(-w_k^{(L-1)\top} y^{(L-1)} - b_k^{(L-1)})}$$

Where t is target of the form $(0, 0, \dots, 1, 0, 0)^T$

- Cross-entropy error

$$E^{(n)} = - \sum_{k=1}^K t_k \ln y_k^{(L)}, \quad y_k^{(L)} = \frac{\exp(w_k^{(L-1)\top} y^{(L-1)} + b_k^{(L-1)})}{\sum_{j=1}^K \exp(w_j^{(L-1)\top} y^{(L-1)} + b_j^{(L-1)})}$$

In what follows, except $E^{(n)}$, for clarity, we will omit the superscript (n) on x, t, u, y, δ etc. for each input sample.

n is the sample index.

Weight adjustment are the same as in MLP

- Weight adjustment

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial E}{\partial w_{ji}^{(l)}} \quad b_j^{(l)} = b_j^{(l)} - \alpha \frac{\partial E}{\partial b_j^{(l)}}$$

Learning rate
↙

where w_{ji} denotes the connection weight from node j to node i and b_j denotes the bias on node j (on any feature map i)

- Weight decay is often used on w_{ji} (not necessary on b_j) which amounts to adding an additional term on the cost function

$$J = E + \frac{\lambda}{2} \sum_{i,j,l} (w_{ji}^{(l)})^2$$

- Weight adjustment on w_{ji} is changed to

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial J}{\partial w_{ji}^{(l)}} = w_{ji}^{(l)} - \alpha \frac{\partial E}{\partial w_{ji}^{(l)}} - \alpha \lambda w_{ji}^{(l)}$$

Main idea

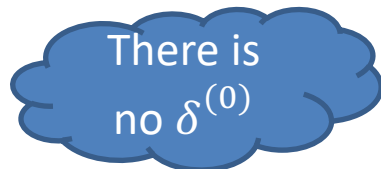
- What we only need to know is $\frac{\partial E}{\partial w_{ji}^{(l)}}$ and $\frac{\partial E}{\partial b_j^{(l)}}$

- Since

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \sum_n \frac{\partial E^{(n)}}{\partial w_{ji}^{(l)}}, \quad \frac{\partial E}{\partial b_j^{(l)}} = \sum_n \frac{\partial E^{(n)}}{\partial b_j^{(l)}}$$

we only need to know $\partial E^{(n)} / \partial w_{ji}^{(l)}$ and $\partial E^{(n)} / \partial b_j^{(l)}$

- If we know $\partial E^{(n)} / \partial u_j^{(l)}$, where $u_j^{(l)}$ denotes the total input to the j -th neuron in the l -th layer, things will be easy
 - $\delta_j^{(l)} \equiv \partial E^{(n)} / \partial u_j^{(l)}$ is called **local gradient** for each sample, same as in MLP



Chain Rule and Composition Rule

Suppose we have:

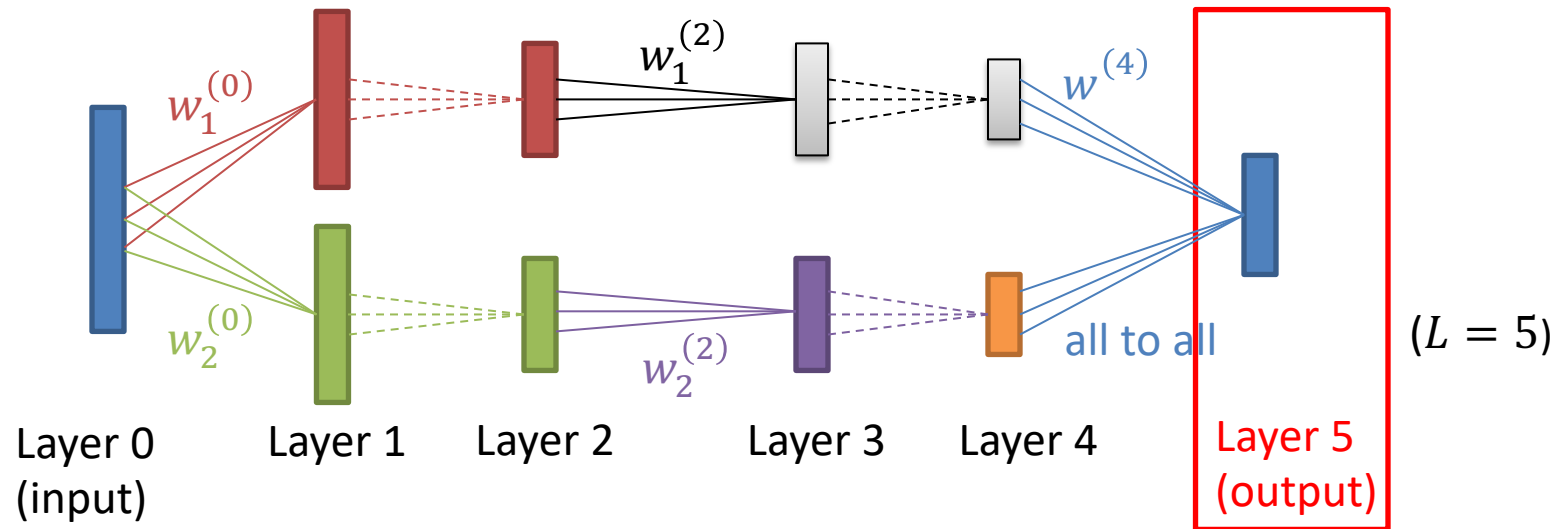
- Independent input variables x_1, x_2, \dots, x_n
- Dependent **intermediate variables**, u_1, u_2, \dots, u_m , each of which is a function of x_1, x_2, \dots, x_n
- Dependent output variables y_1, y_2, \dots, y_p , each of which is a function of u_1, u_2, \dots, u_m

Then for any $i \in \{1, 2, \dots, p\}$ and $j \in \{1, 2, \dots, n\}$ we have

$$\frac{\partial y_i}{\partial x_j} = \sum_{k=1}^m \frac{\partial y_i}{\partial u_k} \frac{\partial u_k}{\partial x_j}$$

Sum over the intermediate variables

Output layer

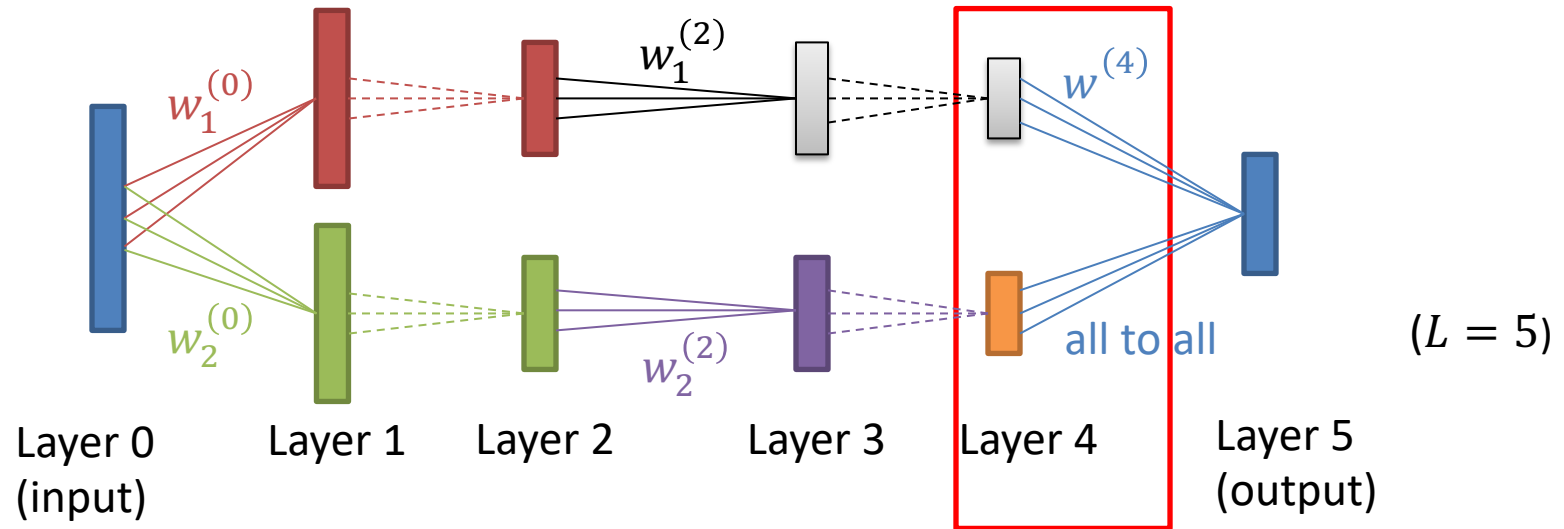


- This is the L -th layer (note there is no $w_k^{(L)}$ or $b_k^{(L)}$)
- The local gradient for each sample ← The same as in MLP!
 - If the **least square error** is used

$$\delta^{(L)} = (y - t) \bullet f'(u^{(L)})$$
 where y is the output of sigmoid functions
 - If the **cross-entropy error** is used

$$\delta^{(L)} = (y - t)$$
 where y is the output of softmax functions

Classification layer (all-to-all)



- This is the $(L - 1)$ -th layer
- Gradient for each sample ← The same as in MLP!

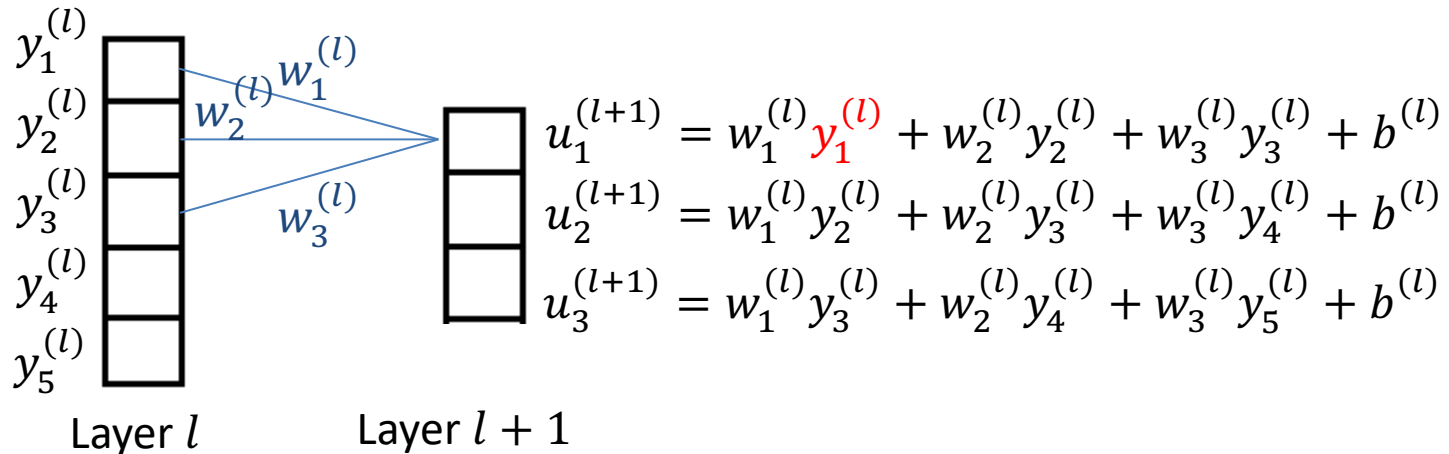
$$\frac{\partial E^{(n)}}{\partial w^{(L-1)}} = \delta^{(L)} (f(u^{(L-1)}))^{\top}, \quad \frac{\partial E^{(n)}}{\partial b^{(L-1)}} = \delta^{(L)}$$

- The local gradient for each sample ← The same as in MLP!

$$\delta^{(L-1)} = (W^{(L-1)})^{\top} \delta^{(L)} \bullet f'(u^{(L-1)})$$

Convolutional layer

If layer l is a convolutional layer, consider one single feature map



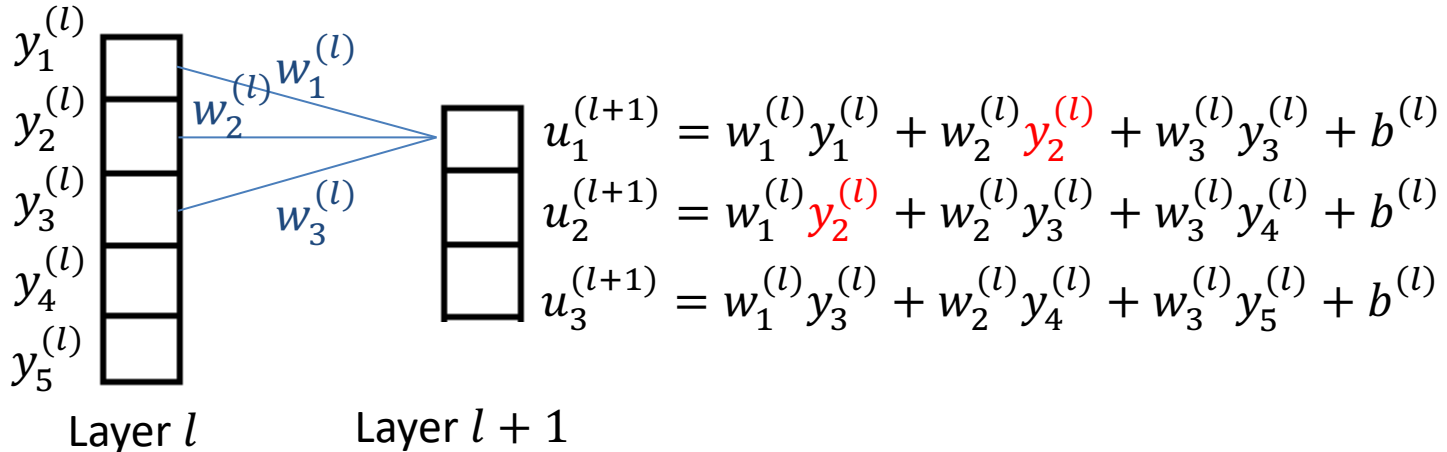
- $y_1^{(l)}$ appears once in $u^{(l+1)}$, and thus in the error function

$$\delta_1^{(l)} = \frac{\partial E^{(n)}}{\partial u_1^{(l)}} = \frac{\partial E^{(n)}}{\partial u_1^{(l+1)}} \frac{\partial u_1^{(l+1)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial u_1^{(l)}} = \delta_1^{(l+1)} w_1^{(l)} f'(u_1^{(l)})$$

Note the subscripts in this slides do not index feature maps, but elements in a feature map.

Convolutional layer

If layer l is a convolutional layer, consider one single feature map



- $y_2^{(l)}$ appears twice in $u^{(l+1)}$, and thus in the error function


$$\begin{aligned} \delta_2^{(l)} &= \frac{\partial E^{(n)}}{\partial u_2^{(l)}} = \frac{\partial E^{(n)}}{\partial u_1^{(l+1)}} \frac{\partial u_1^{(l+1)}}{\partial y_2^{(l)}} \frac{\partial y_2^{(l)}}{\partial u_2^{(l)}} + \frac{\partial E^{(n)}}{\partial u_2^{(l+1)}} \frac{\partial u_2^{(l+1)}}{\partial y_2^{(l)}} \frac{\partial y_2^{(l)}}{\partial u_2^{(l)}} \\ &= \delta_1^{(l+1)} w_2^{(l)} f'(u_2^{(l)}) + \delta_2^{(l+1)} w_1^{(l)} f'(u_2^{(l)}) \end{aligned}$$

- Similarly we can obtain $\delta_3^{(l)}, \delta_4^{(l)}$ and $\delta_5^{(l)}$

Convolutional layer

- Local gradient in the vector form

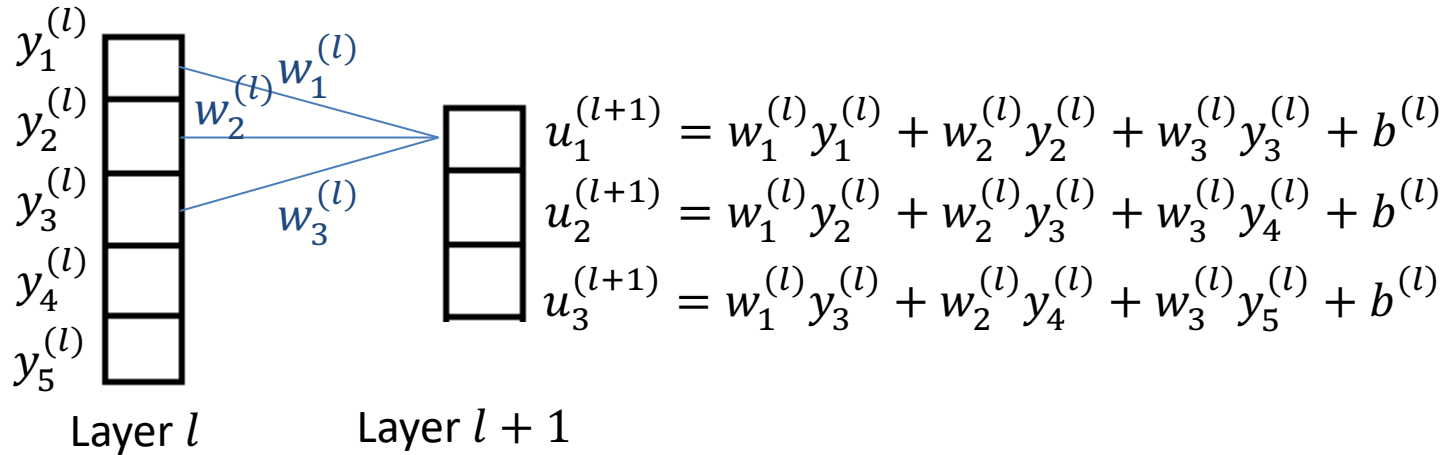
$$\frac{\partial E^{(n)}}{\partial u^l} = \underbrace{\begin{pmatrix} \delta_1^{(l+1)} w_1^{(l)} \\ \delta_1^{(l+1)} w_2^{(l)} + \delta_2^{(l+1)} w_1^{(l)} \\ \delta_1^{(l+1)} w_3^{(l)} + \delta_2^{(l+1)} w_2^{(l)} + \delta_3^{(l+1)} w_1^{(l)} \\ \delta_2^{(l+1)} w_3^{(l)} + \delta_3^{(l+1)} w_2^{(l)} \\ \delta_3^{(l+1)} w_3^{(l)} \end{pmatrix}}_{\text{Full convolution of } \delta^{(l+1)} \text{ and } w^{(l)}} \bullet \begin{pmatrix} f'(u_1^{(l)}) \\ f'(u_2^{(l)}) \\ f'(u_3^{(l)}) \\ f'(u_4^{(l)}) \\ f'(u_5^{(l)}) \end{pmatrix}$$


 Elementwise multiplication

- Therefore

$$\delta^l = \left(\delta^{(l+1)} *_{\text{full}} w^{(l)} \right) \bullet \left(f'(u^{(l)}) \right)$$

Convolutional layer



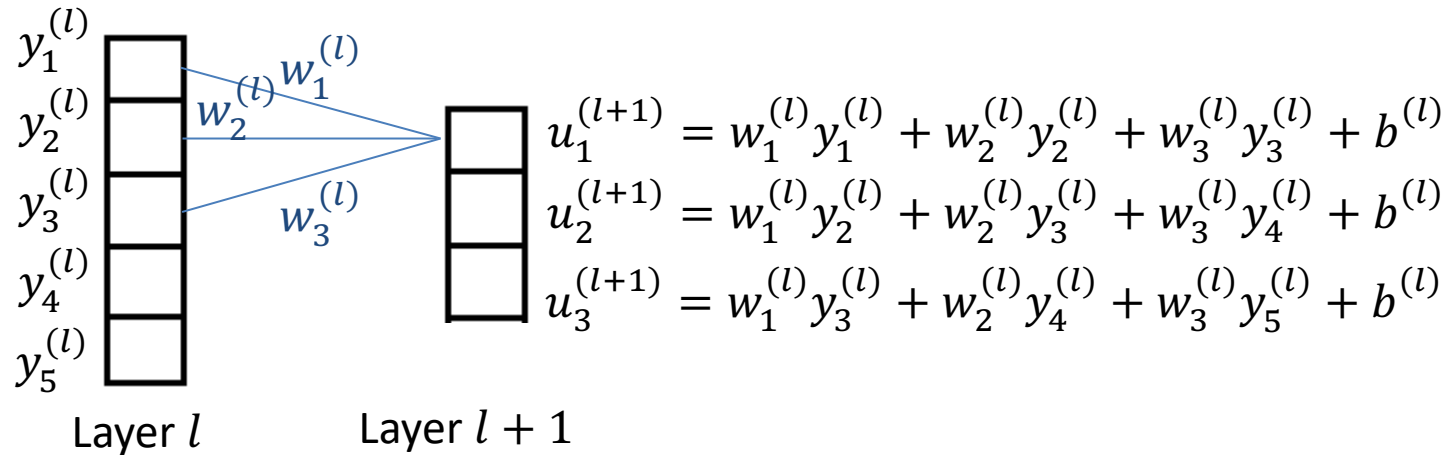
- Gradient of $w^{(l)}$: scalar form

$$\frac{\partial E^{(n)}}{\partial w_1^{(l)}} = \sum_{i=1}^3 \frac{\partial E^{(n)}}{\partial u_i^{(l+1)}} \frac{\partial u_i^{(l+1)}}{\partial w_1^{(l)}} = \delta_1^{(l+1)} y_1^{(l)} + \delta_2^{(l+1)} y_2^{(l)} + \delta_3^{(l+1)} y_3^{(l)}$$

$$\frac{\partial E^{(n)}}{\partial w_2^{(l)}} = \sum_{i=1}^3 \frac{\partial E^{(n)}}{\partial u_i^{(l+1)}} \frac{\partial u_i^{(l+1)}}{\partial w_2^{(l)}} = \delta_1^{(l+1)} y_2^{(l)} + \delta_2^{(l+1)} y_3^{(l)} + \delta_3^{(l+1)} y_4^{(l)}$$

$$\frac{\partial E^{(n)}}{\partial w_3^{(l)}} = \sum_{i=1}^3 \frac{\partial E^{(n)}}{\partial u_i^{(l+1)}} \frac{\partial u_i^{(l+1)}}{\partial w_3^{(l)}} = \delta_1^{(l+1)} y_3^{(l)} + \delta_2^{(l+1)} y_4^{(l)} + \delta_3^{(l+1)} y_5^{(l)}$$

Convolutional layer



- Gradient of $w^{(l)}$: vector form

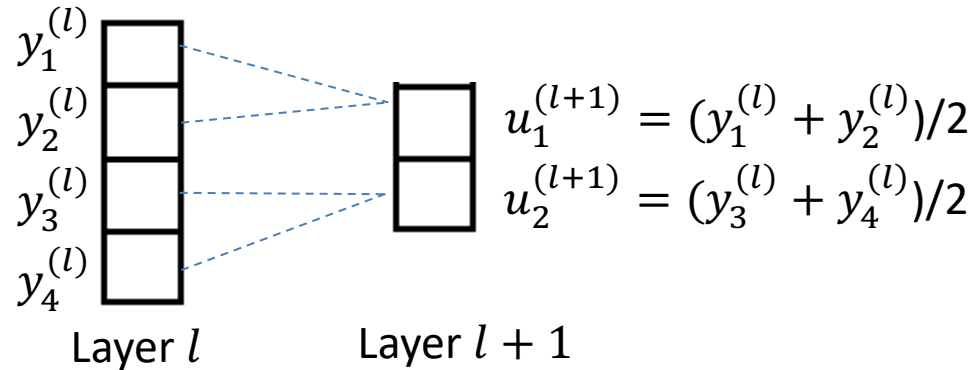
$$\frac{\partial E^{(n)}}{\partial w^{(l)}} = y^{(l)} *_{\text{valid}} \text{rot180}(\delta^{(l+1)})$$

- Gradient of $b^{(l)}$

$$\frac{\partial E^{(n)}}{\partial b^{(l)}} = \sum_{i=1}^3 \frac{\partial E^{(n)}}{\partial u_i^{(l+1)}} \frac{\partial u_i^{(l+1)}}{\partial b^{(l)}} = \sum_i \delta_i^{(l+1)}$$

Average pooling layer

If layer l is an average pooling layer, consider one single feature map



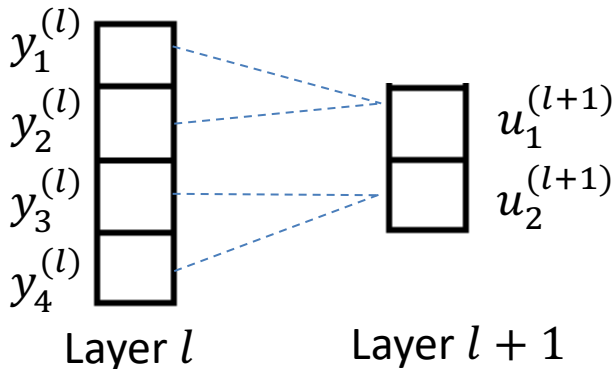
- Local gradient in the scalar form

$$\delta_1^{(l)} = \frac{\partial E^{(n)}}{\partial u_1^{(l)}} = \frac{\partial E^{(n)}}{\partial u_1^{(l+1)}} \frac{\partial u_1^{(l+1)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial u_1^{(l)}} = \delta_1^{(l+1)} \frac{1}{2} f'(u_1^{(l)})$$

$$\delta_2^{(l)} = \frac{\partial E^{(n)}}{\partial u_2^{(l)}} = \frac{\partial E^{(n)}}{\partial u_1^{(l+1)}} \frac{\partial u_1^{(l+1)}}{\partial y_2^{(l)}} \frac{\partial y_2^{(l)}}{\partial u_2^{(l)}} = \delta_1^{(l+1)} \frac{1}{2} f'(u_2^{(l)})$$

Similarly we can obtain $\delta_3^{(l)} = \delta_2^{(l+1)} \frac{1}{2} f'(u_3^{(l)})$, $\delta_4^{(l)} = \delta_2^{(l+1)} \frac{1}{2} f'(u_4^{(l)})$

Average pooling layer



$$\begin{aligned}\delta_1^{(l)} &= \delta_1^{(l+1)} \frac{1}{2} f'(u_1^{(l)}) \\ \delta_2^{(l)} &= \delta_1^{(l+1)} \frac{1}{2} f'(u_2^{(l)}) \\ \delta_3^{(l)} &= \delta_2^{(l+1)} \frac{1}{2} f'(u_3^{(l)}) \\ \delta_4^{(l)} &= \delta_2^{(l+1)} \frac{1}{2} f'(u_4^{(l)})\end{aligned}$$

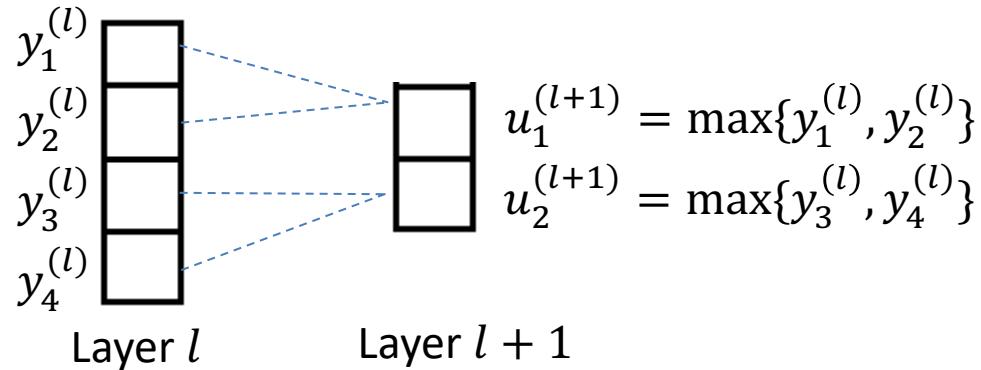
- Local gradient in the vector form

$$\delta^{(l)} = \frac{1}{\text{poolingsize}} \text{upsample}(\delta^{(l+1)}) \bullet f'(u^{(l)})$$

$$\text{upsample}(a) \triangleq \begin{pmatrix} a_1 \\ a_1 \\ \vdots \\ a_n \\ a_n \end{pmatrix} \begin{matrix} \left. \vphantom{\begin{pmatrix} a_1 \\ a_1 \\ \vdots \\ a_n \\ a_n \end{pmatrix}} \right\} \text{Poolingsize} \\ \left. \vphantom{\begin{pmatrix} a_1 \\ a_1 \\ \vdots \\ a_n \\ a_n \end{pmatrix}} \right\} \text{Poolingsize} \end{matrix}$$

Max pooling layer

If layer l is a max pooling layer, consider one single feature map



- If $y_1^{(l)} \geq y_2^{(l)}$,

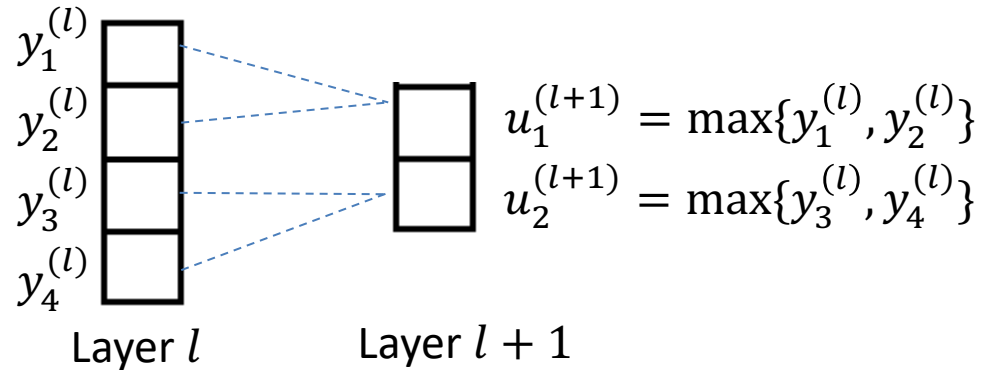
$$\frac{\partial E^{(n)}}{\partial u_1^{(l)}} = \frac{\partial E^{(n)}}{\partial u_1^{(l+1)}} \frac{\partial u_1^{(l+1)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial u_1^{(l)}} = \delta_1^{(l+1)} f'(u_1^{(l)}), \quad \frac{\partial E^{(n)}}{\partial u_2^{(l)}} = 0$$

- Else

$$\frac{\partial E^{(n)}}{\partial u_1^{(l)}} = 0, \quad \frac{\partial E^{(n)}}{\partial u_2^{(l)}} = \delta_1^{(l+1)} f'(u_2^{(l)})$$

Max pooling layer

If layer l is a max pooling layer



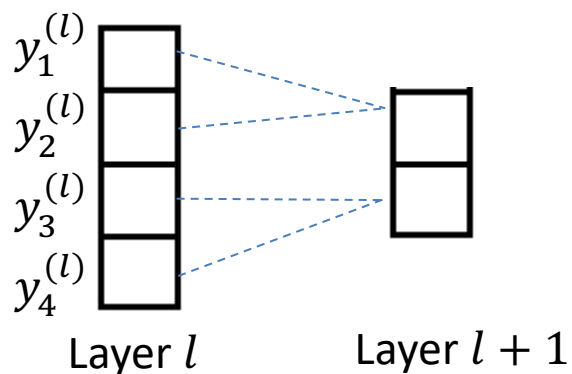
- If $y_3^{(l)} \geq y_4^{(l)}$,

$$\frac{\partial E^{(n)}}{\partial u_3^{(l)}} = \frac{\partial E^{(n)}}{\partial u_2^{(l+1)}} \frac{\partial u_2^{(l+1)}}{\partial y_3^{(l)}} \frac{\partial y_3^{(l)}}{\partial u_3^{(l)}} = \delta_2^{(l+1)} f'(u_3^{(l)}), \quad \frac{\partial E^{(n)}}{\partial u_4^{(l)}} = 0$$

- Else

$$\frac{\partial E^{(n)}}{\partial u_3^{(l)}} = 0, \quad \frac{\partial E^{(n)}}{\partial u_4^{(l)}} = \delta_2^{(l+1)} f'(u_4^{(l)})$$

Max pooling layer



Suppose $y_1^{(l)} \geq y_2^{(l)}$ and $y_3^{(l)} \geq y_4^{(l)}$

$$\begin{aligned} \frac{\partial E^{(n)}}{\partial u_1^{(l)}} &= \delta_1^{(l+1)} f'(u_1^{(l)}), & \frac{\partial E^{(n)}}{\partial u_3^{(l)}} &= \delta_2^{(l+1)} f'(u_3^{(l)}) \\ \frac{\partial E^{(n)}}{\partial u_2^{(l)}} &= 0, & \frac{\partial E^{(n)}}{\partial u_4^{(l)}} &= 0 \end{aligned}$$

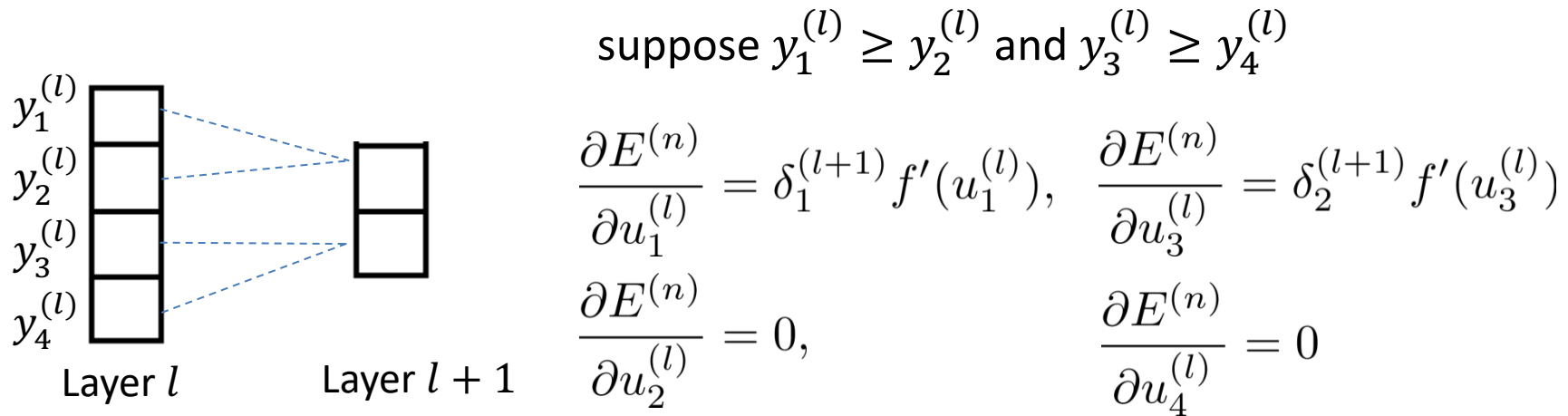
- Local gradient in the vector form (suppose $y_1^{(l)} \geq y_2^{(l)}$ and $y_3^{(l)} \geq y_4^{(l)}$)

$$\delta^{(l)} = \frac{\partial E^{(n)}}{\partial u^{(l)}} = \begin{pmatrix} \delta_1^{(l+1)} \\ 0 \\ \delta_2^{(l+1)} \\ 0 \end{pmatrix} \bullet f'(u^{(l)}) = \mathcal{M}(y^{(l)}) \bullet \text{upsample}(\delta^{(l+1)}) \bullet f'(u^{(l)}),$$

$$\mathcal{M}(y^{(l)}) = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Where 1 obtained at the maximal value of y

Max pooling layer



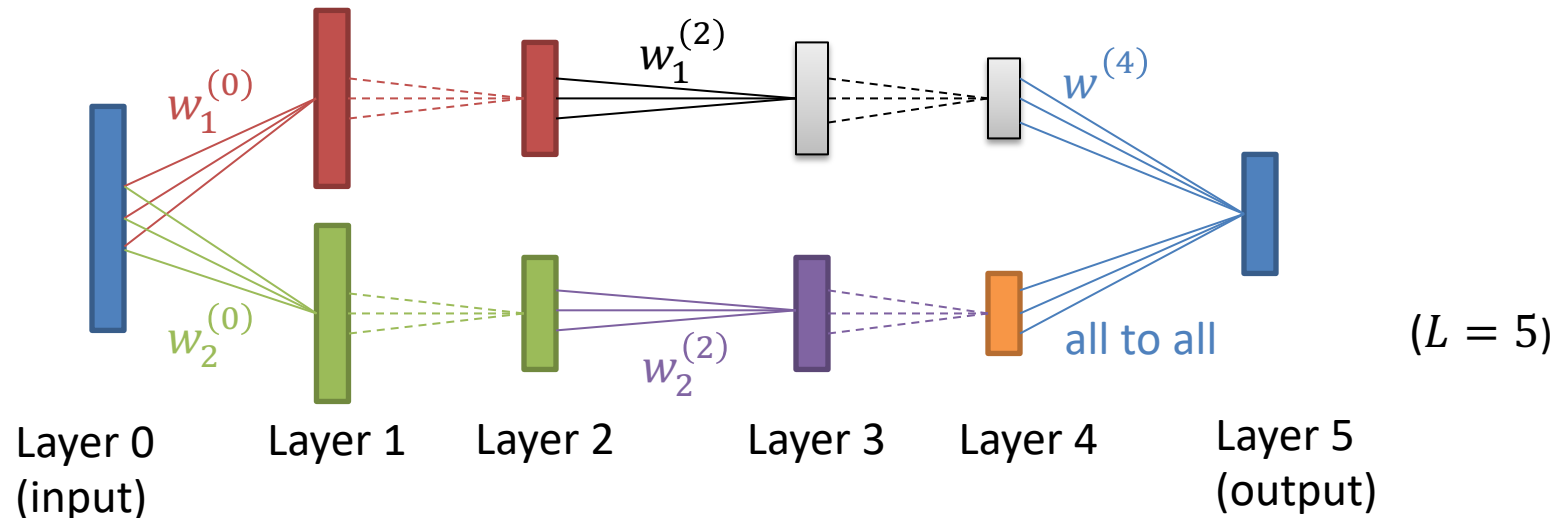
- Local gradient in the vector form (suppose $y_1^{(l)} \geq y_2^{(l)}$ and $y_3^{(l)} \geq y_4^{(l)}$)

In general $\mathcal{M}(a) \triangleq \begin{pmatrix} \frac{1\{block_1\}}{\vdots} \\ \frac{1\{block_n\}}{\vdots} \end{pmatrix}$

where $1\{block_k\}$ is a block with only one element equal to 1, whose index is given by the maximal value of each pooling block,

and other elements equal to 0

BP Algorithm summary



- Given a structure, calculate the local sensitivity $\delta^{(l)}$ in every layer from $l = L$ to $l = 1$ including
 - Output layer, classification layer, convolutional layers, pooling layers
- Then calculate the gradients w.r.t. parameters $w^{(l)}$ and $b^{(l)}$ in each of the following layers using $\delta^{(l)}$
 - Classification layer, convolutional layers

BP Algorithm summary

For $l = L, L - 1, \dots, 0$, do

- If $l = L$: $\delta^{(L)} = (y - t) \bullet f'(u^{(L)})$ or $\delta^{(L)} = (y - t)$
- If $l = L - 1$: $\delta^{(L-1)} = (W^{(L-1)})^\top \delta^{(L)} \bullet f'(u^{(L-1)})$

$$\frac{\partial E^{(n)}}{\partial w^{(L-1)}} = \delta^{(L)} (f(u^{(L-1)}))^\top, \quad \frac{\partial E^{(n)}}{\partial b^{(L-1)}} = \delta^{(L)}$$

- If $0 \leq l \leq L - 2$ is a convolutional layer:

$$\delta_p^l = \left(\delta_p^{(l+1)} *_{\text{full}} w_p^{(l)} \right) \bullet \left(f'(u_p^{(l)}) \right)$$

$$\frac{\partial E^{(n)}}{\partial w_p^{(l)}} = y_p^{(l)} *_{\text{valid}} \text{rot90}(\delta_p^{(l+1)}, 2), \quad \frac{\partial E^{(n)}}{\partial b_p^{(l)}} = \sum_i (\delta_p^{(l+1)})_i$$

- If $0 \leq l \leq L - 2$ is a pooling layer (avg. or max.):

$$\delta_p^{(l)} = \frac{1}{\text{pooling size}} \text{upsample}(\delta_p^{(l+1)}) \bullet f'(u_p^{(l)})$$

$$\text{or } \delta_p^{(l)} = \mathcal{M}(y_p^{(l)}) \bullet \text{upsample}(\delta_p^{(l+1)}) \bullet f'(u_p^{(l)})$$

p indexes feature map

BP Algorithm summary

- Do weight adjustment

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial E}{\partial w_{ji}^{(l)}}, \quad b_j^{(l)} = b_j^{(l)} - \alpha \frac{\partial E}{\partial b_j^{(l)}}$$

where $w_{ji}^{(l)}$ denotes the connection weight from node i to node j and $b_j^{(l)}$ denotes the bias on node j (in any feature map)

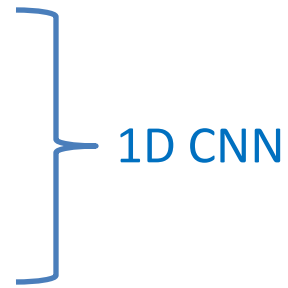
- Note
 - The overall gradient

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \sum_n \frac{\partial E^{(n)}}{\partial w_{ji}^{(l)}}, \quad \frac{\partial E}{\partial b_j^{(l)}} = \sum_n \frac{\partial E^{(n)}}{\partial b_j^{(l)}}$$

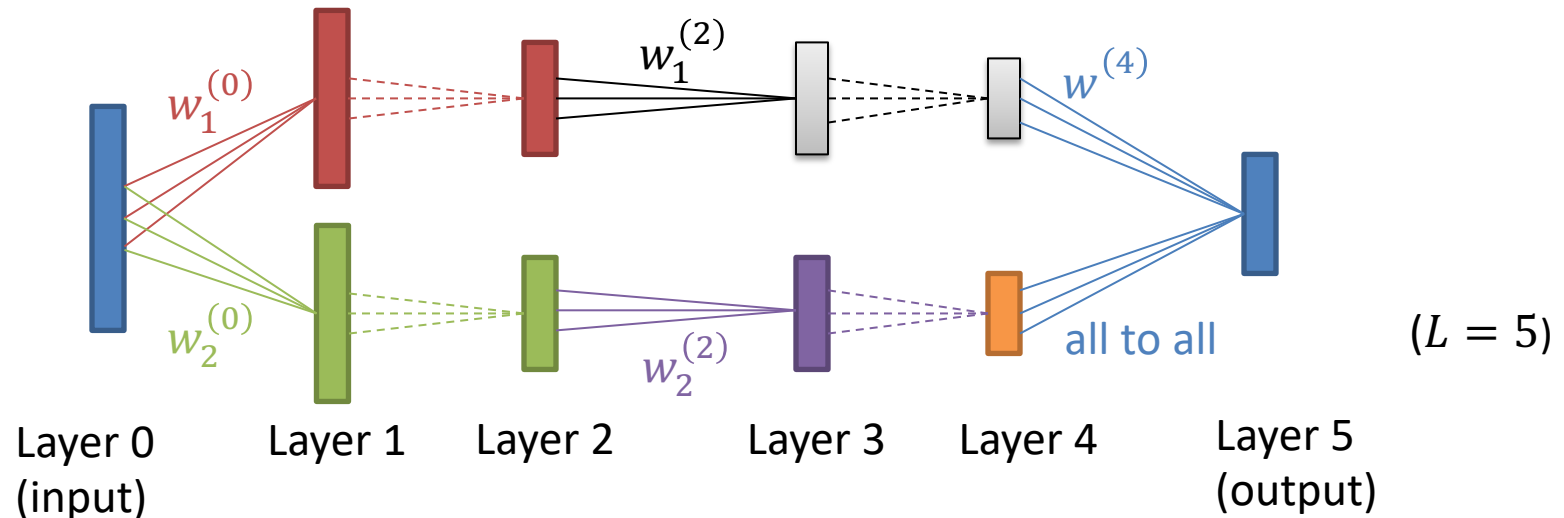
- Weight decay is often used

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial E}{\partial w_{ji}^{(l)}} - \eta w_{ji}^{(l)}$$

Outline

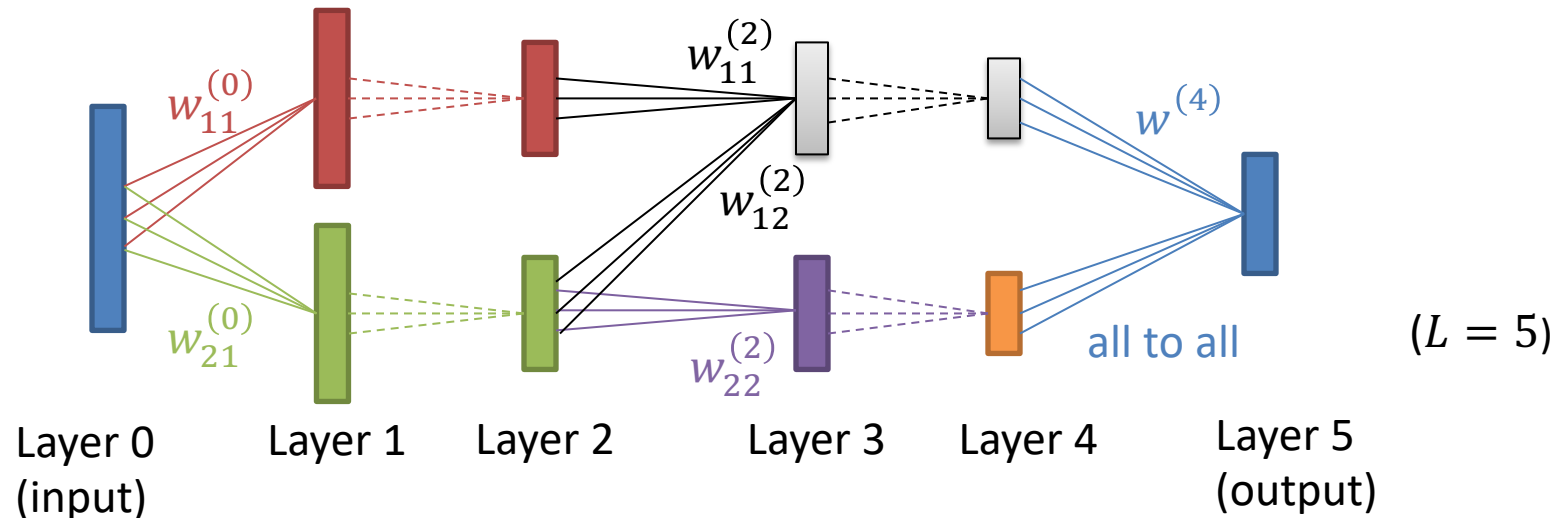
- Forward pass
 - Backward pass
 - Feature combination
 - 2D CNN
- 
- 1D CNN

Combination of multiple feature maps



- In this example, the classification layer (all-to-all) is a feature combination layer
- Feature combination can be also performed in convolutional layers
 - The convolutional layers often have multiple feature maps
 - The input layer may have three maps (RGB)
- Feature combination can increase the complexity of features

An example

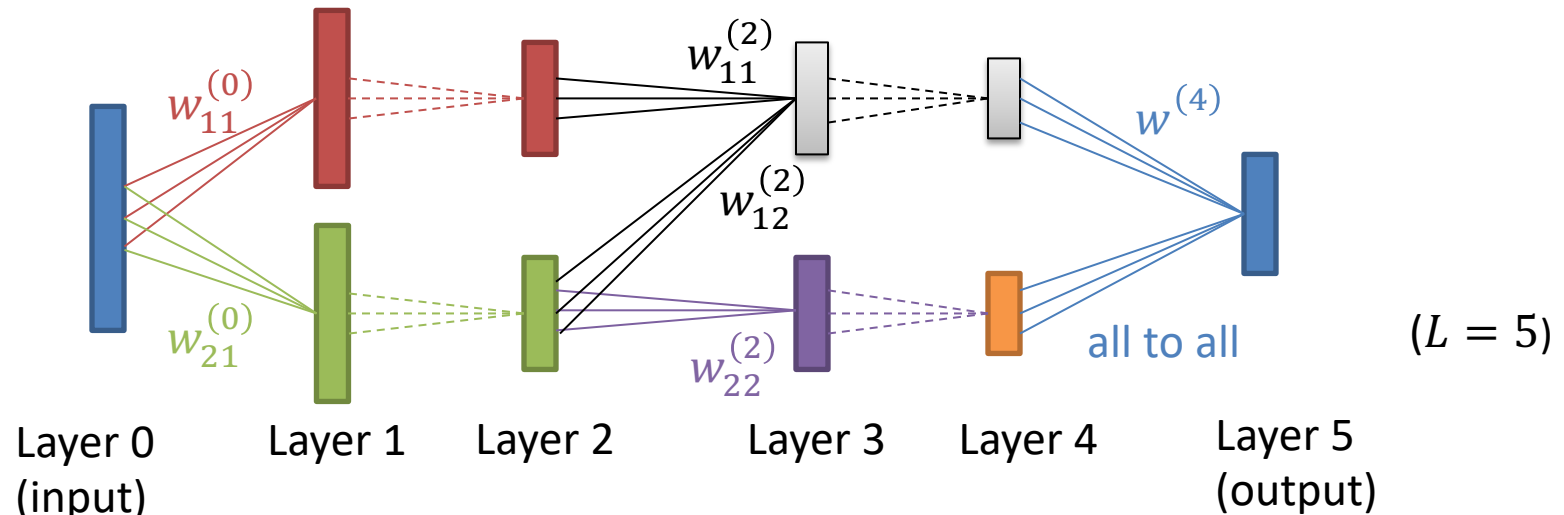


- Let $w_{qp}^{(l)}$ denote the filter connecting the p -th feature map in layer l to the q -th feature map in layer $l + 1$ (different from MLP)
- The first feature map in layer 3 combines the output of two feature maps in layer 2

$$y_1^{(3)} = f \left(y_1^{(2)} *_{valid} \text{rot180} \left(w_{11}^{(2)} \right) + y_2^{(2)} *_{valid} \text{rot180} \left(w_{12}^{(2)} \right) + b_1^{(2)} \right)$$

where f is the activation function

Forward pass



One bias vector
per map

- For layer $l = 0: 1: L - 2$, do
 - If the l -th layer is a convolution layer, we obtain a new feature map

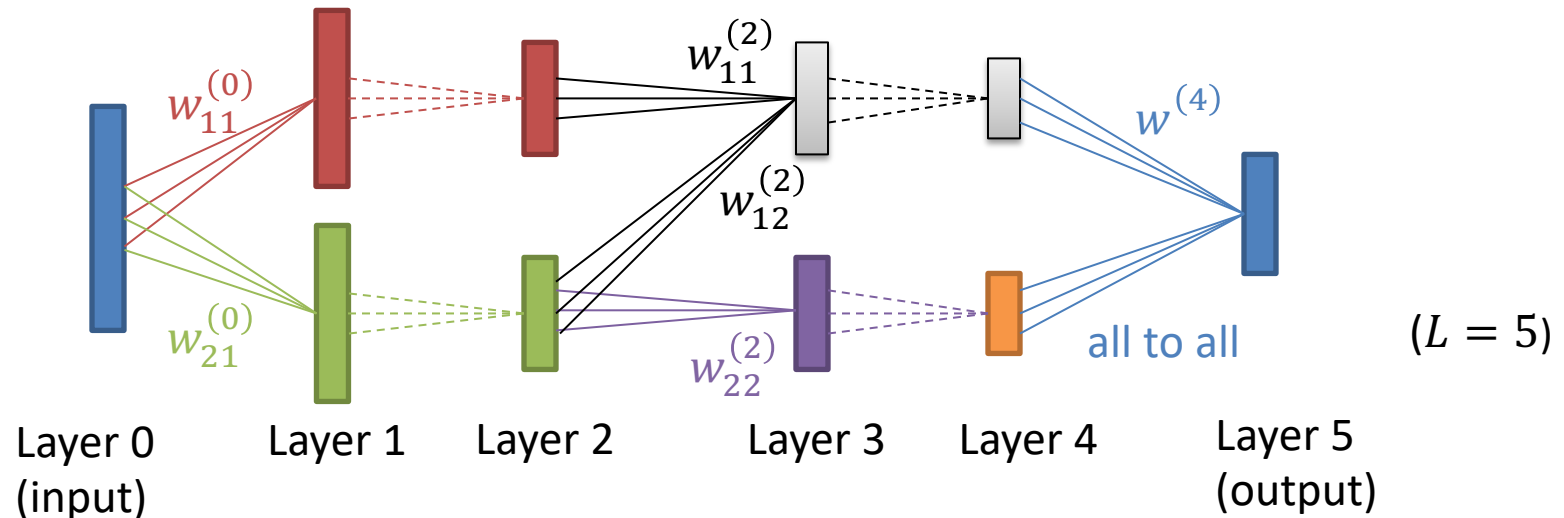
$$y_q^{(l+1)} = f \left(\sum_{p \in M_q} y_p^{(l)} *_{valid} \text{rot180} \left(w_{qp}^{(l)} \right) + b_q^{(l)} \right)$$

where f is the activation function, M_q denotes a selection of *input maps* in layer l

To calculate the output of layer 1, $M_1 = ?$ $M_2 = ?$
 To calculate the output of layer 3, $M_1 = ?$ $M_2 = ?$

M_q often contains
all input maps

Forward pass



- For layer $l = 0: 1: L - 2$, do

- If the l -th layer is a pooling layer

$$y_p^{(l+1)} = \text{pooling}(y_p^{(l)})$$

- For layer $l = L - 1$

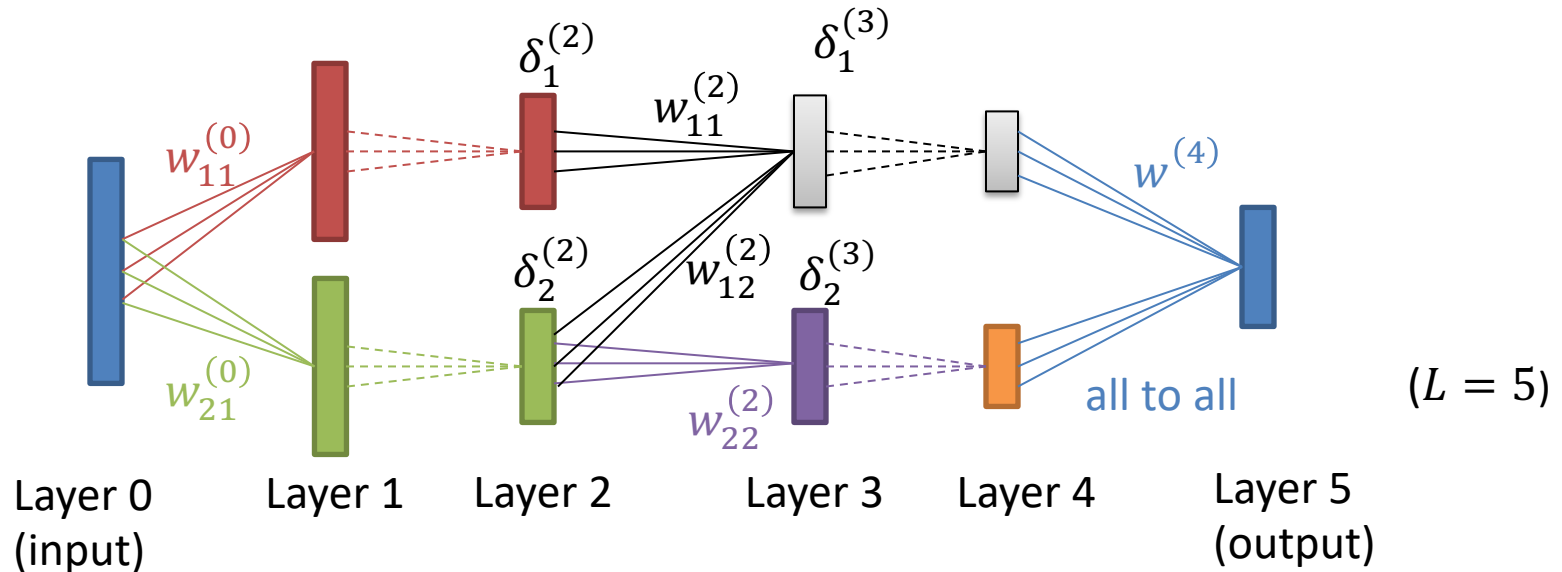
$$y^{(L)} = \text{sigmoid}(W^{(L-1)}y^{(L-1)} + b^{(L-1)})$$

$$y^{(L)} = \text{softmax}(W^{(L-1)}y^{(L-1)} + b^{(L-1)})$$

- Do prediction with $y^{(L)}$

The same as before

Backward pass: an example



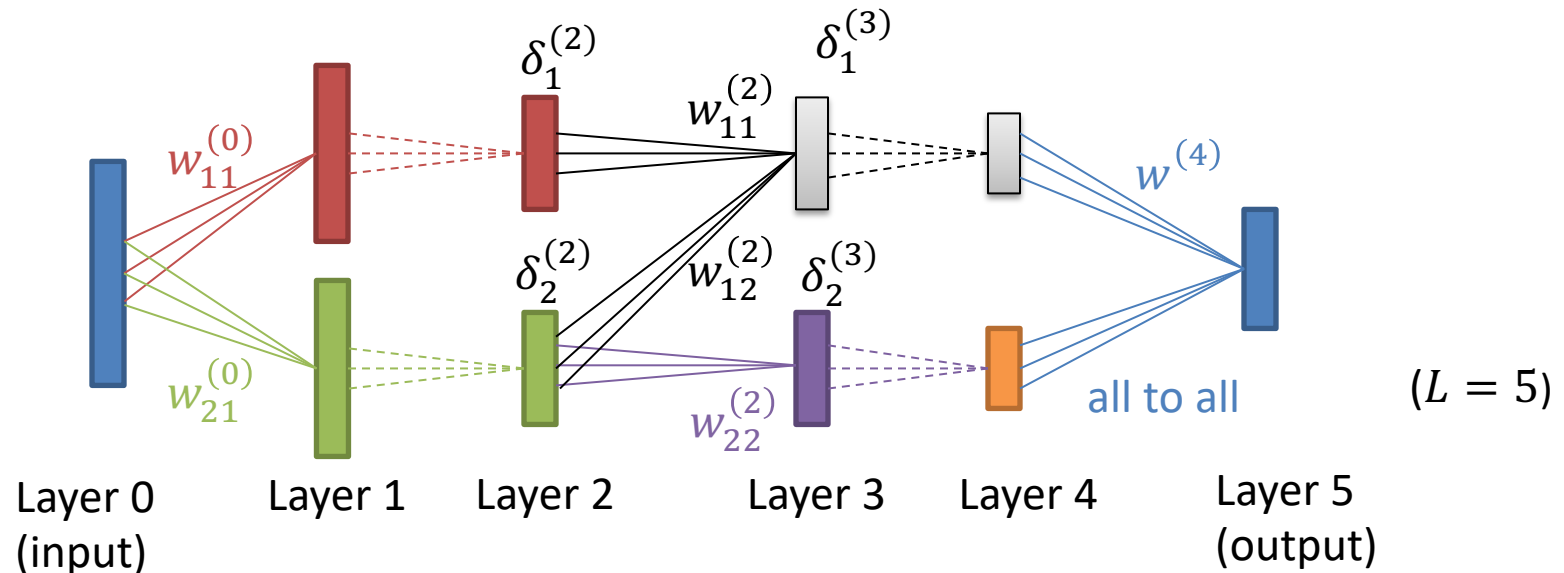
- The local gradient $\delta_1^{(3)}$ and $\delta_2^{(3)}$ in layer 3 do not change (they are determined by the subsequent pooling layer)
- The following gradients in layer 2 do not change

$$\frac{\partial E^{(n)}}{\partial w_{11}^{(2)}} = y_1^{(2)} *_{\text{valid}} \text{rot180}(\delta_1^{(3)}), \quad \frac{\partial E^{(n)}}{\partial b_1^{(2)}} = \sum_i (\delta_1^{(3)})_i,$$

$$\frac{\partial E^{(n)}}{\partial w_{22}^{(2)}} = y_2^{(2)} *_{\text{valid}} \text{rot180}(\delta_2^{(3)}), \quad \frac{\partial E^{(n)}}{\partial b_2^{(2)}} = \sum_i (\delta_2^{(3)})_i.$$



Backward pass: an example



- Similarly, the gradient $\partial E^{(n)} / \partial w_{12}^{(2)}$ is determined by $\delta_1^{(3)}$

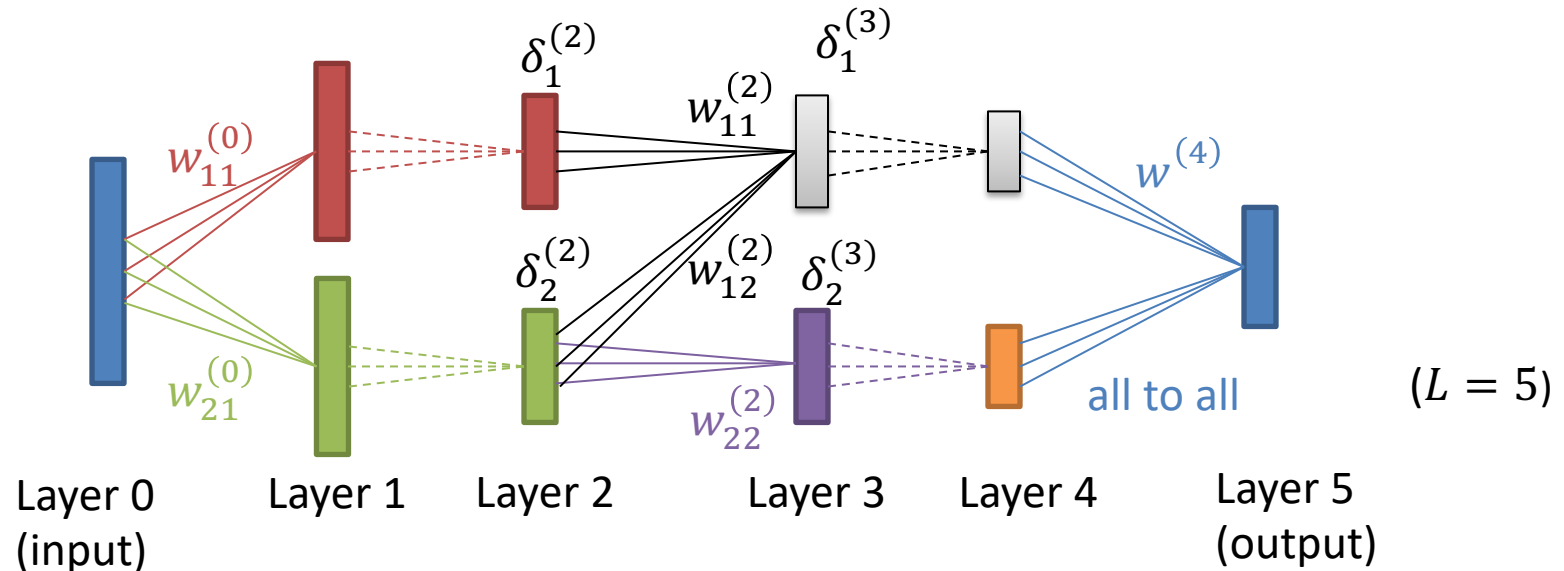
$$\frac{\partial E^{(n)}}{\partial w_{12}^{(2)}} = y_2^{(2)} *_{\text{valid}} \text{rot180}(\delta_1^{(3)})$$

Note $b_1^{(2)}$ has been determined in the previous slide, which is shared by $w_{11}^{(2)}$ and $w_{12}^{(2)}$

- In summary

$$\frac{\partial E^{(n)}}{\partial w_{qp}^{(2)}} = y_p^{(2)} *_{\text{valid}} \text{rot180}(\delta_q^{(3)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(2)}} = \sum_i (\delta_q^{(3)})_i$$

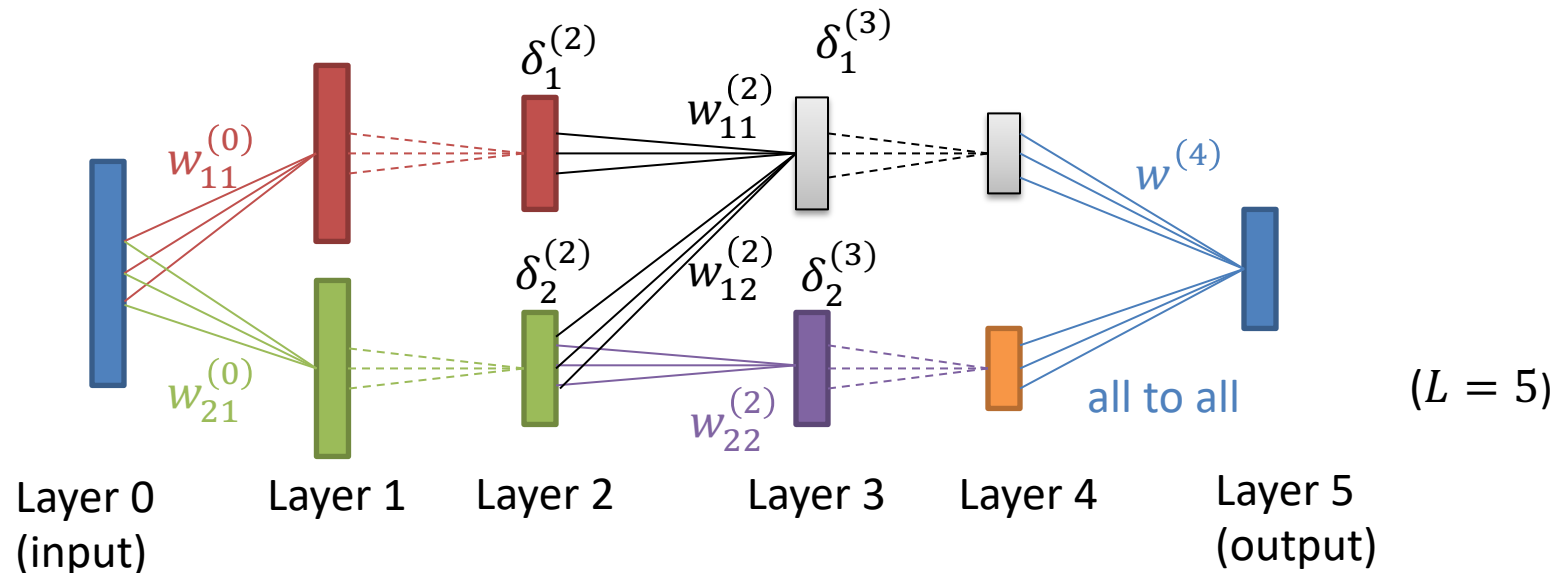
Backward pass: an example



- **Local gradient $\delta_1^{(2)}$** does not change ?
 - the input of the first feature map in layer 2 affects the output of the network (thus the error) only through $w_{11}^{(2)}$
- **Local gradient $\delta_2^{(2)}$** is different from before ?
 - the input of the second feature map in layer 2 affects the output of the network (thus the error) through both $w_{12}^{(2)}$ and $w_{22}^{(2)}$

$$\delta_2^{(2)} = \sum_{q=1}^2 \left(\delta_q^{(3)} *_{\text{full}} w_{q2}^{(2)} \right) \bullet \left(f'(u_2^{(2)}) \right)$$

Backward pass in general



- If $0 \leq l \leq L - 2$ is a convolutional layer
 - The local gradient ($l \neq 0$)

$$\delta_p^{(l)} = \sum_{q \in \tilde{M}_p} \left(\delta_q^{(l+1)} *_{\text{full}} w_{qp}^{(l)} \right) \bullet \left(f'(u_p^{(l)}) \right)$$

Note
difference
from M_q

where \tilde{M}_p denotes a selection of *output maps* in layer $l + 1$

- The gradients are determined by $\delta_q^{(l+1)}$

$$\frac{\partial E^{(n)}}{\partial w_{qp}^{(l)}} = y_p^{(l)} *_{\text{valid}} \text{rot180}(\delta_q^{(l+1)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\delta_q^{(l+1)})_i$$

BP Algorithm

For $l = L, L - 1, \dots, 0$, do

- If $l = L$: $\delta^{(L)} = (y - t) \bullet f'(u^{(L)})$ or $\delta^{(L)} = (y - t)$
- If $l = L - 1$: $\delta^{(L-1)} = (W^{(L-1)})^\top \delta^{(L)} \bullet f'(u^{(L-1)})$
 $\frac{\partial E^{(n)}}{\partial w^{(L-1)}} = \delta^{(L)} (f(u^{(L-1)}))^\top, \quad \frac{\partial E^{(n)}}{\partial b^{(L-1)}} = \delta^{(L)}$

- If $0 \leq l \leq L - 2$ is a convolutional layer:

$$\delta_p^{(l)} = \sum_{q \in \tilde{M}_p} \left(\delta_q^{(l+1)} *_{\text{full}} w_{qp}^{(l)} \right) \bullet \left(f'(u_p^{(l)}) \right), \forall l \neq 0$$

$$\frac{\partial E^{(n)}}{\partial w_{qp}^{(l)}} = y_p^{(l)} *_{\text{valid}} \text{rot180}(\delta_q^{(l+1)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\delta_q^{(l+1)})_i$$

- If $1 \leq l \leq L - 2$ is a pooling layer:

$$\delta_p^{(l)} = \frac{1}{\text{poolingsize}} \text{upsample}(\delta_p^{(l+1)}) \bullet f'(u_p^{(l)})$$

or $\delta_p^{(l)} = \mathcal{M}(y^{(l)}) \bullet \text{upsample}(\delta_p^{(l+1)}) \bullet f'(u_p^{(l)})$

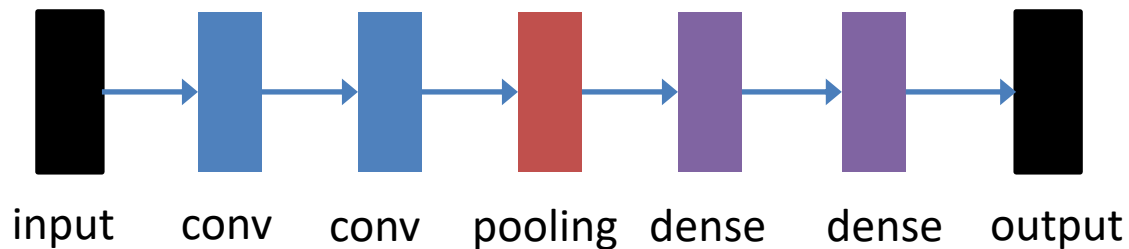
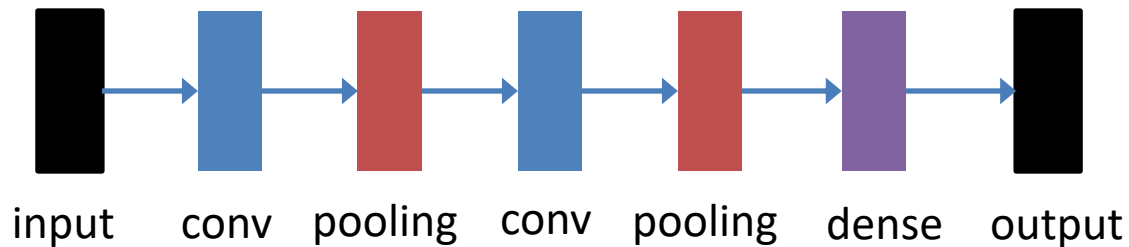
p and q index
feature maps

Implementation

- Run forward process
 - Calculate $f(u^l)$ and $f'(u^l)$ for $l = 1, 2, \dots, L$
- Run backward process
 - Calculate $\delta^{(l)}$ and $\partial E / \partial W^{(l-1)}, \partial E / \partial b^{(l-1)}$ for $l = L, L - 1, \dots, 1$
- Update $W^{(l)}$ and $b^{(l)}$ for $l = 0, 1, \dots, L - 1$
- Modular programming ← Basic idea of Caffe
 - Implement the layer as a class and provide functions for forward calculation and backward calculation, respectively
 - The forward functions and backward functions differ according to the type of the layer, e.g., input layer, convolutional layer, pooling layer, softmax output layer, sigmoid output layer, etc.
 - Then you can design different structures of CNN by specifying the layer modules in a main file

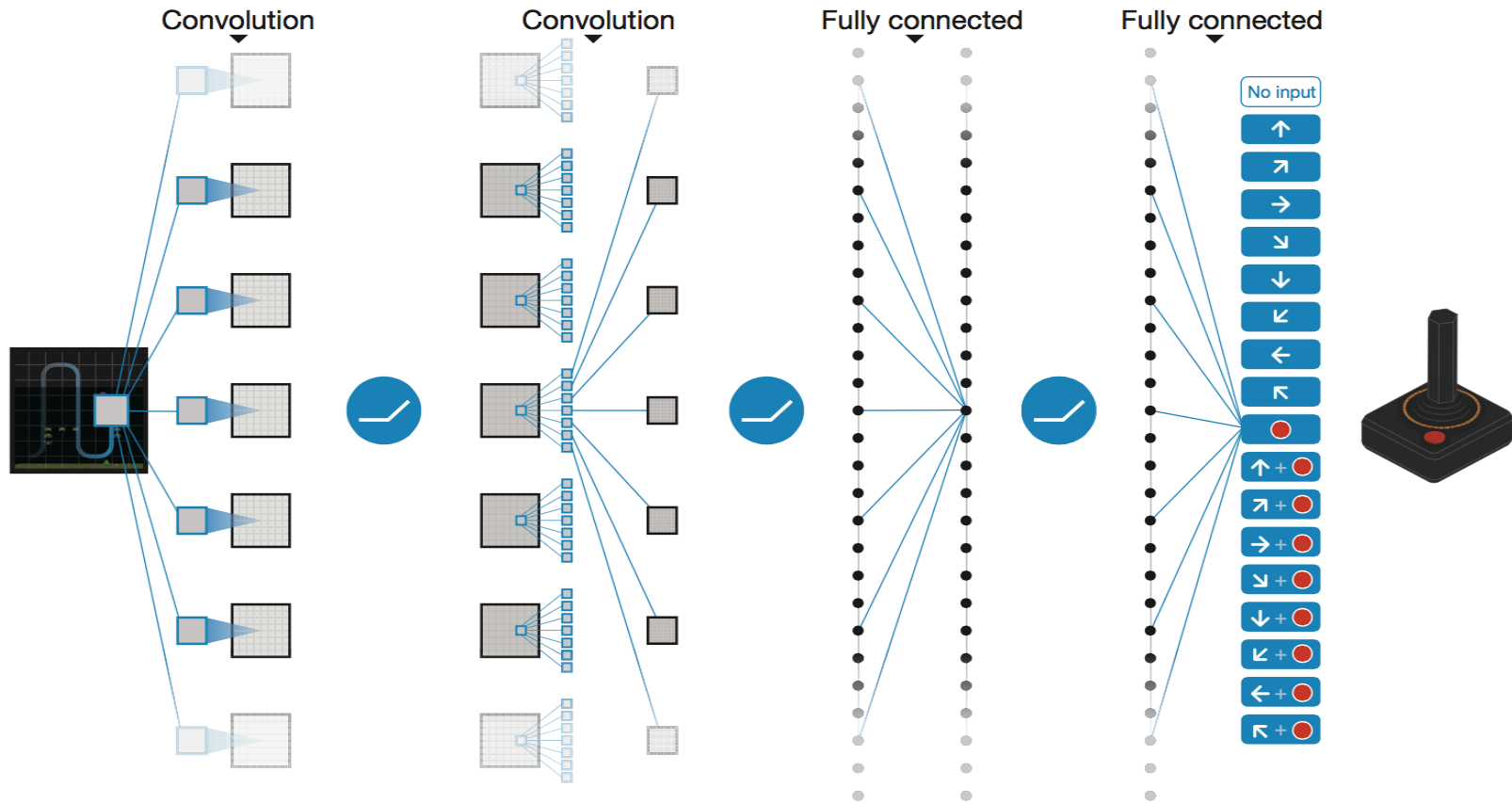
Implementation

- The modules can be stacked in various structures

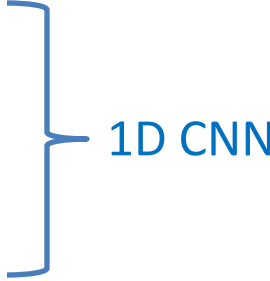


Dense: all-to-all connections

Possible without Pooling, Why?



Outline

- Forward pass
 - Backward pass
 - Feature combination
 - 2D CNN
- 
- 1D CNN

2D CNN

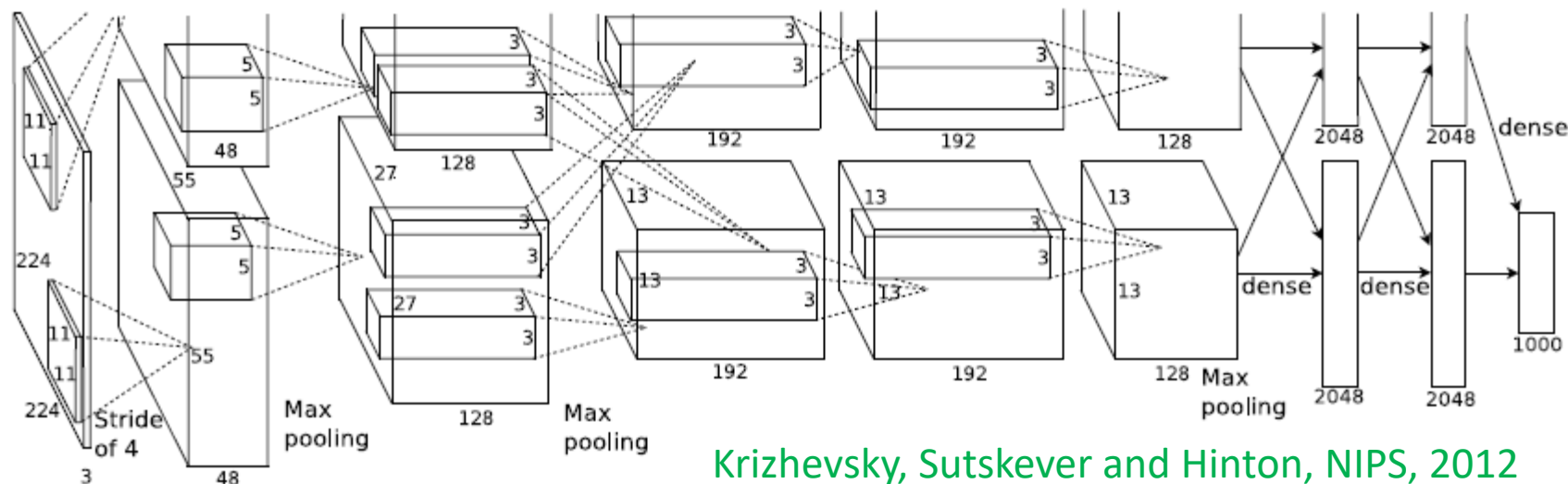
- The forward pass and backward pass are the same as in the 1D case except
 - The convolution (either “full” or “valid”), pooling, upsample, etc. operations are performed in 2D case. E.g.

$$\text{upsample}(a) \triangleq \left(\begin{array}{cc|ccc} a_{11} & a_{11} & \dots & a_{1m} & a_{1m} \\ a_{11} & a_{11} & \dots & a_{1m} & a_{1m} \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline a_{n1} & a_{n1} & \dots & a_{nm} & a_{nm} \\ a_{n1} & a_{n1} & \dots & a_{nm} & a_{nm} \end{array} \right) \quad \text{where } a \in R^{n \times m}$$

- The gradient w.r.t. the bias is $\frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i \sum_j (\delta_q^{(l+1)})_{ij}$

where i, j index the elements in the q -th feature map in layer $l + 1$

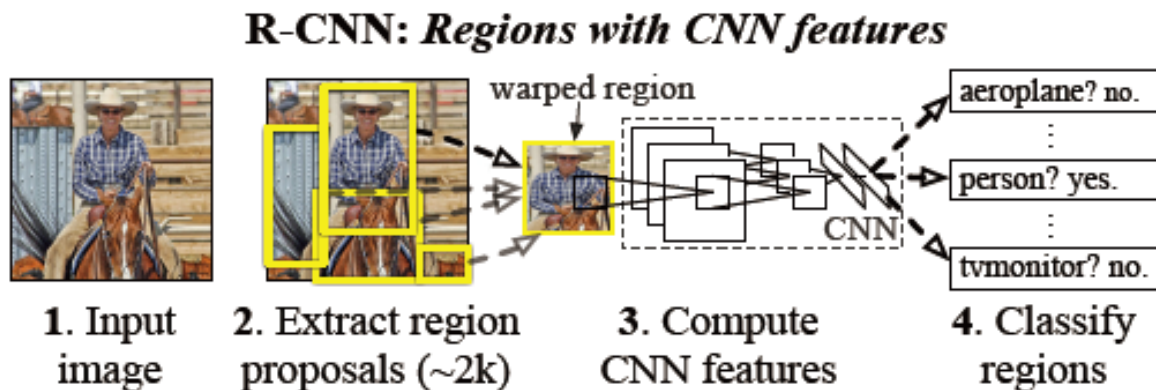
CNN for image classification



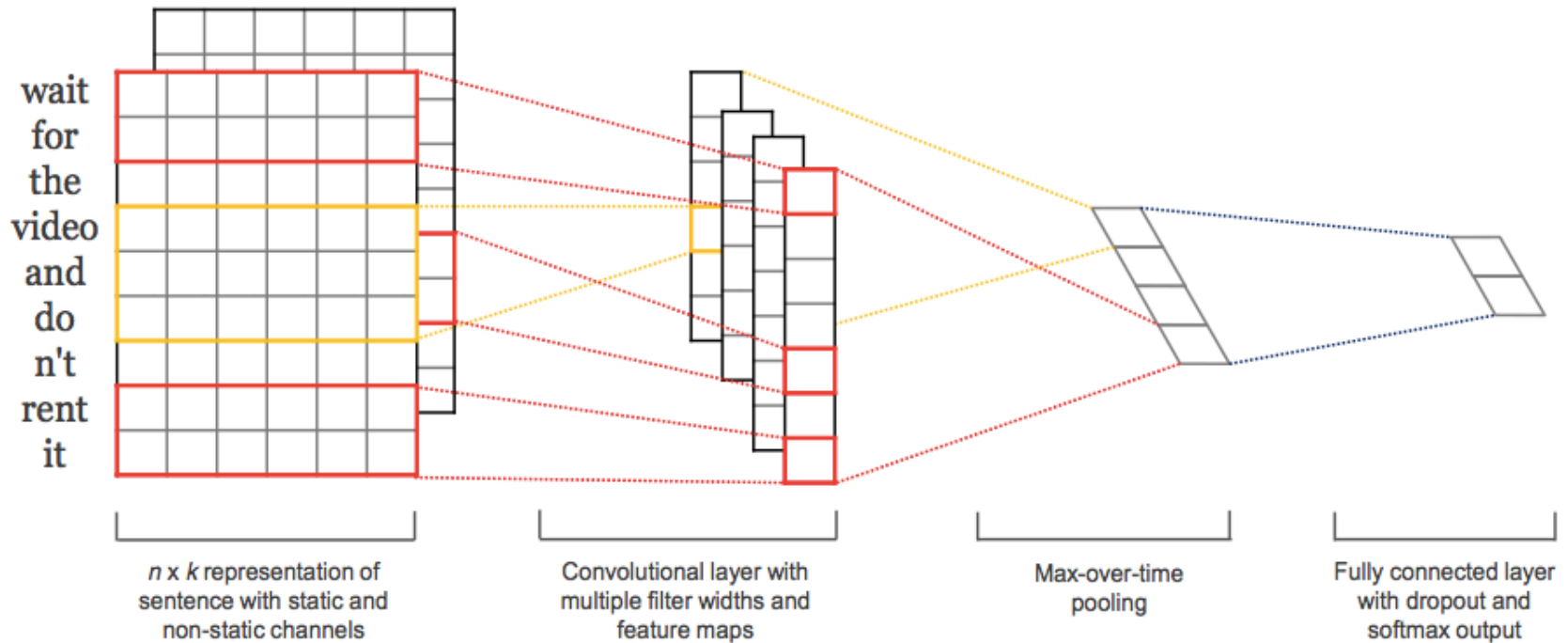
- Network dimension: 150,528(input)-253,440-186,624-64,896-64,896-43,264-4096-4096-1000(output)
- In total: 60 million parameters
- Task: classify 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes
- Results: state-of-the-art accuracy on ImageNet

Generic features for computer vision

- The features trained on 1.2M images in ImageNet are generic
 - They have led to state-of-the-art accuracies on other image classification benchmark datasets such as Caltech-101, CIFAR-10
 - They have led to state-of-the-art accuracies in object detection tasks



CNN for Text Processing



Source codes

- Theano @ University of Montreal
- Caffe @ UC Berkley
- OverFeat @ NYU
- Cuda-convnet by Alex Krizhevsky

Summary

- Forward pass
 - Convolution + pooling
- Backward pass
 - BP algorithm
- Feature combination
 - Increase feature complexity
- 2D CNN