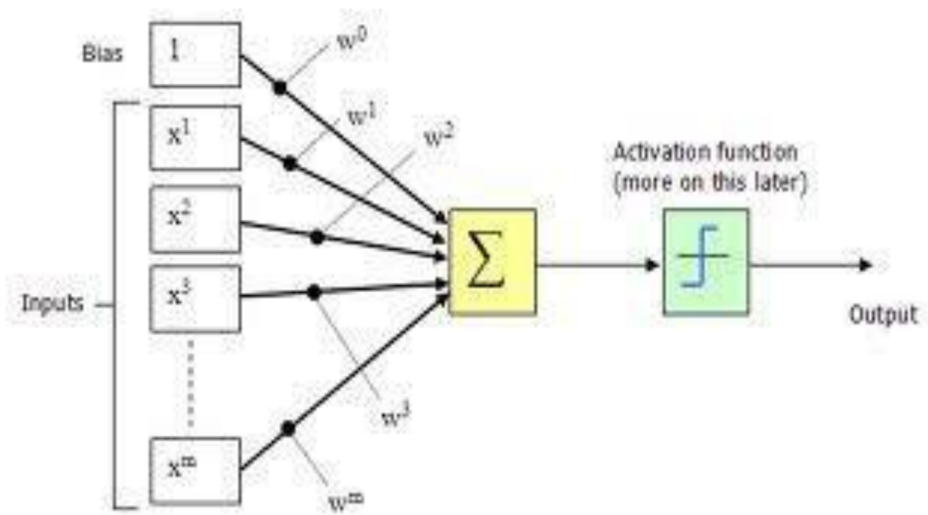
Generic Features

Contents

- Neural network
- Covolutional neural network

Perceptron

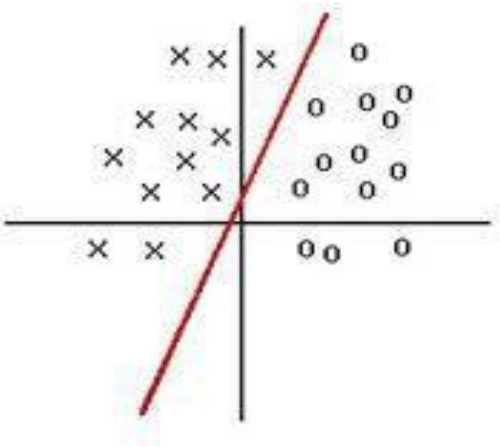
- Perceptron is the basic element of neural network.



- Output $o(\vec{x}) = sng(\vec{w}.\vec{x}) = \begin{cases} 1 \text{ nếu } \vec{w}.\vec{x} > 0 \\ -1 \text{ nếu ngược lại} \end{cases}$

Perceptron

- Preceptron is a hyperplane which seperates data into two parts



Perceptron

- Weight ‘s updating:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

- If we know or define the loss (error) function *E*

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

Perceptron

- Error function and weight is updated by gradient of error function

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n}\right]$$

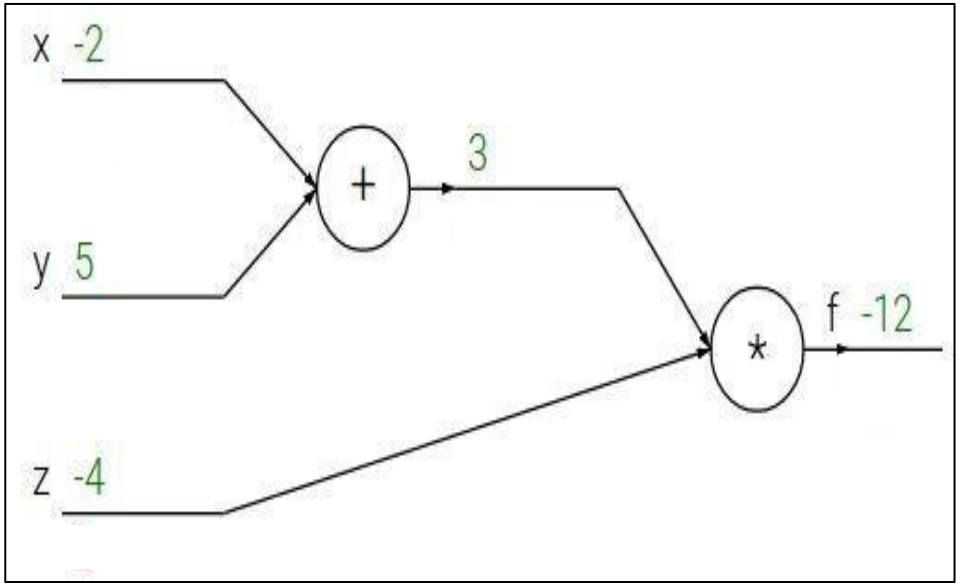
$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta \sum_{d\in D}(t_d - o_d)x_{id}$$

Back propagation

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



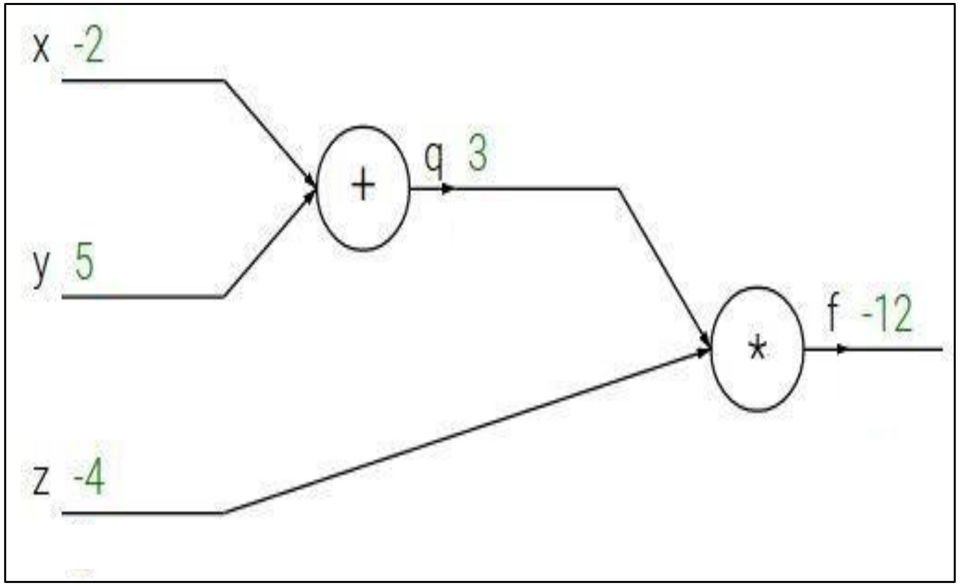
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

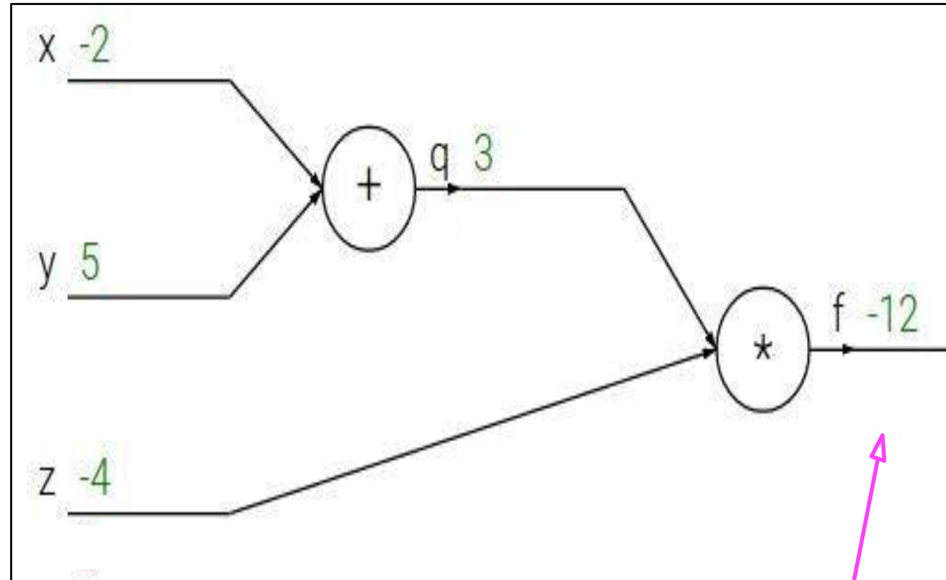
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

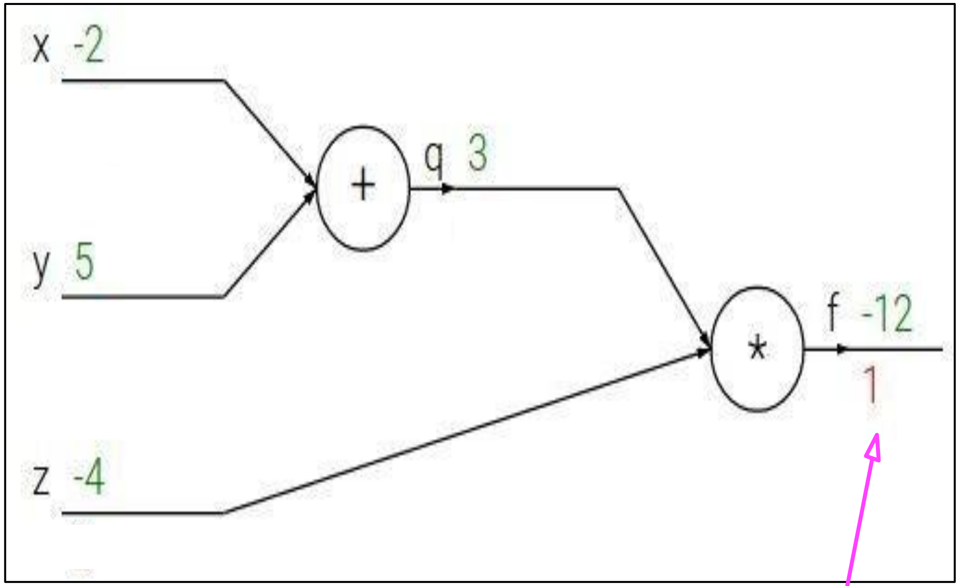
$$\boxed{\frac{\partial f}{\partial f}}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

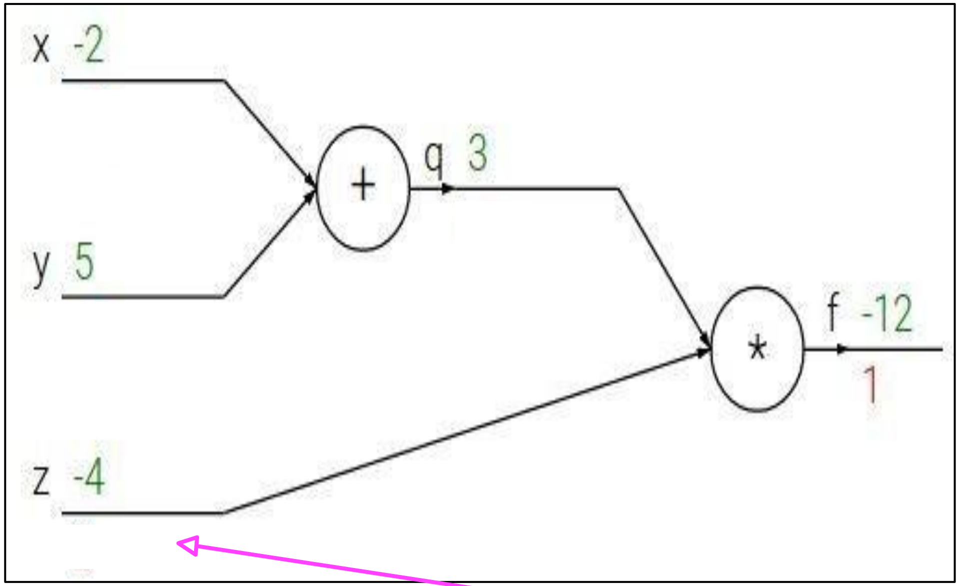
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial f}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

$$\boxed{\frac{\partial f}{\partial z}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

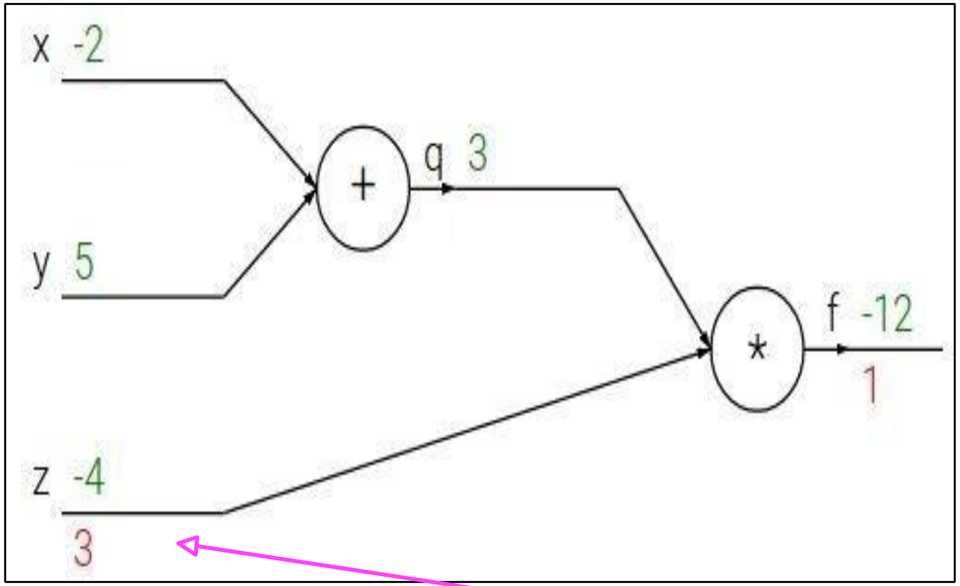
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

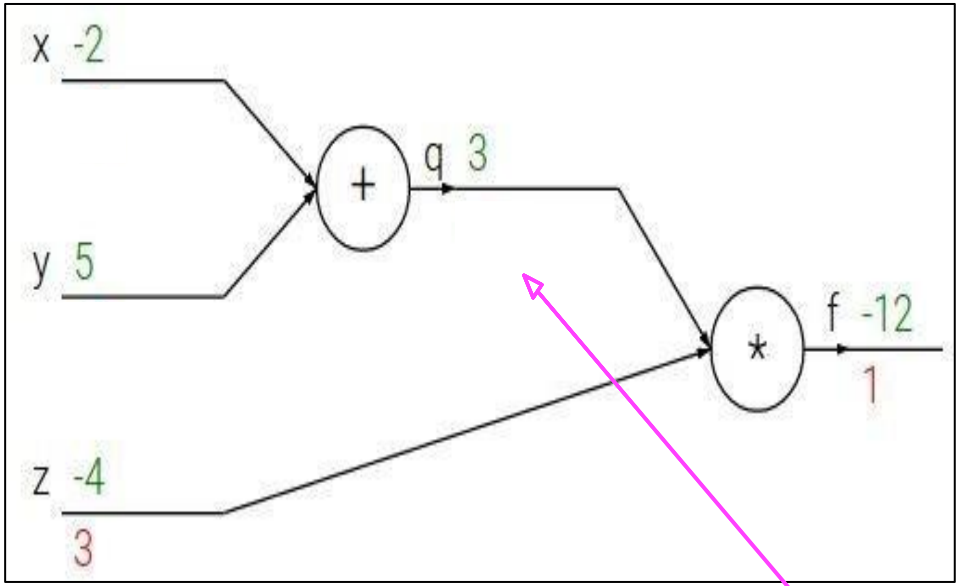
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial z}}$$

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

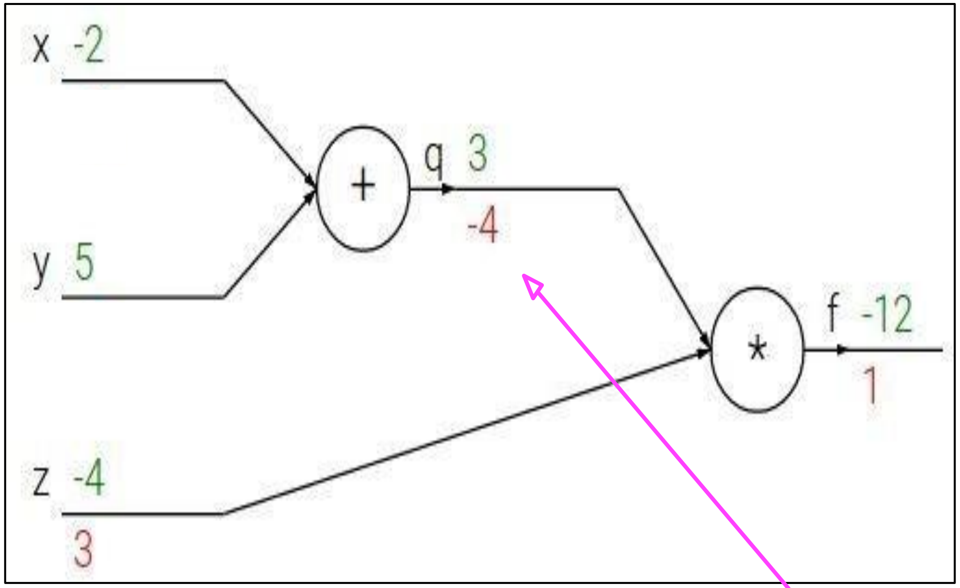
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial q}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

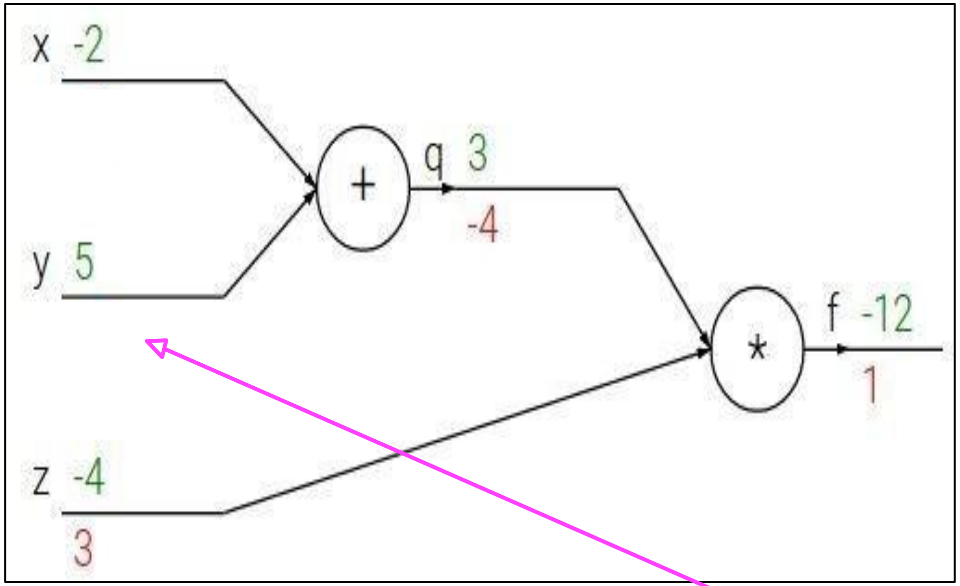
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

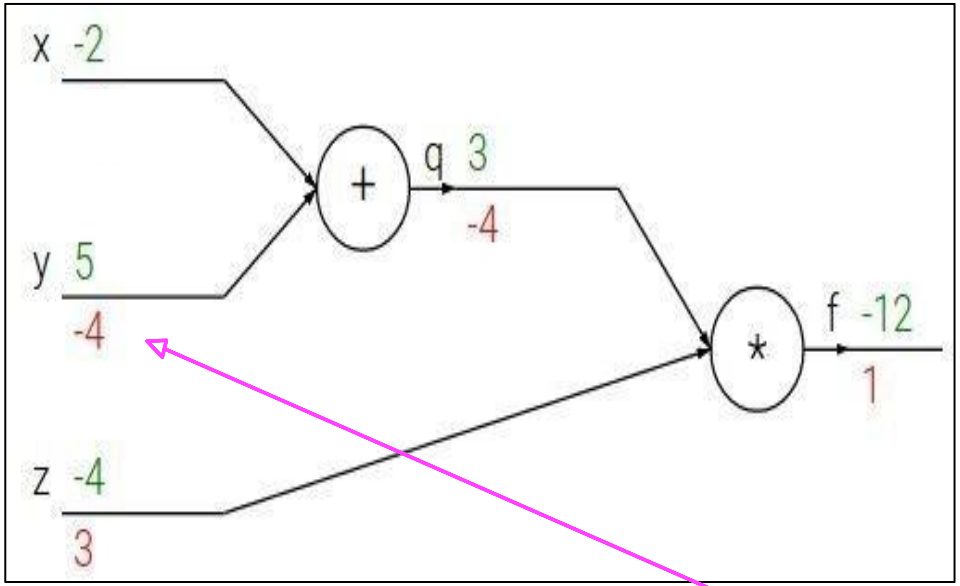
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial y}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

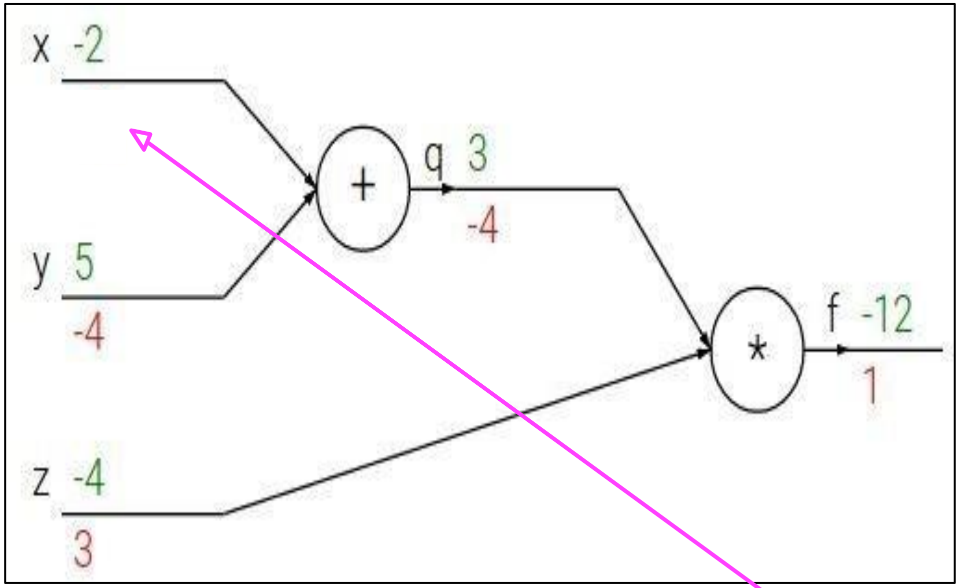
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

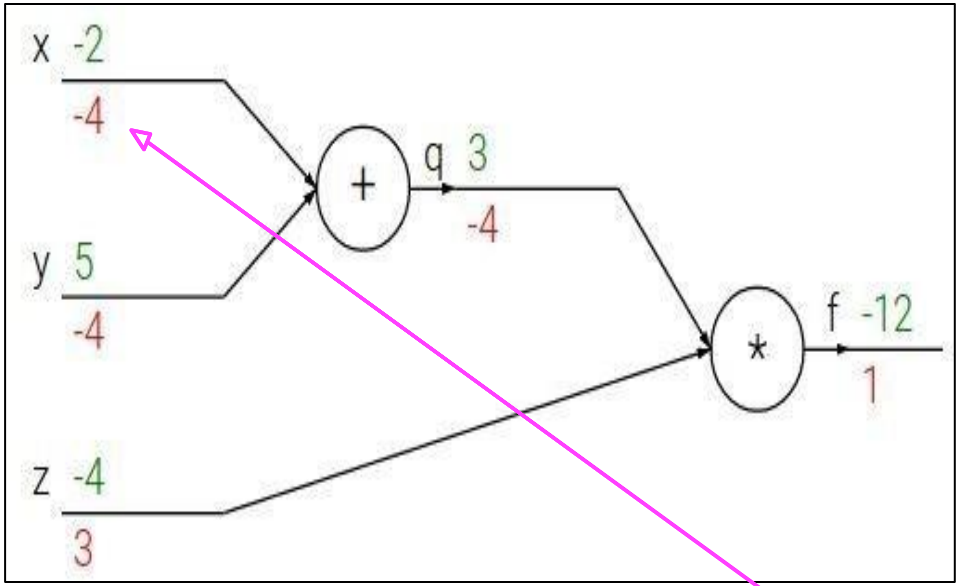
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

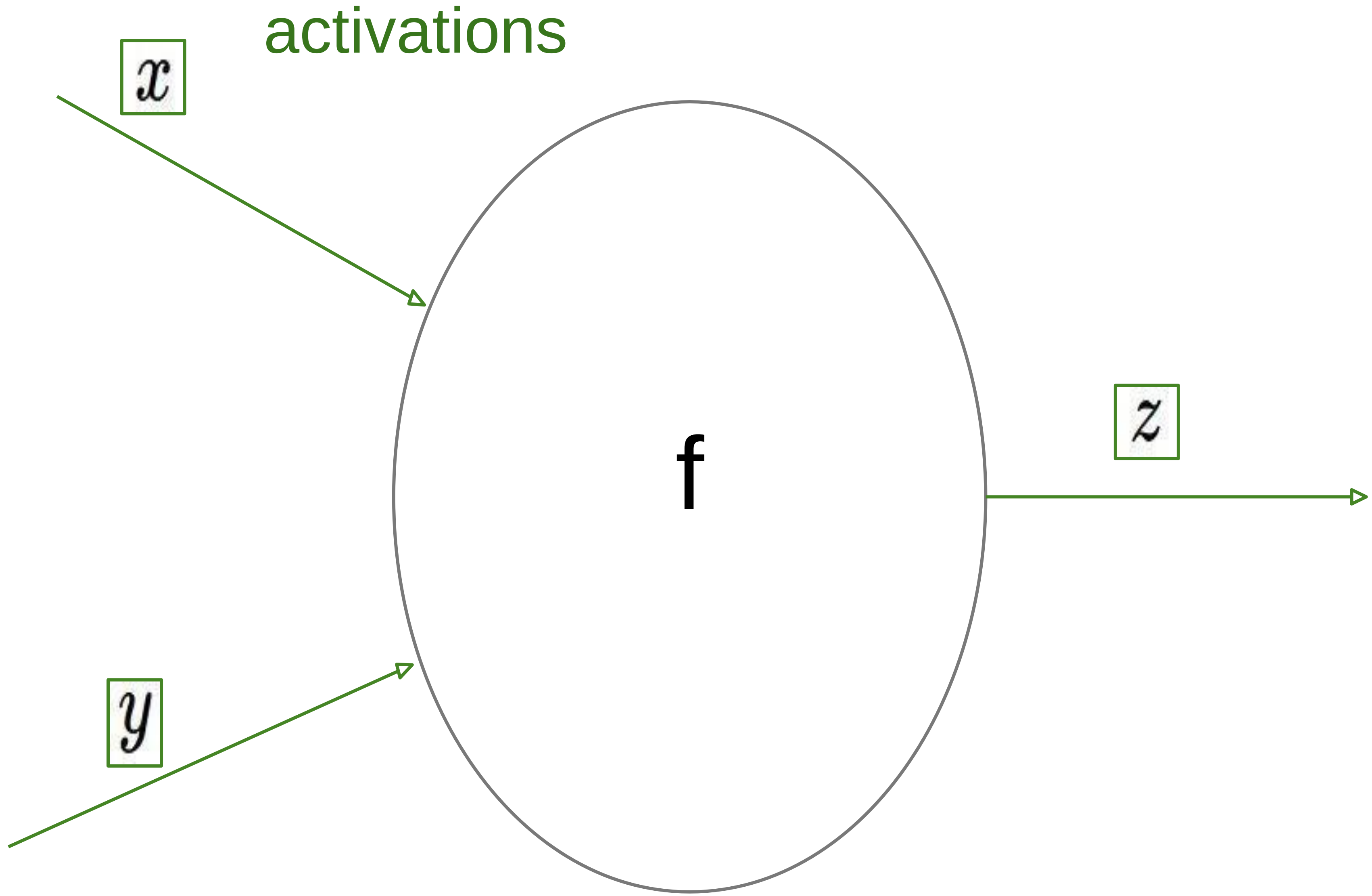
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

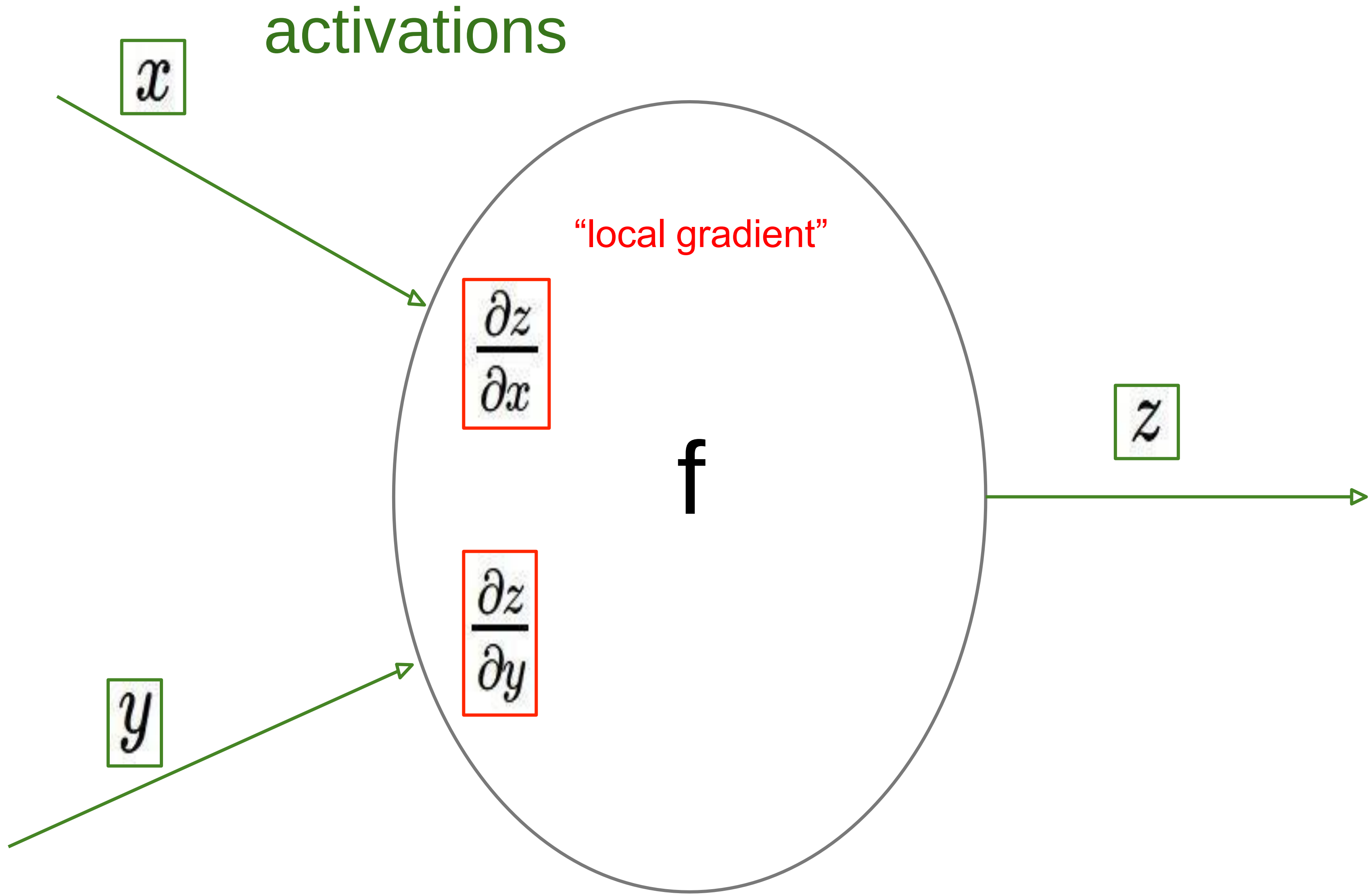
$$\boxed{\frac{\partial f}{\partial x}}$$

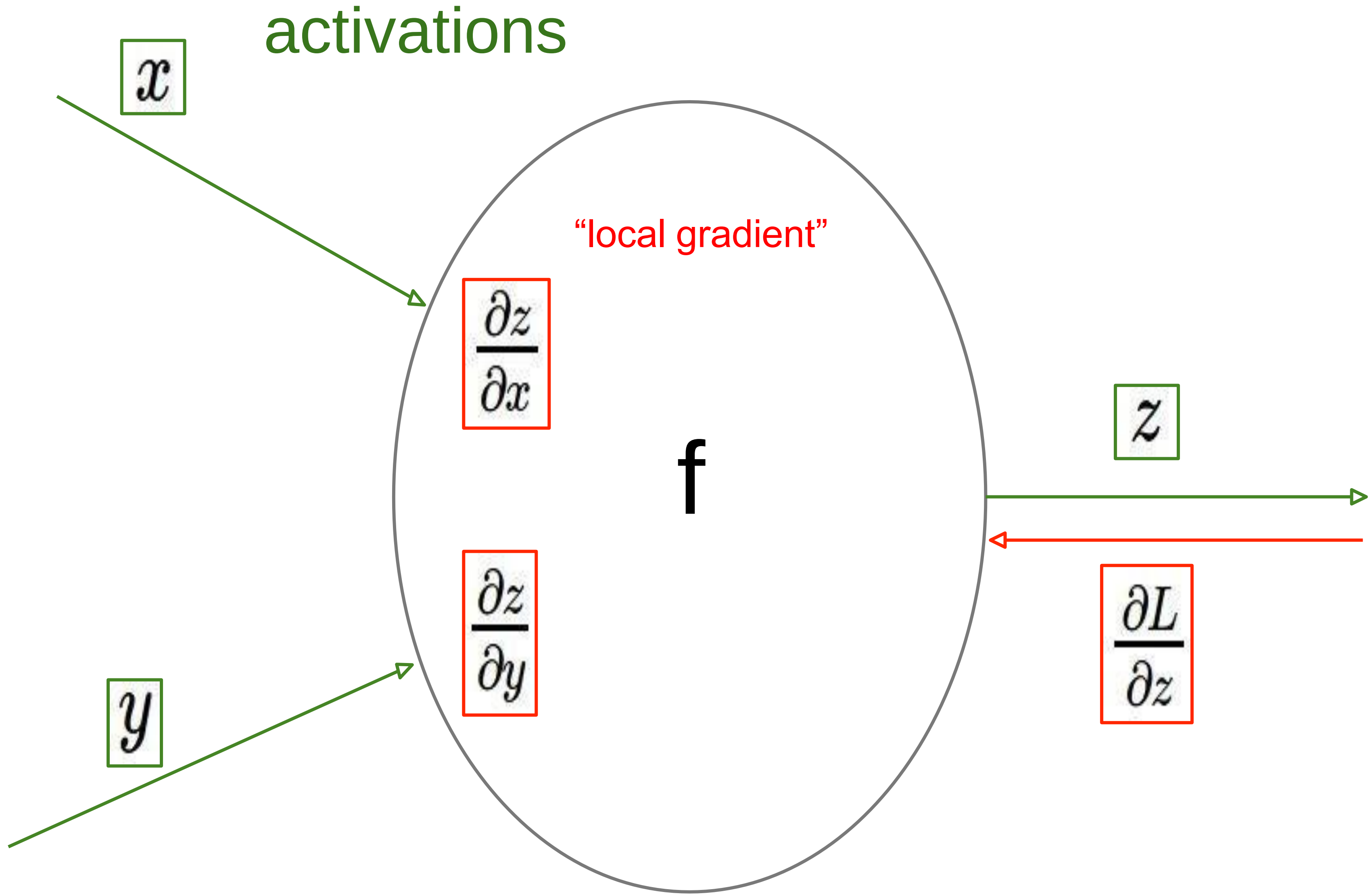
Chain rule:

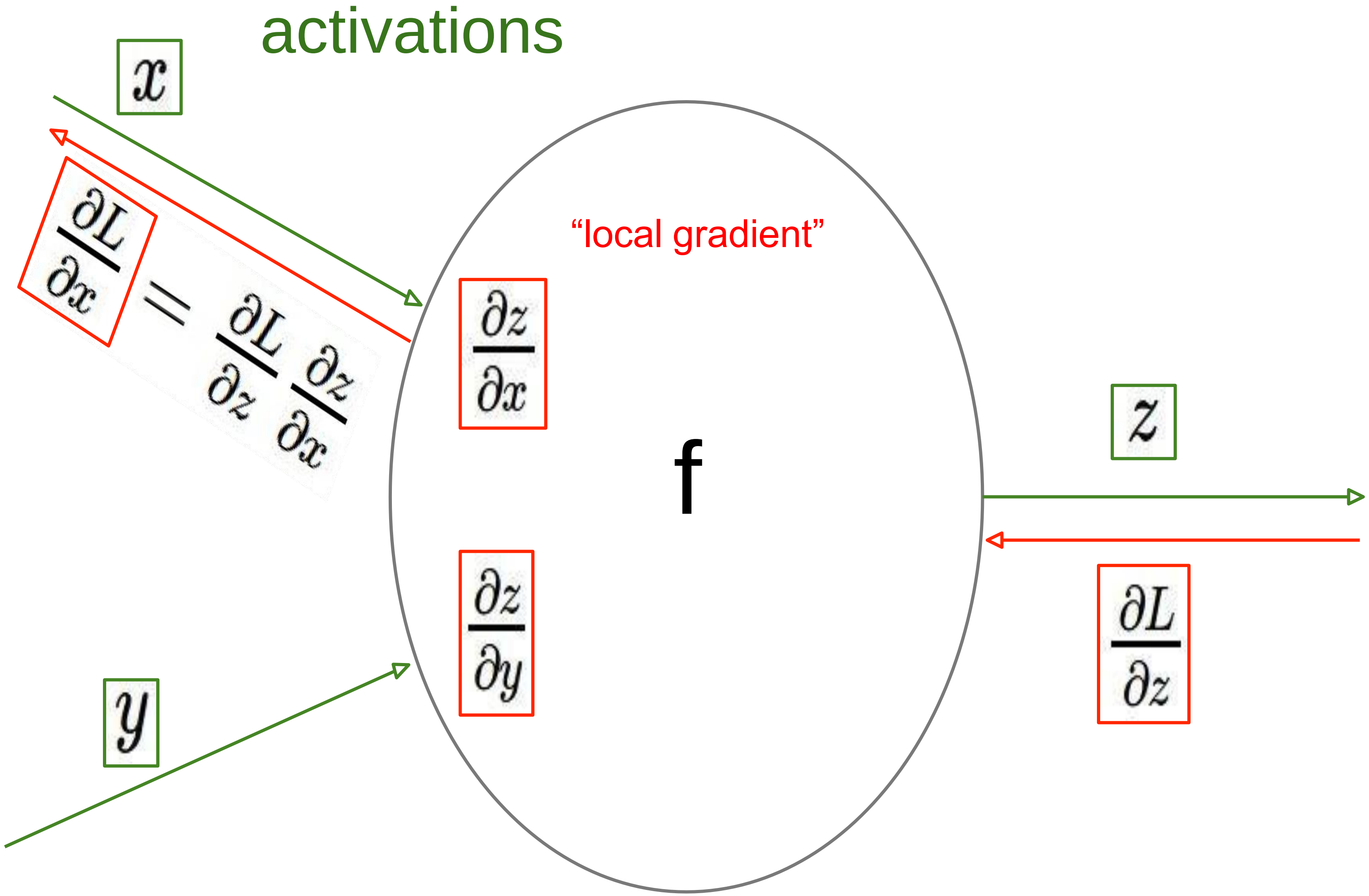
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$


activations
x
y
f
z

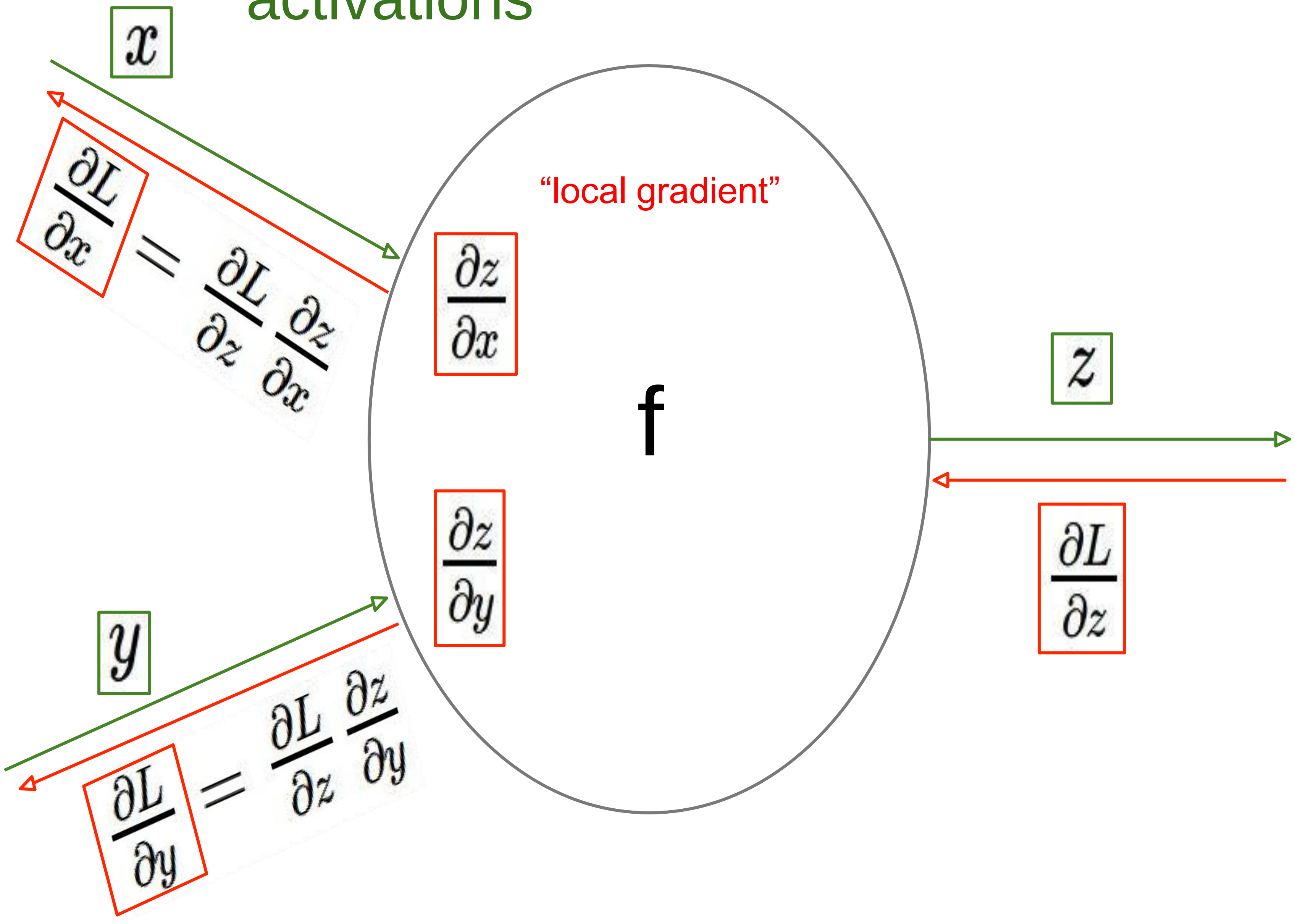

activations
x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y

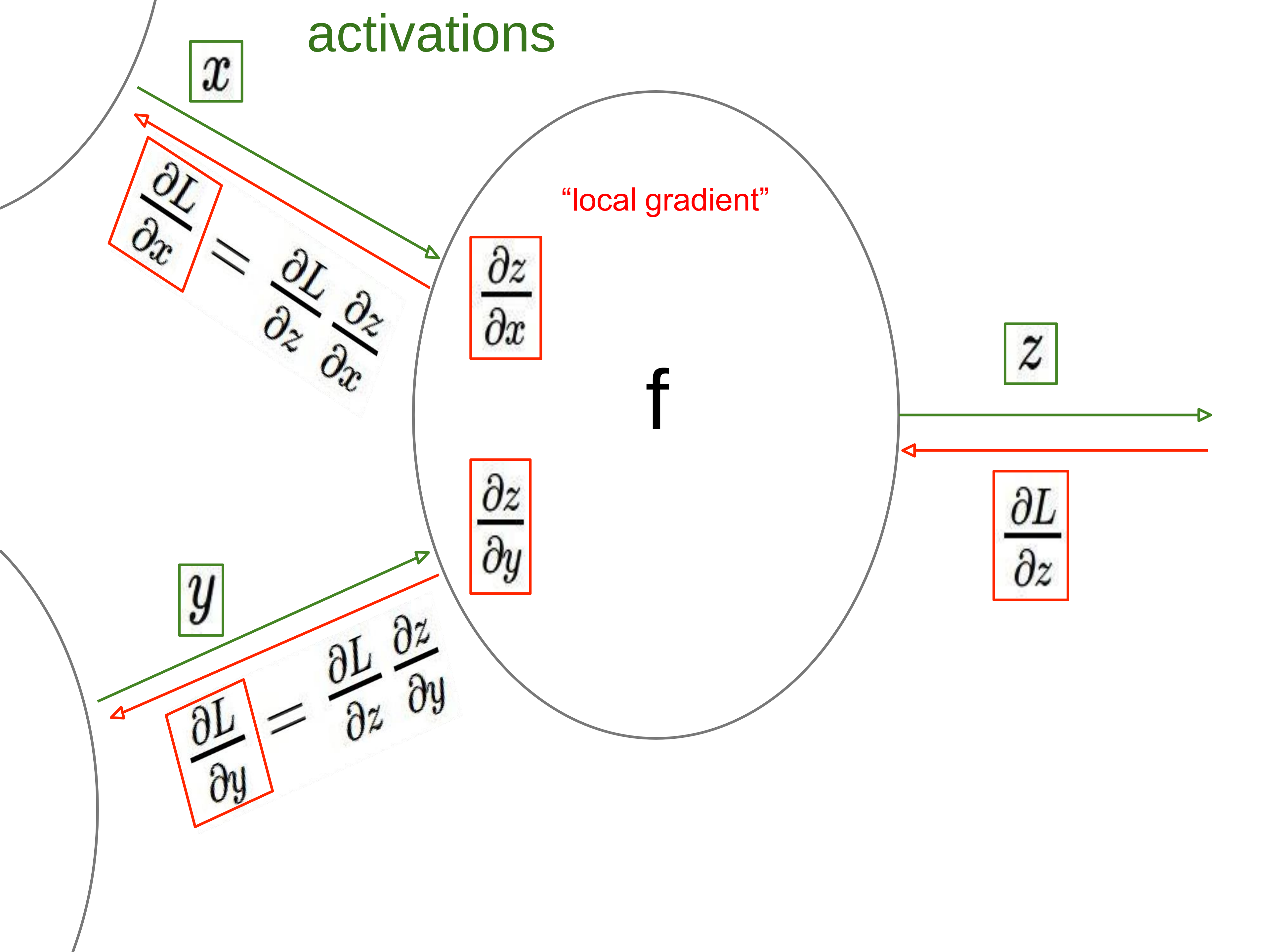

activations
x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y



activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y

activations

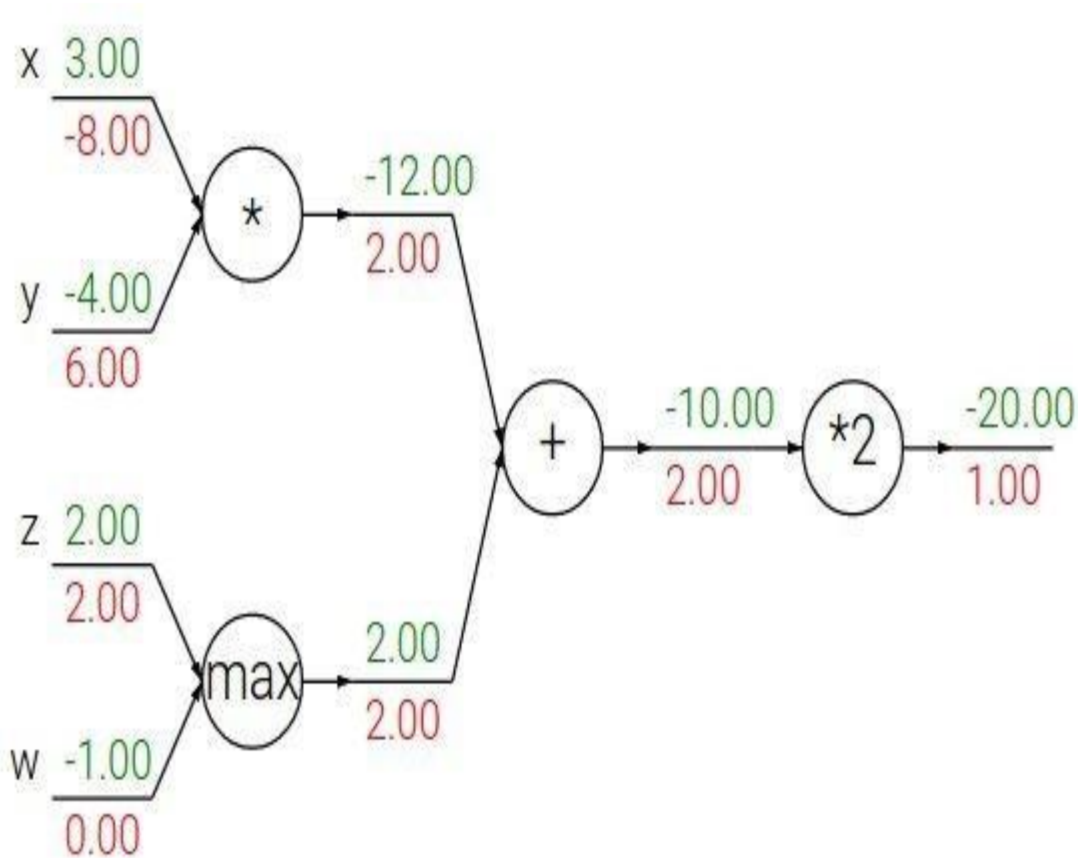




activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

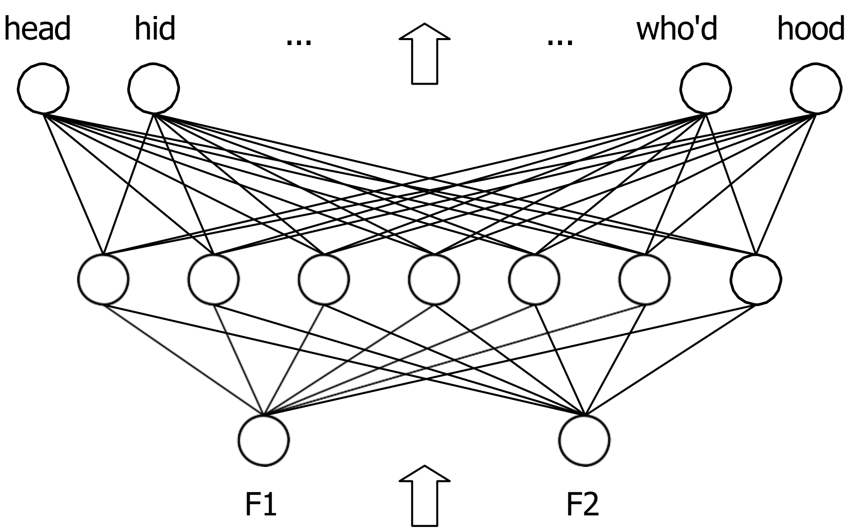
Patterns in backward flow

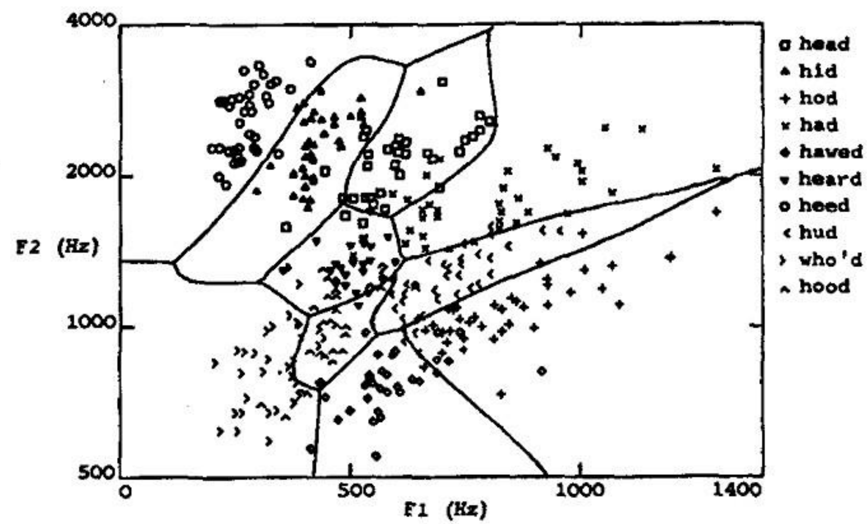
add gate: gradient distributor
max gate: gradient router
mul gate: gradient… "switcher"?



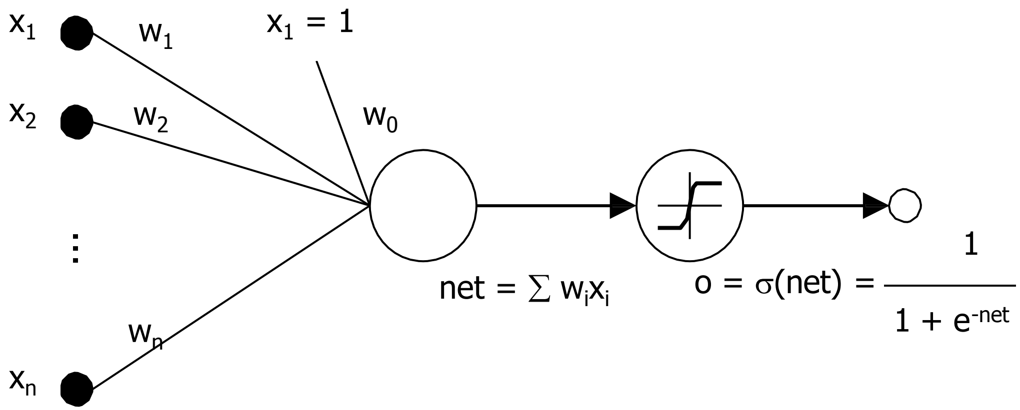
Multilayer network

Perceptron is a linear classifier.
Multilayer network is a nonlinear classifier.





Sigmoid function



$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$$

Error function

Def :

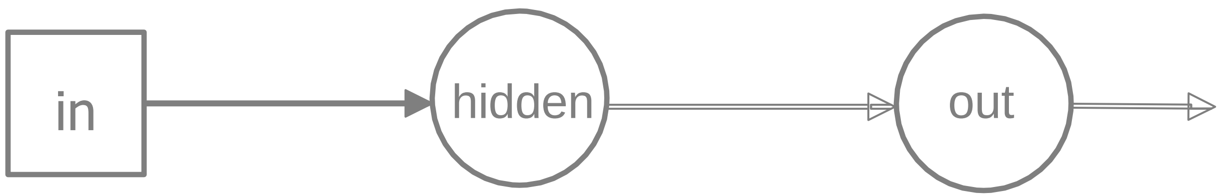
$$E(w) = \tfrac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

ouputs: set of outputs

t_{kd} , o_{kd} : true value and estimated value of outputs

Forward process

BackPropagation (training_examples, η, n_{in}, n_{out}, n_{hidden})

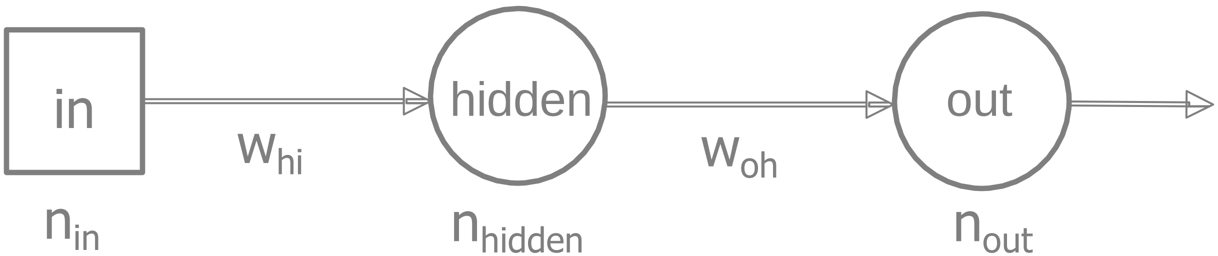


n_{in} n_{hidden} n_{out}

n_{in} Inputs,
n_{hidden} hidden nodes,
n_{out} Outputs.

Forward process

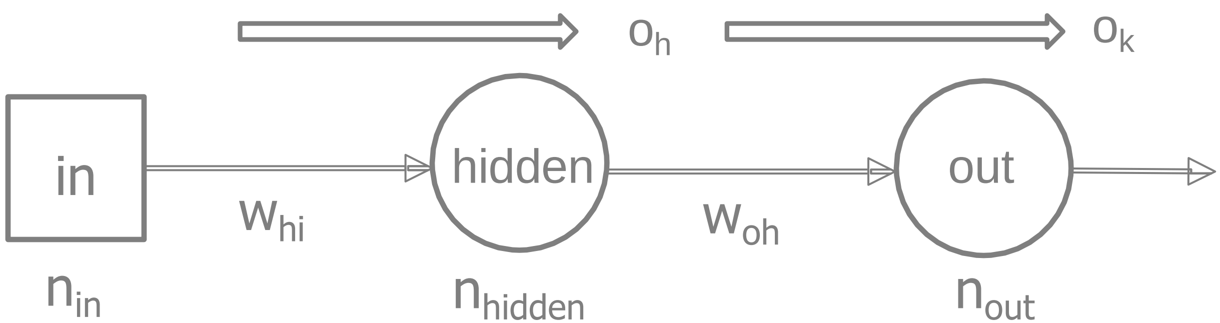
BackPropagation (training_examples, η, n_{in}, n_{out}, n_{hidden})



Randomize initial weight

Forward process

BackPropagation (training_examples, η, n_{in}, n_{out}, n_{hidden})

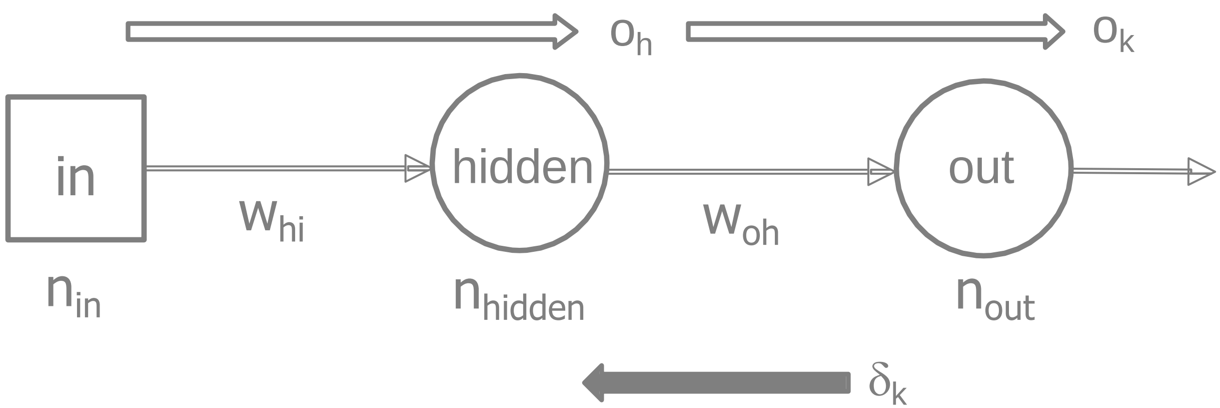


Push each input (*x,y*) into neuron network :
x – data; y - label

1. For each input *x*, calculate ouput o_u for each neuron *u* of the network.

Backward process

BackPropagation (training_examples, η, n_{in}, n_{out}, n_{hidden})

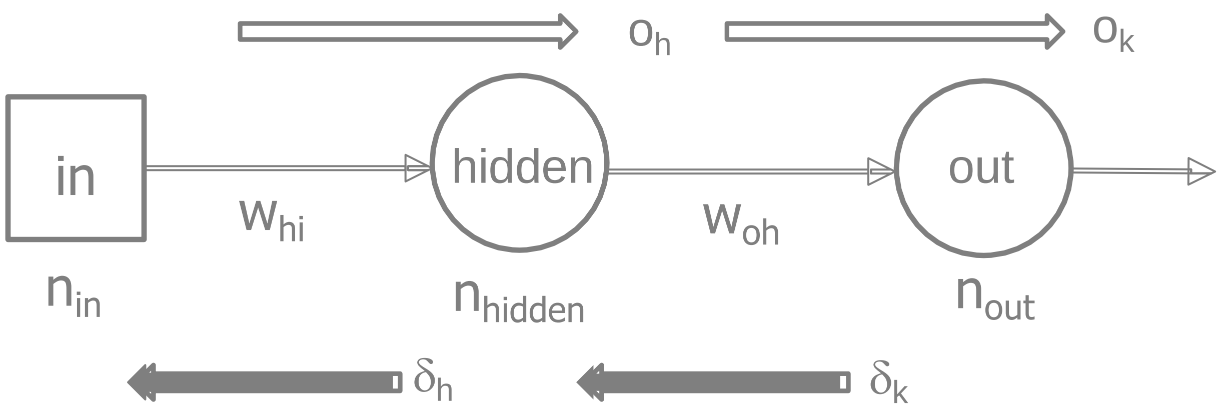


2. For each ouput o_k , calculate transferred error δ_k

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

Backward process

BackPropagation (training_examples, η, n_{in}, n_{out}, n_{hidden})

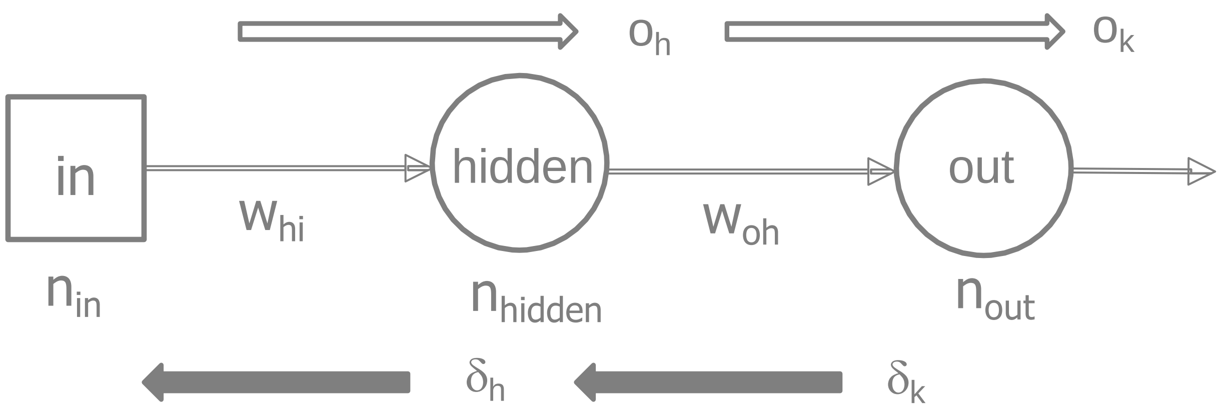


3. For each output of hidden neuron o_h , calculate transferred error δ_h

$$\delta_h = o_h(1 - o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

Backward process

BackPropagation (training_examples, η, n_{in}, n_{out}, n_{hidden})

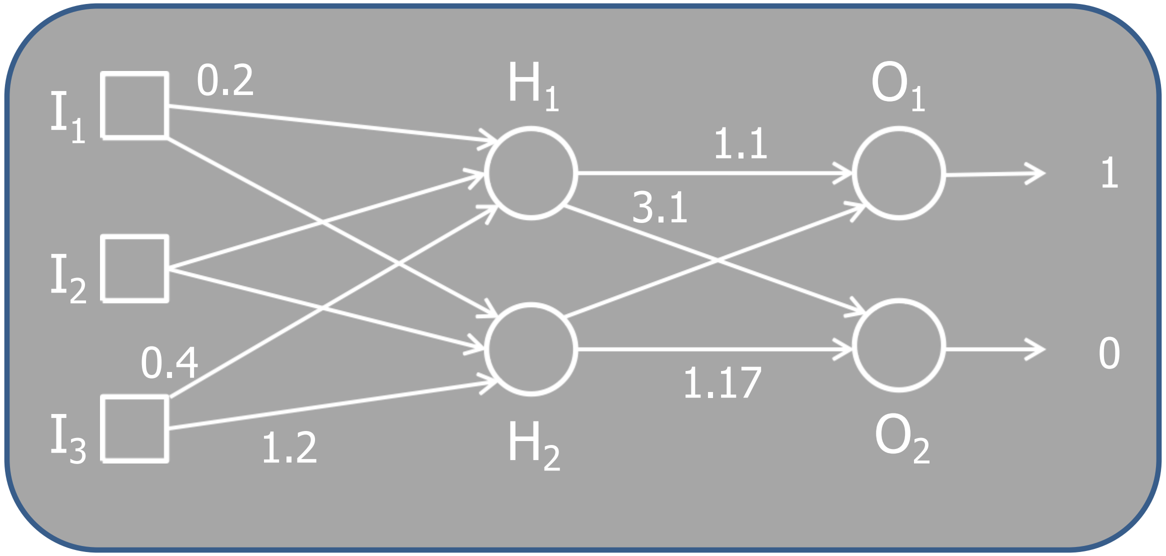


4. Update w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

Examples



Input: $\vec{x} = (10, 30, 20)$

Target: $\vec{t} = (1, 0)$

Learning rate: $\eta = 0.1$

Examples

1. For each input x, calculate output o_u corresponding to neuron u in network

$$o = \sigma(net) = \frac{1}{1 + e^{-net}} \quad \text{with } net = \Sigma w_{ji}x_{ji}$$

H_1: $net_{H1} = 10 * 0.2 + 30 * (-0.1) + 20 * 0.4 = 7$

$o_{H1} = \sigma(net_{H1}) = 0.9990$

H_2: $net_{H2} = 10 * 0.7 + 30 * (-1.2) + 20 * 1.2 = -5$

$o_{H2} = \sigma(net_{H2}) = 0.0067$

O_1: $net_{O1} = 0.9990 * 1.1 + 0.0067 * 0.1 = 1.0996$

$o_{O1} = \sigma(net_{O1}) = 0.7501$

O_2: $net_{O2} = 0.9990 * 3.1 + 0.0067 * 1.17 = 3.1047$

$o_{O2} = \sigma(net_{O2}) = 0.9571$

Examples

2. For each output o_k of output layer, calculate error δ_k

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

$\delta_{O1} = o_{O1}(1 - o_{O1})(t_{O1} - o_{O1}) = 0.750\ (1 - 0.750)(1 - 0.750) = 0.0469$

$\delta_{O2} = o_{O2}(1 - o_{O2})(t_{O2} - o_{O2}) = 0.957\ (1 - 0.957)(0 - 0.957) = -\ 0.0394$

3. For each neuron of hidden layer, calculate error δ_h

$$\delta_h = o_h(1 - o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

$\delta_{H1} = o_{H1}(1 - o_{H1})[(w_{11} * \delta_{O1}) + (w_{21} * \delta_{O2})]$
$= 0.999(1 - 0.999)[(1.1 * 0.0469) + (3.1 * (-0.0394))]$
$= -\ 0.0000705$

$\delta_{H2} = o_{H2}(1 - o_{H2})[(w_{12} * \delta_{O1}) + (w_{22} * \delta_{O2})]$
$= 0.0067(1 - 0.0067)[(0.1 * 0.0469) + (1.17 * (-0.0394))]$
$= -\ 0.000275$

Examples

4. Update weight Output-Hidden w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = \eta\ \delta_j\ x_{ji}$$

Hidden	Output	η	δ_O	$o_H = x_{ji}$	$\Delta = \eta \delta_O x_{ji}$	Old W	New W
H_1	O_1	0.1	0.0469	0.999	0.000469	1.1	1.100469
H_1	O_2	0.1	- 0.0394	0.999	-0.00394	3.1	3.09606
H_2	O_1	0.1	0.0469	0.0067	0.0000314	0.1	0.1000314
H_2	O_2	0.1	- 0.0394	0.0067	-0.0000264	1.17	1.1699736

Examples

4. Update weight w_{ji} in Hidden - Input

Input	Hidden	η	δ_H	x_I	$\Delta = \eta\delta_O x_{ji}$	Old W	New W
I_1	H_1	0.1	-0.0000705	10	-0.0000705	0.2	0.1999295
I_1	H_2	0.1	-0.000275	10	-0.000275	0.7	0.699725
I_2	H_1	0.1	-0.0000705	30	-0.0002115	-0.1	-0.1000705
I_2	H_2	0.1	-0.000275	30	-0.000825	-1.2	-1.200825
I_3	H_1	0.1	-0.0000705	20	-0.000141	0.4	0.399859
I_3	H_2	0.1	-0.000275	20	-0.00055	1.2	1.19945

Examples

Updated value

$$\Delta \mathbf{w}_{ji}(\mathbf{n}) = \boldsymbol{\eta}\, \delta_i \mathbf{x}_{ji} + \alpha \Delta \mathbf{w}_{ji}(\mathbf{n-1})$$

n: iteration number

$0 \leq \alpha < 1$: momentum value

Equations

- $E_d = ½ \sum_{k \in ouputs} (t_k - o_k)^2$

$$\Delta w_{ji} = - \eta \frac{\partial E_d}{\partial w_{ji}}$$

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$

Equations

<u>Case 1:</u> Updated weights of Output layer

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} ½ \, \Sigma_{k \in ouputs} (t_k - o_k)^2 = \frac{\partial}{\partial o_j} ½ \, (t_j - o_j)^2 \qquad (\frac{\partial}{\partial o_j} (t_k - o_k)^2 = 0 \text{ với } k \neq j)$$

$$= 2*1/2*(t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j} = -(t_j - o_j)$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = o_j(1 - o_j)$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j) \quad => \quad \boxed{\Delta w_{ji} = \eta(t_j - o_j)\, o_j(1 - o_j) x_{ji}}$$

Equations

<u>Case 2:</u> Updated weights of Hidden – Input

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \frac{\partial o_j}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)$$

Let $\delta_j = - \frac{\partial E_d}{\partial net_j}$ => $\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} -\delta_k \, w_{kj}$

$$\boxed{\Delta w_{ji} = \eta \, \delta_j \, x_{ji}}$$

Convolutional Networks

Neural Networks that use convolution in place of general matrix multiplication in atleast one layer

Next:

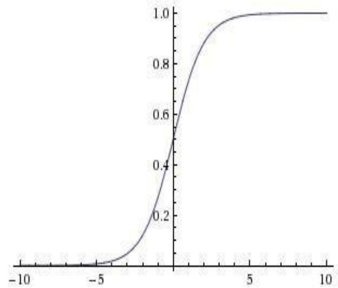
- What is convolution?
- What is pooling?
- What is the motivation for such architectures (remember LeNet?)

Activation function

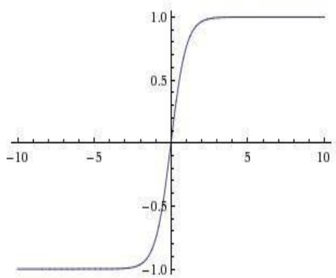
Activation Functions

Sigmoid

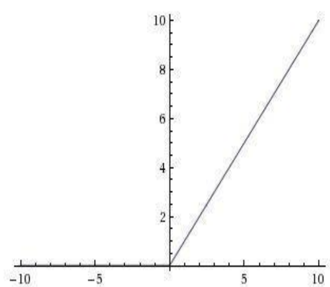
$$\sigma(x) = 1/(1+e^{-x})$$



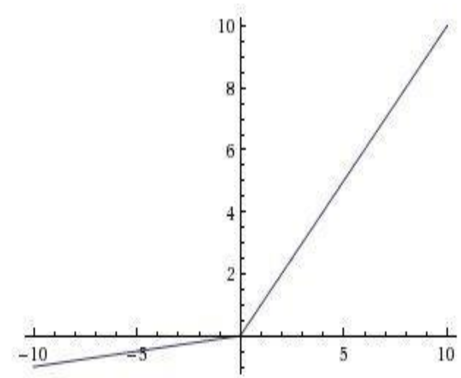
tanh tanh(x)



ReLU max(0,x)



Leaky ReLU

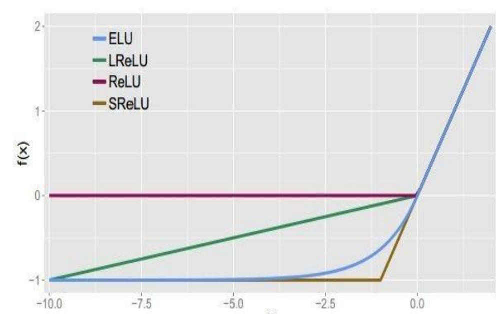


Maxout

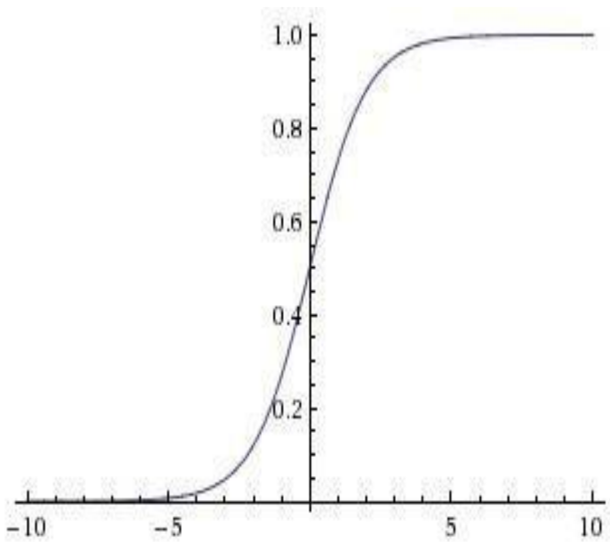
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



Activation Functions

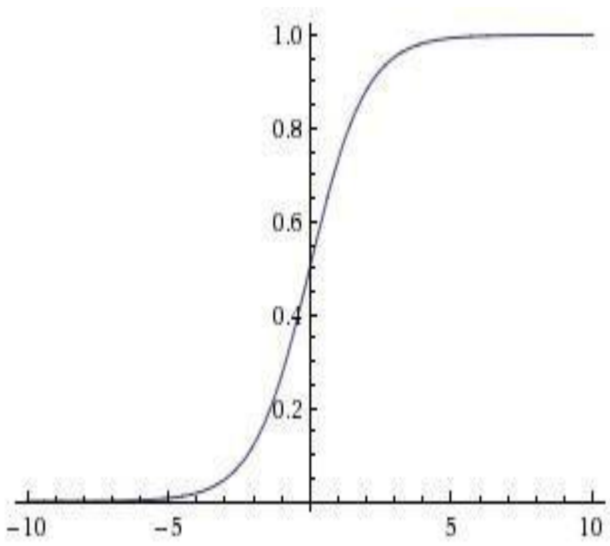


Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

Activation Functions



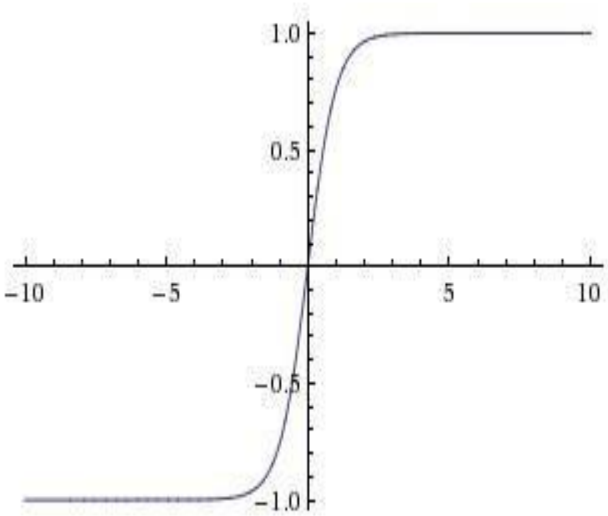
Sigmoid

$$\sigma(x) = 1/(1+e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

Activation Functions

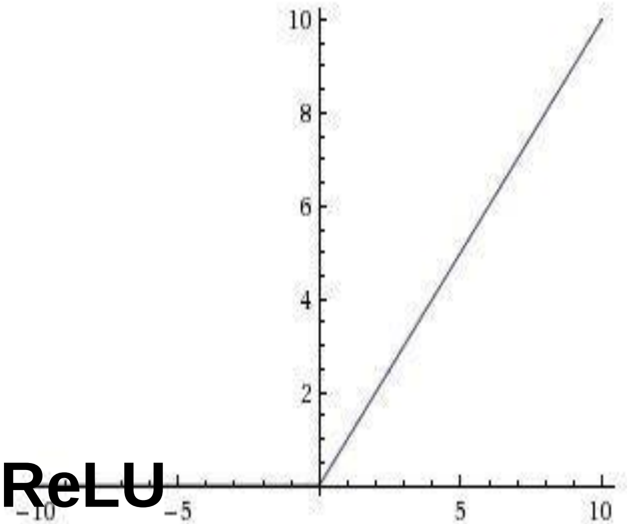


tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

Activation Functions



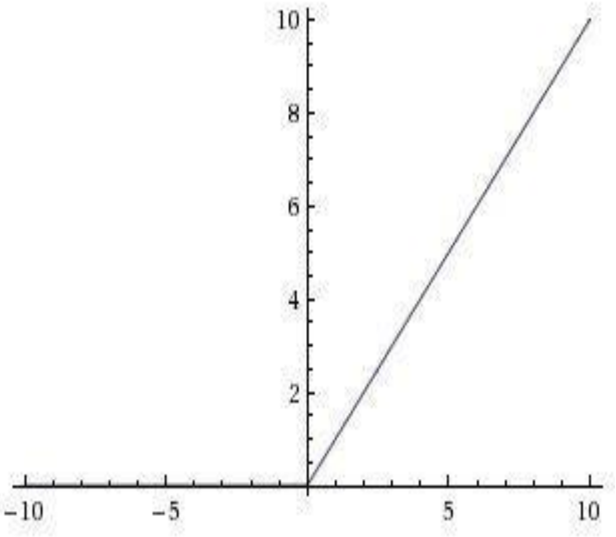
ReLU
(Rectified Linear Unit)

Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

[Krizhevsky et al., 2012]

Activation Functions



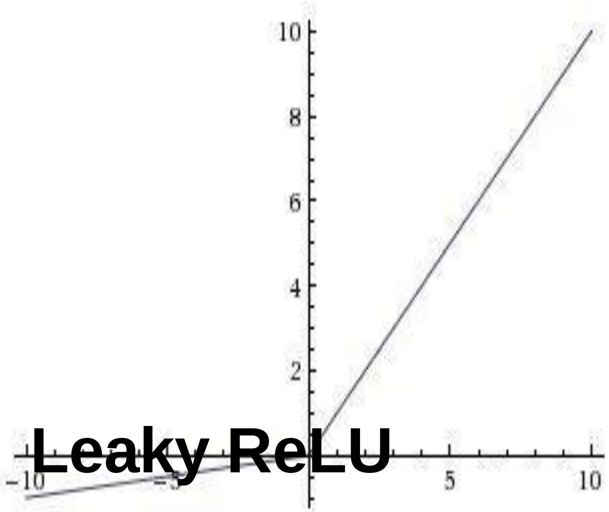
ReLU
(Rectified Linear Unit)

Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

- Not zero-centered output
- ReLU units can “die”

Activation Functions



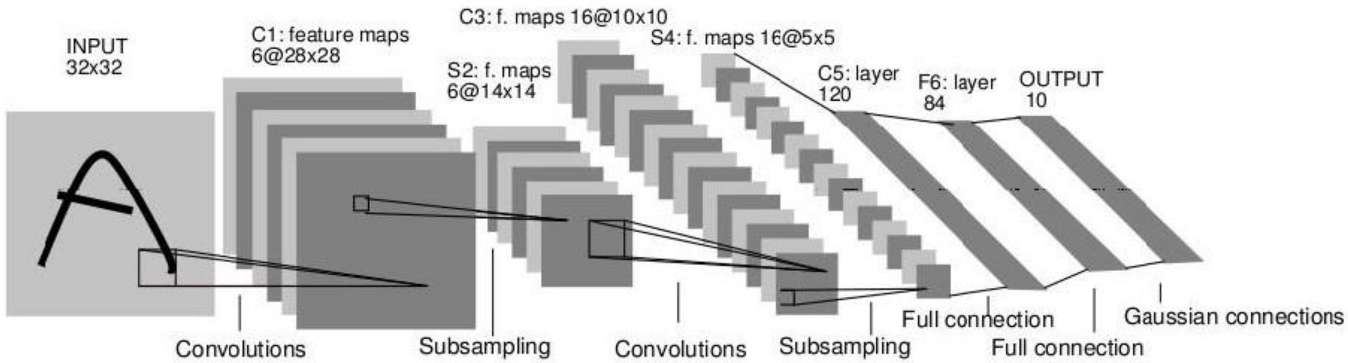
Leaky ReLU

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

[Mass et al., 2013] [He et al., 2015]

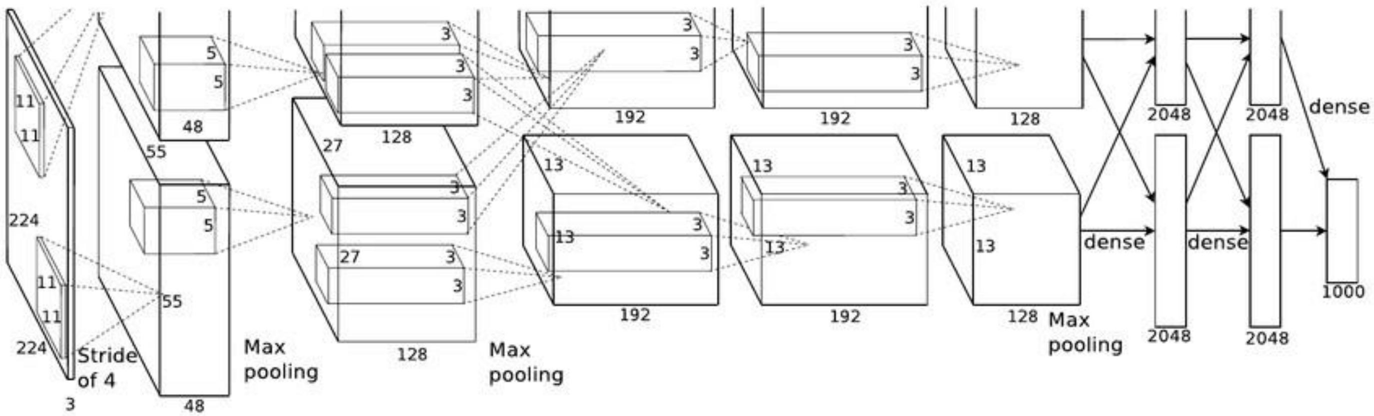
$$f(x) = \max(0.01x, x)$$

LeNet-5 (LeCun, 1998)



The original Convolutional Neural Network model goes back to 1989 (LeCun)

AlexNet (Krizhevsky, Sutskever, Hinton 2012)



ImageNet 2012 15.4% error rate

Convolutional Neural Network

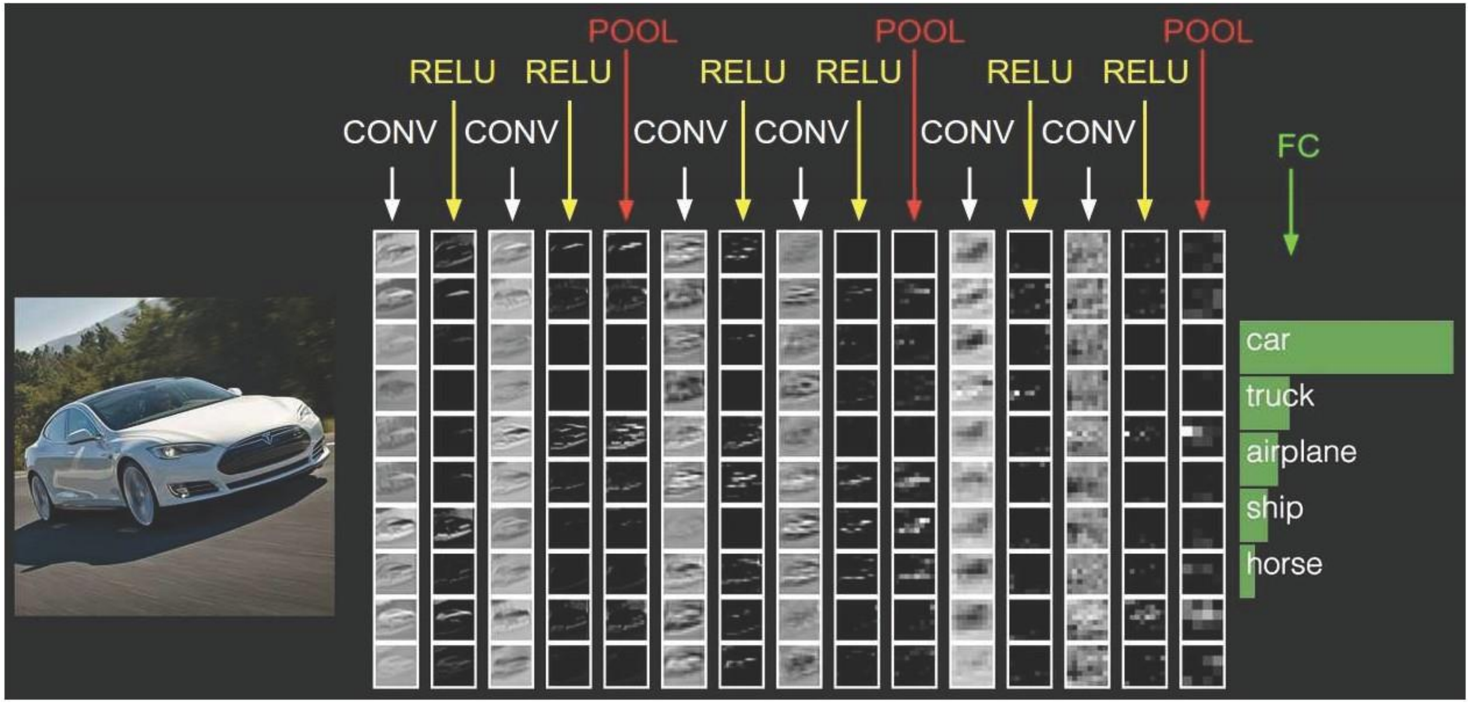
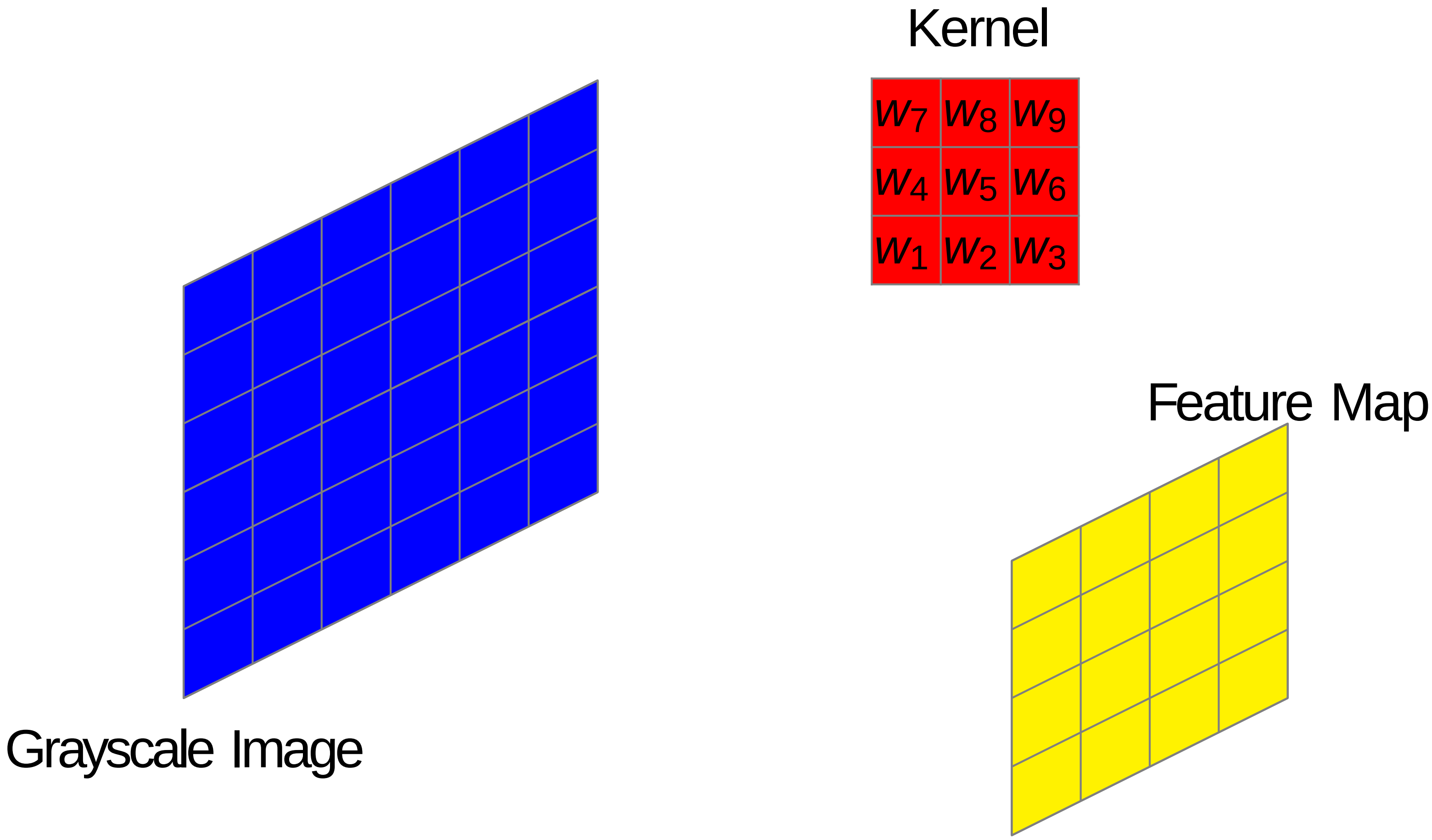


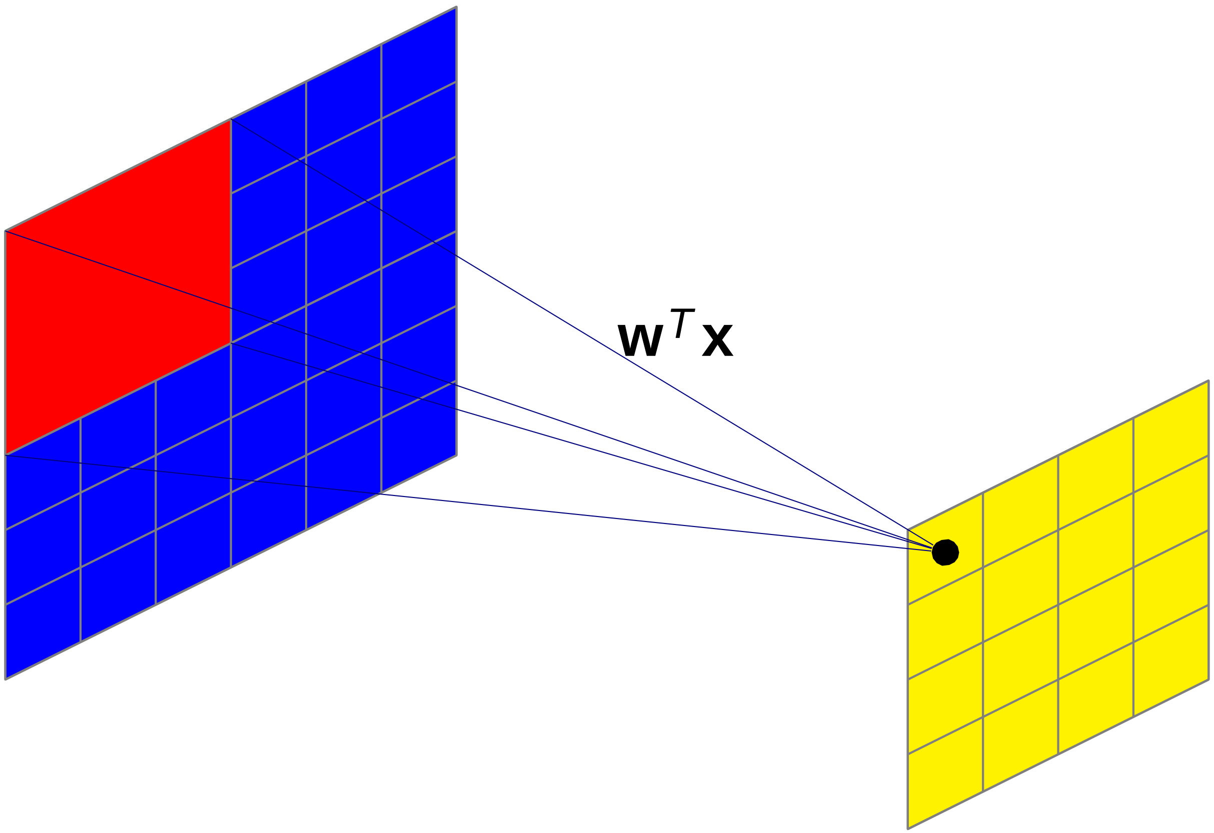
Figure: Andrej Karpathy

Convolution

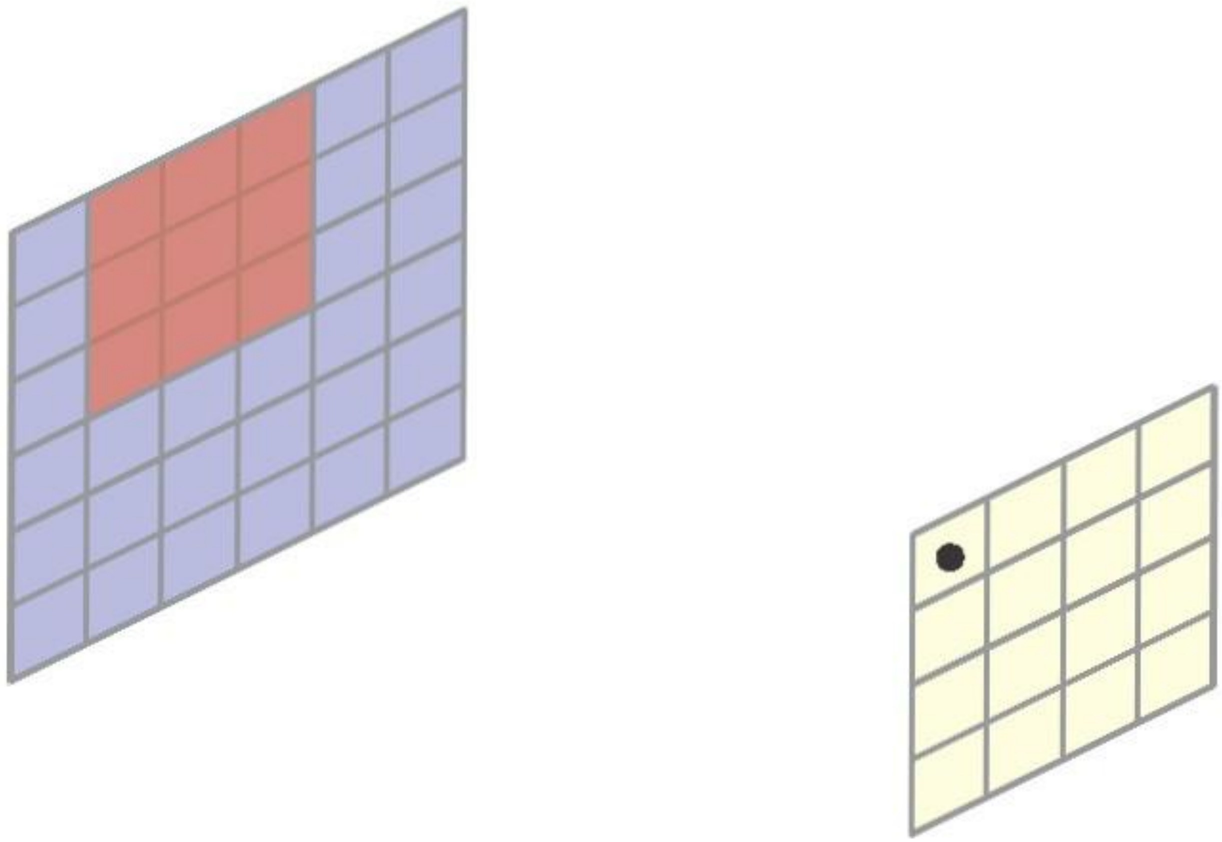


- Convolve image with kernel having weights **w** (learned by backpropagation)

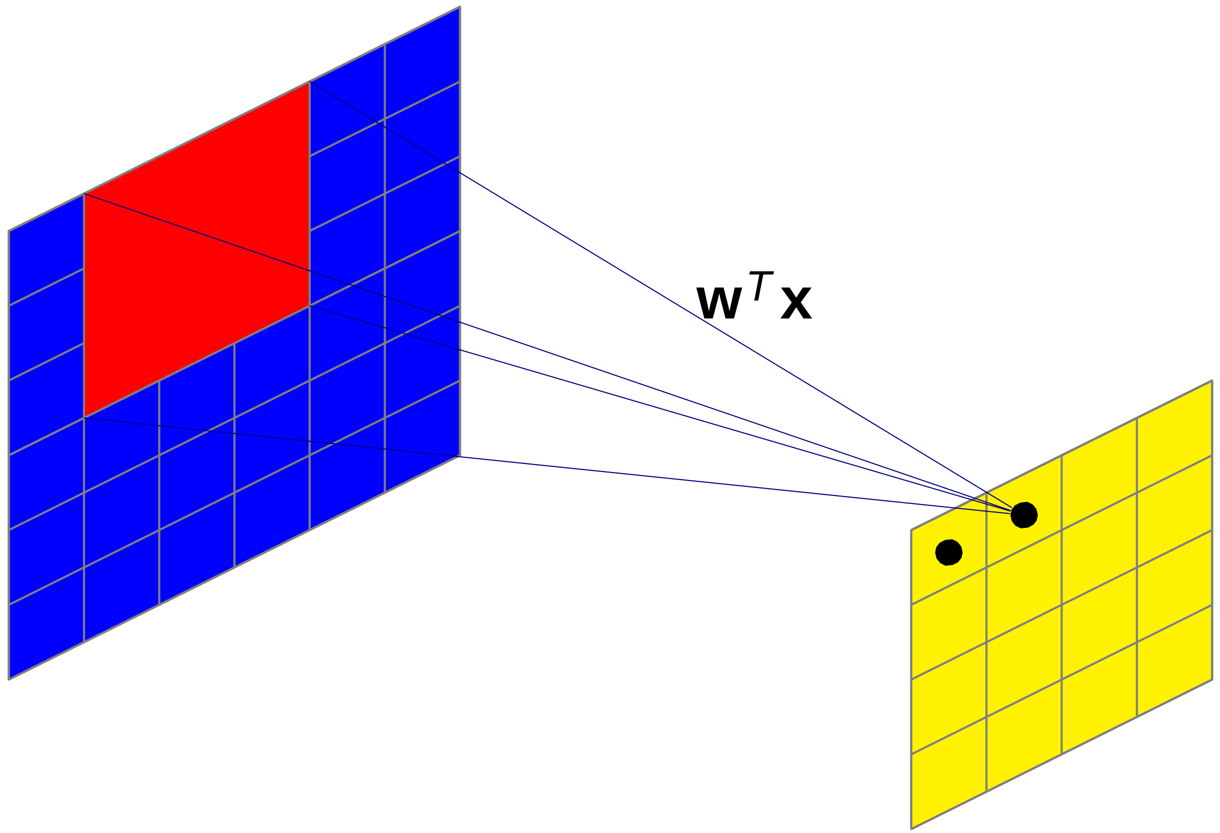
Convolution



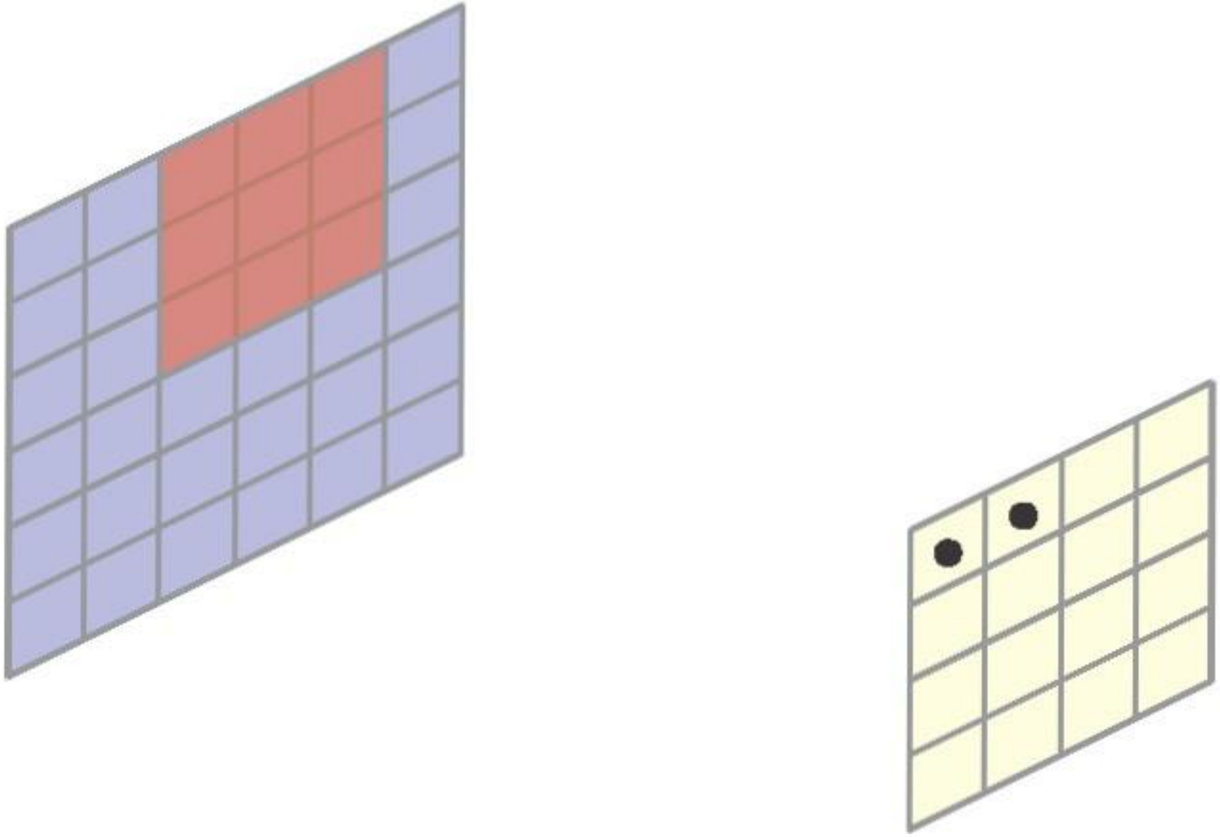
Convolution



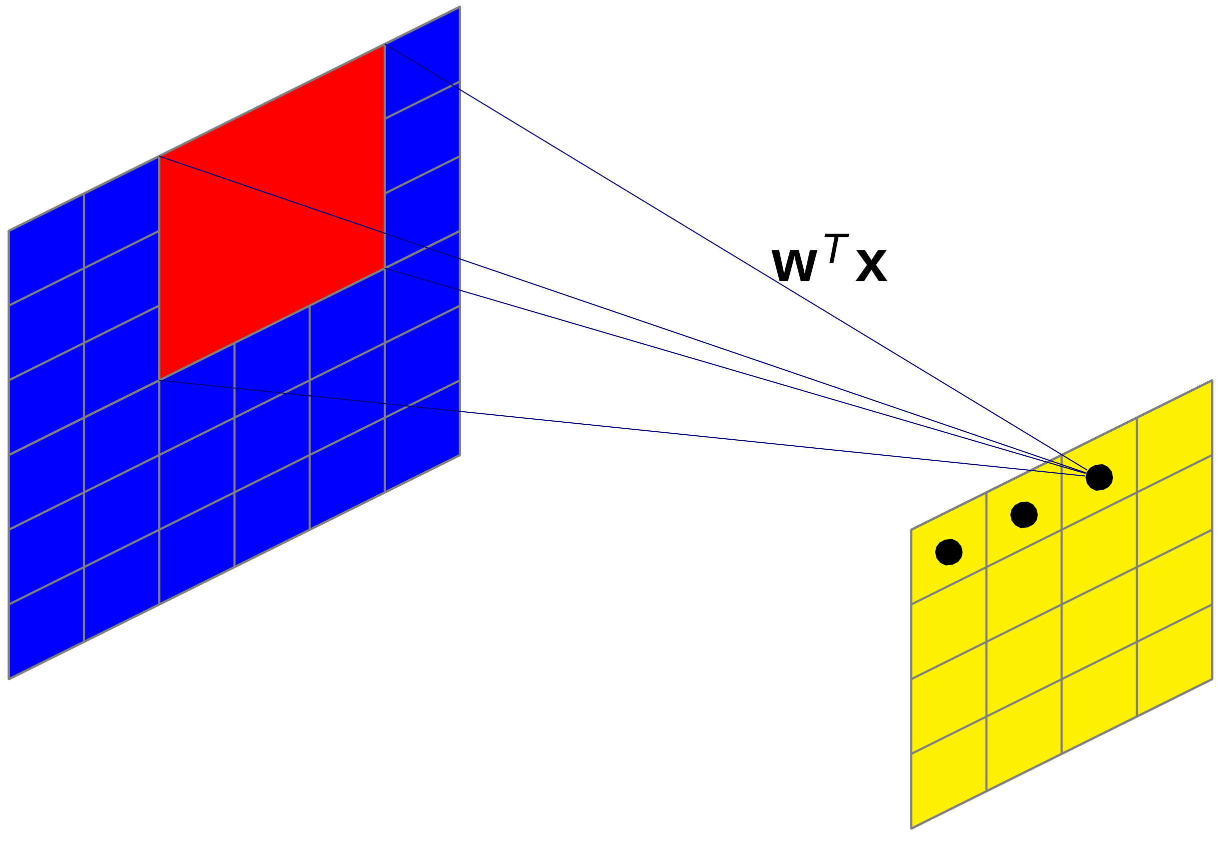
Convolution



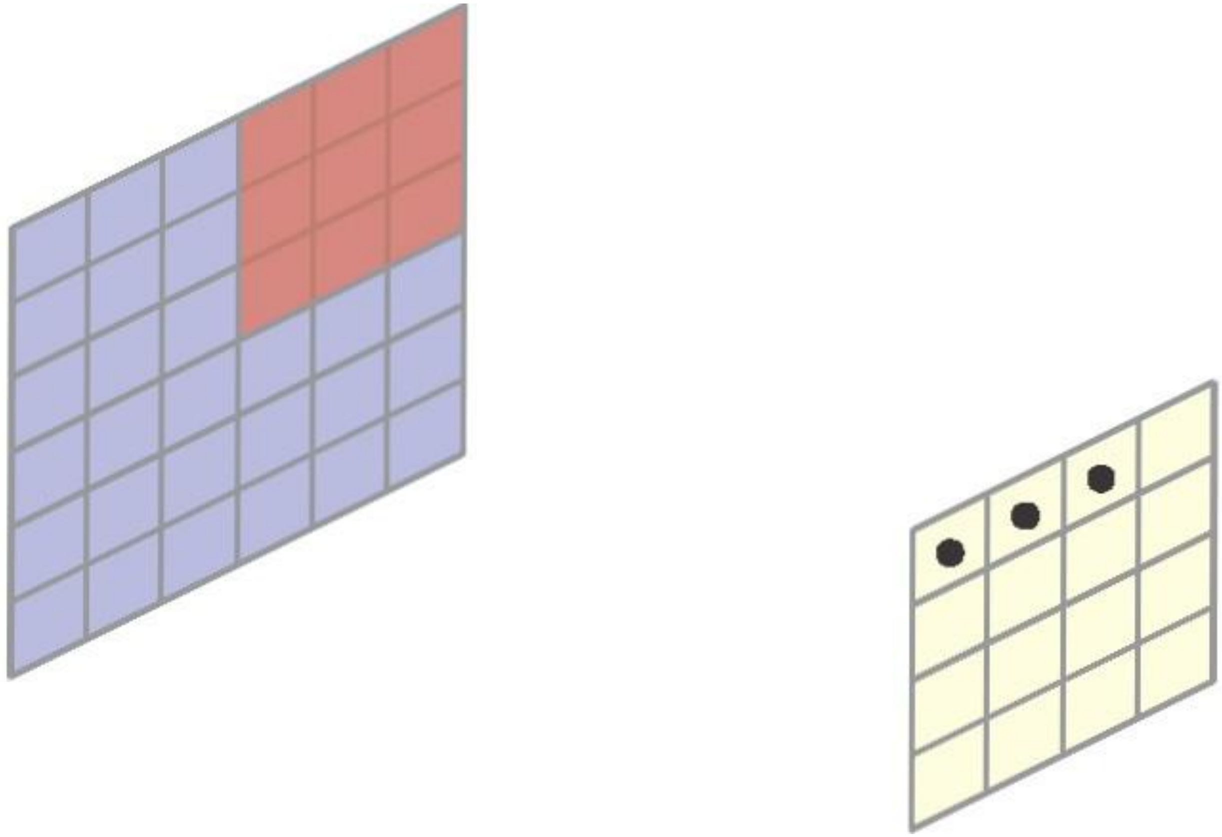
Convolution



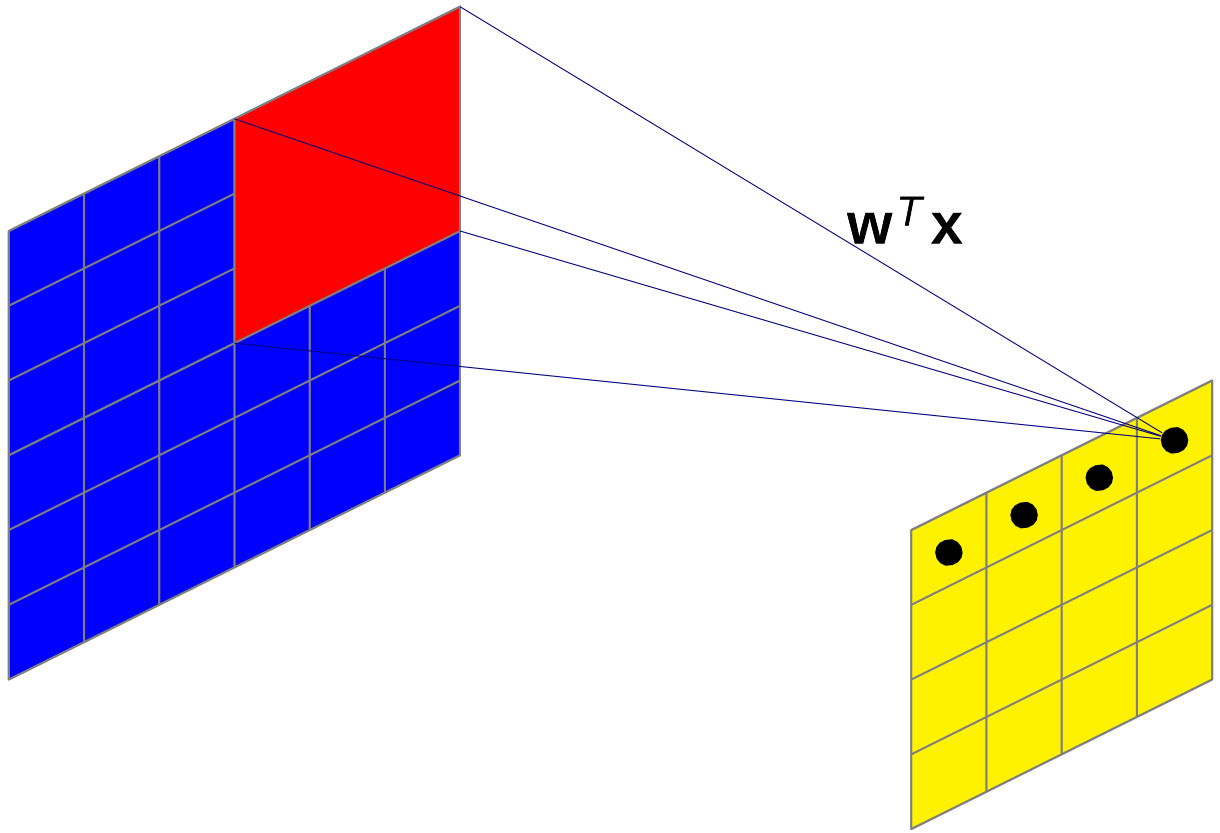
Convolution



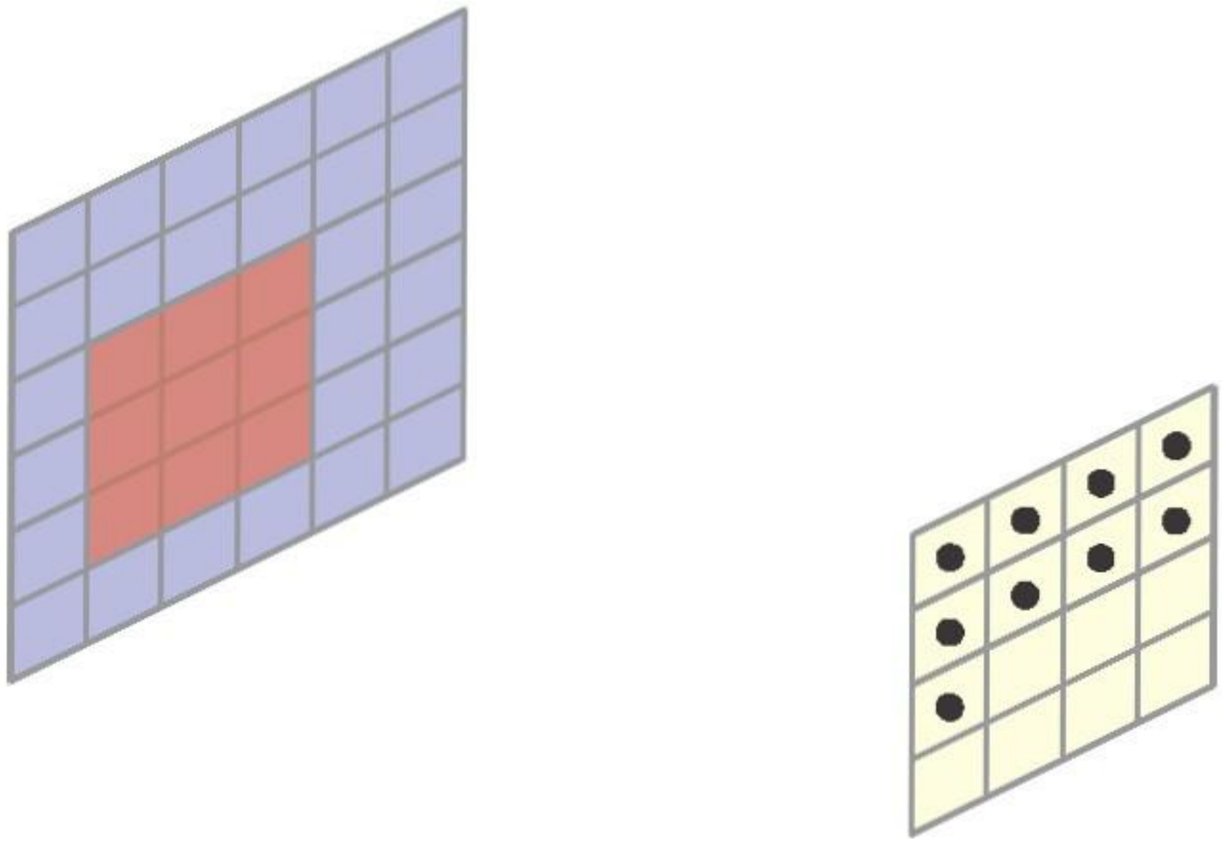
Convolution



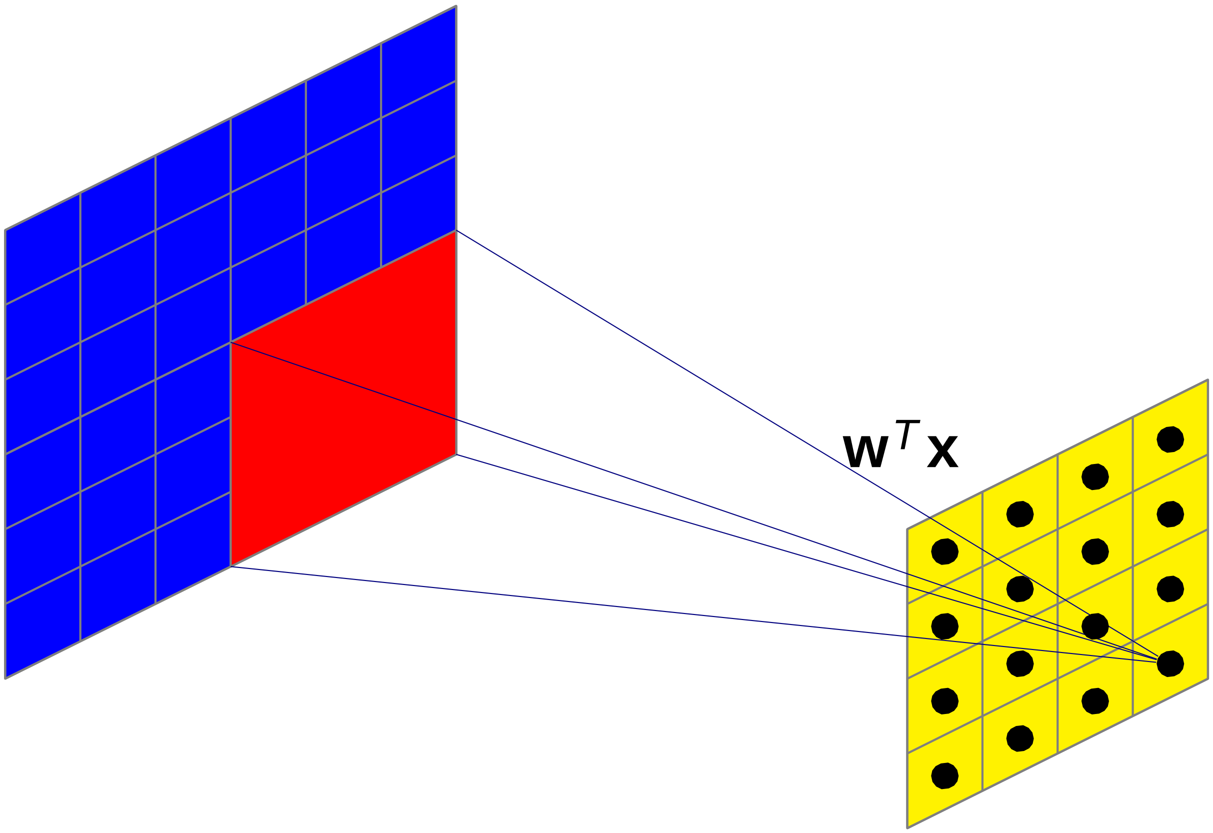
Convolution



Convolution



Convolution



- What is the number of parameters?

Output Size

- We used stride of 1, kernel with receptive field of size 3 by 3
- Output size:

$$\frac{N-K}{S} + 1$$

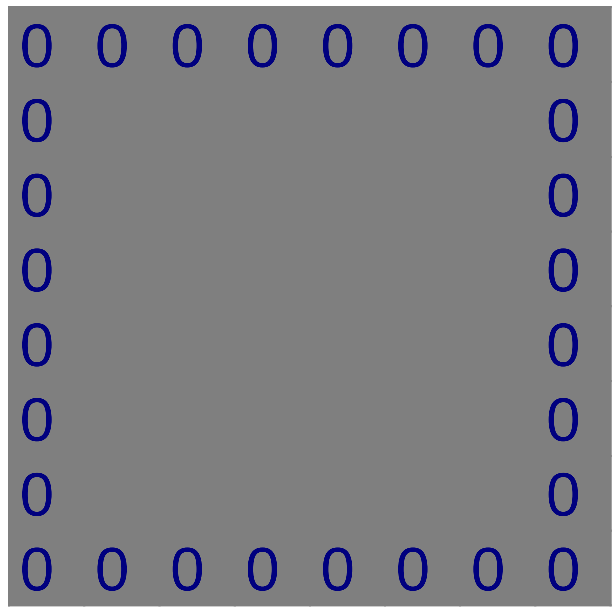
In previous example: $N = 6$, $K = 3$, $S = 1$, Output size = 4

For $N = 8$, $K = 3$, $S = 1$, output size is 6

Zero Padding

Often, we want the output of a convolution to have the same size as the input. Solution: Zero padding.

In our previous example:



Common to see convolution layers with stride of 1, filters of size K, and zero padding with $\frac{K-1}{2}$ to preserve size

In practice

We have only considered a 2-D image as a running example

But we could operate on volumes (e.g. RGB Images would be depth 3 input, filter would have same depth)

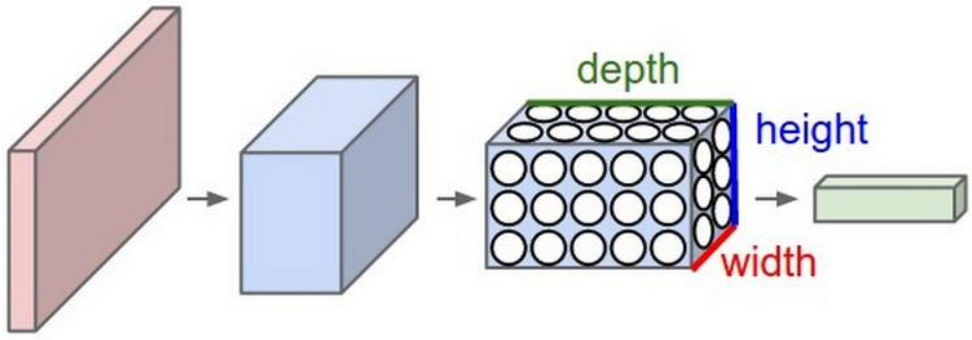


Image from Wikipedia

Output Size

For convolutional layer:

- Suppose input is of size $W_1 \times H_1 \times D_1$
- Filter size is K and stride S
- We obtain another volume of dimensions $W_2 \times H_2 \times D_2$
- As before:

$$W_2 = \frac{W_1 - K}{S} + 1 \text{ and } H_2 = \frac{H_1 - K}{S} + 1$$

- Depths will be equal

Convolutional Layer Parameters

Example volume: $28 \times 28 \times 3$ (RGB Image)

100 3×3 filters, stride 1

What is the zero padding needed to preserve size?

Number of parameters in this layer?

For every filter: $3 \times 3 \times 3 + 1 = 28$ parameters

Total parameters: $100 \times 28 = 2800$

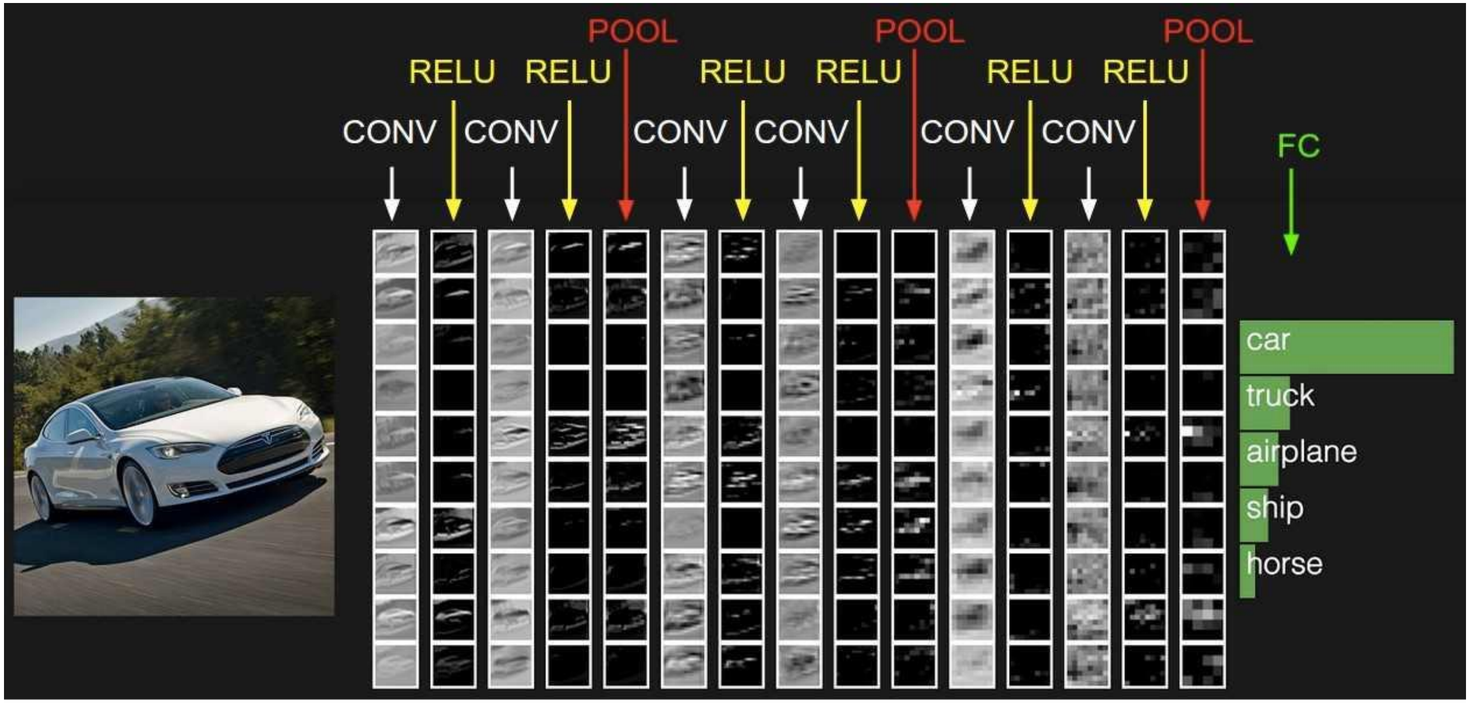
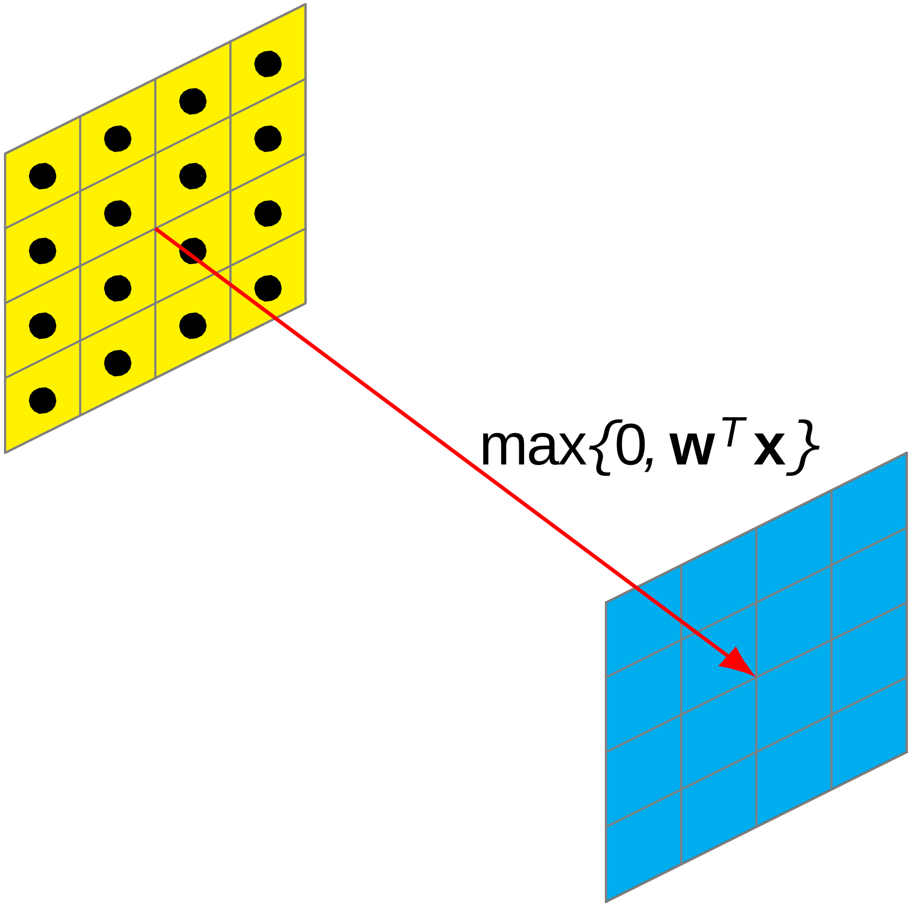


Figure: Andrej Karpathy

Non-Linearity



- After obtaining feature map, apply an elementwise non-linearity to obtain a transformed feature map (same size)

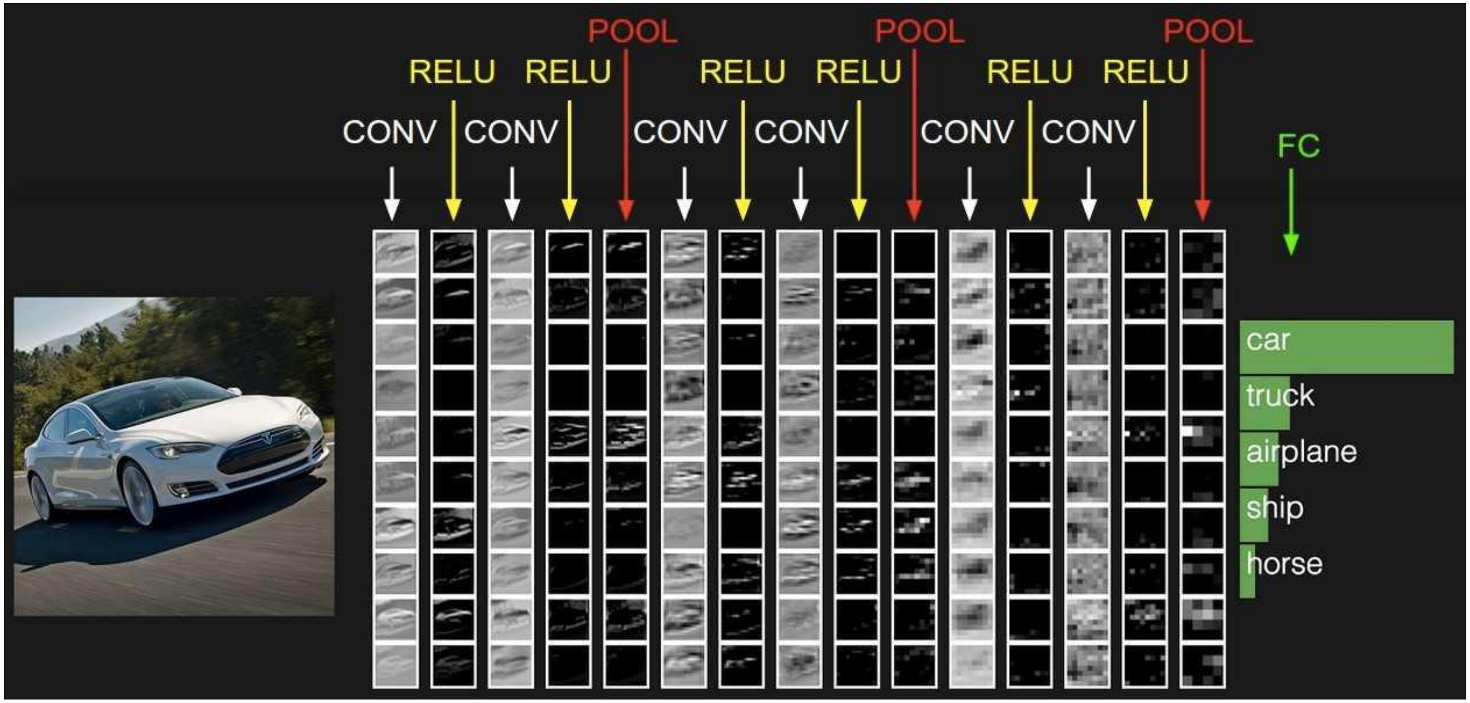
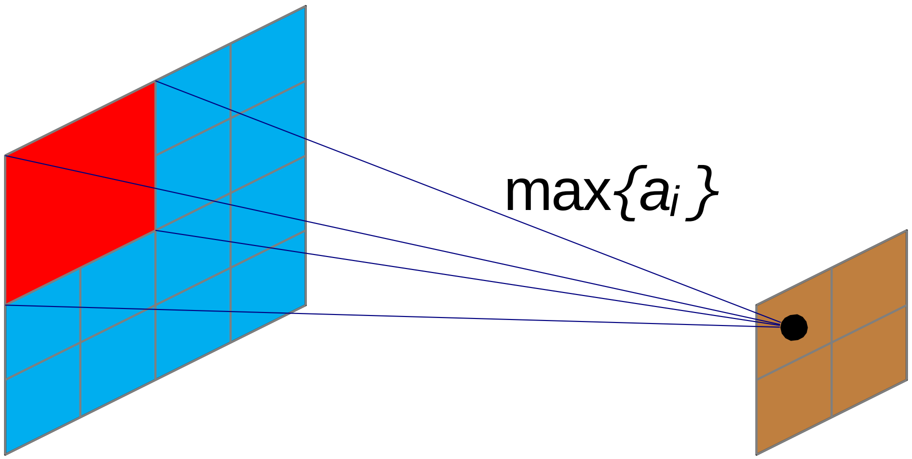
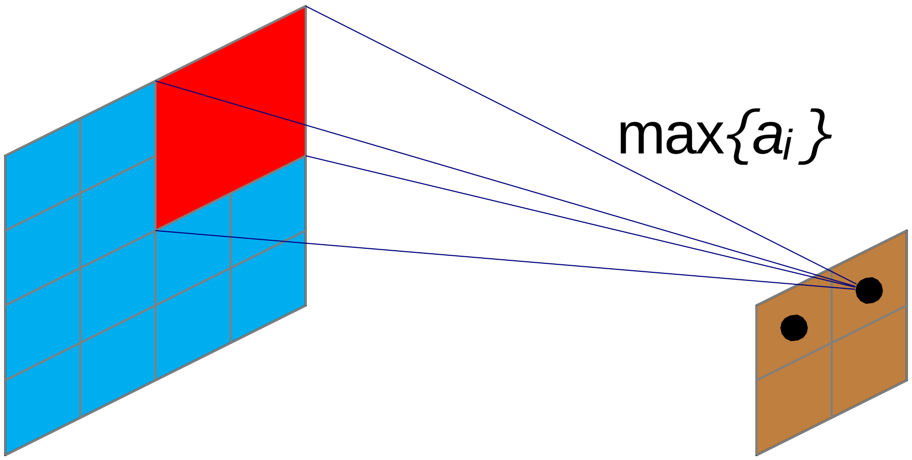


Figure: Andrej Karpathy

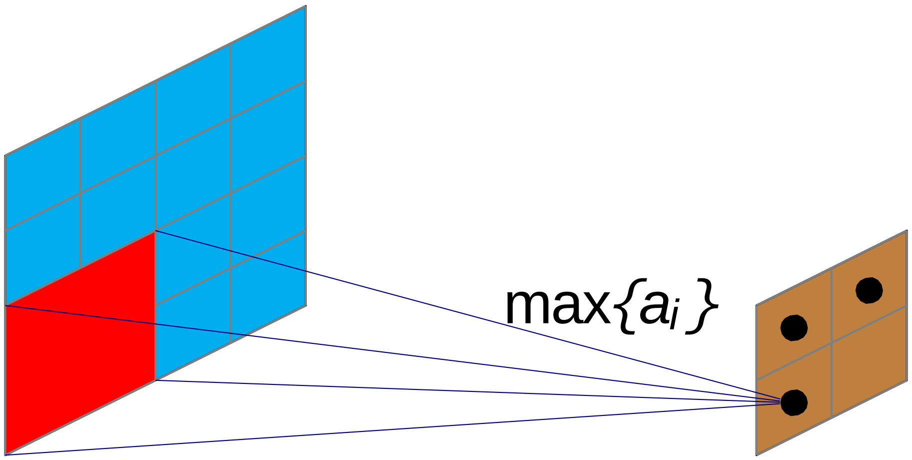
Pooling



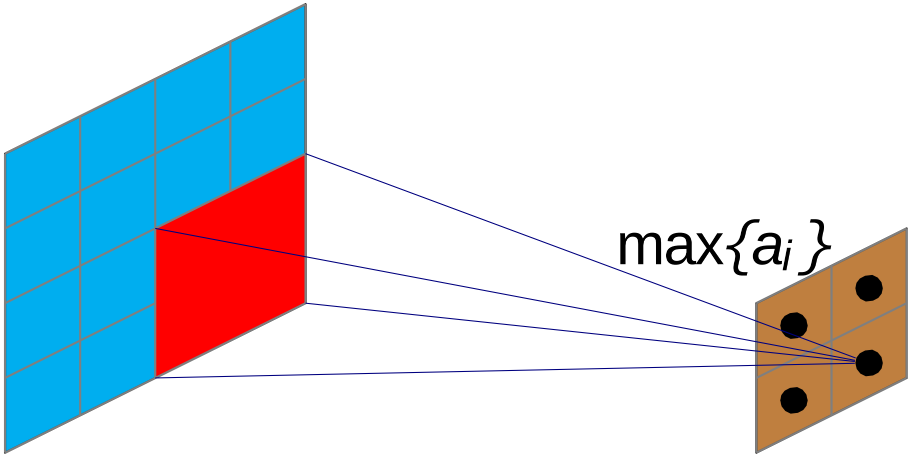
Pooling



Pooling

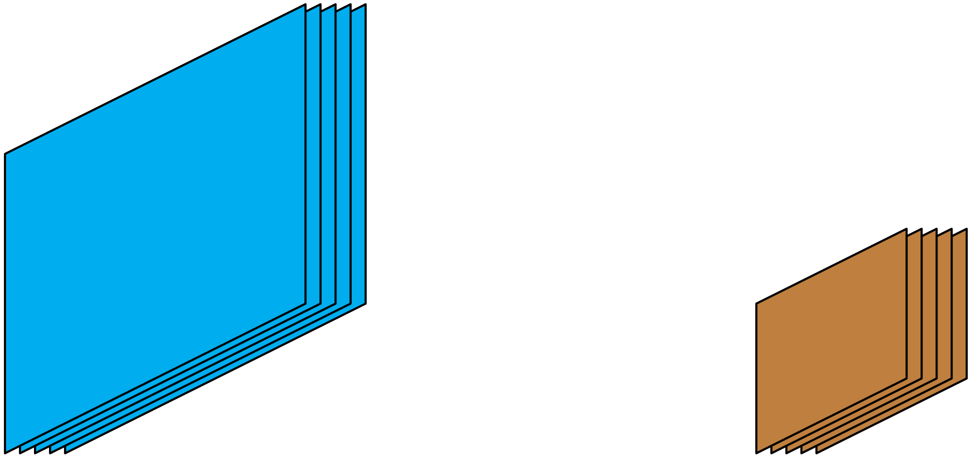


Pooling



- Other options: Average pooling, L2-norm pooling, random pooling

Pooling

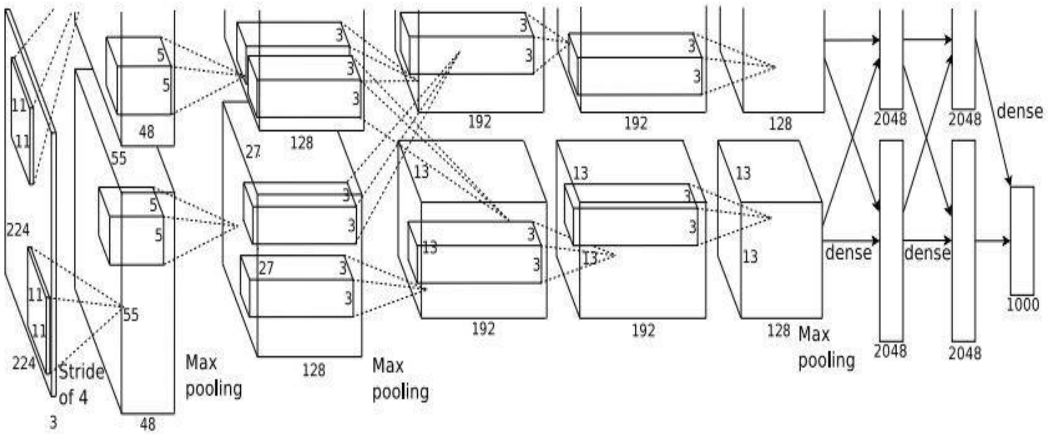


- We have multiple feature maps, and get an equal number of subsampled maps
- This changes if cross channel pooling is done

AlexNet example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

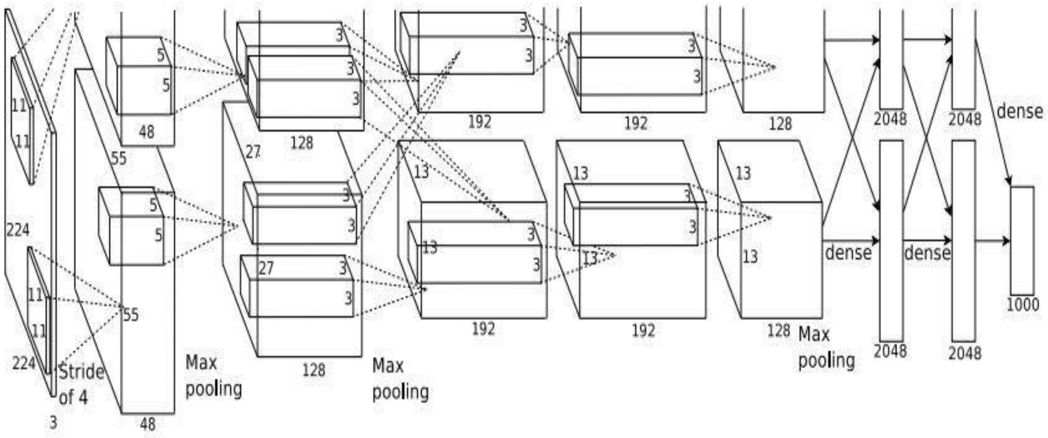
=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

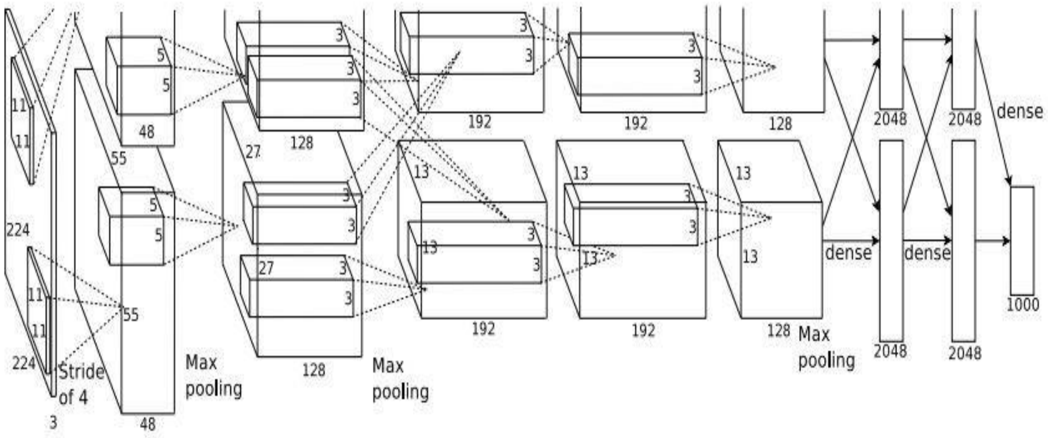
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]

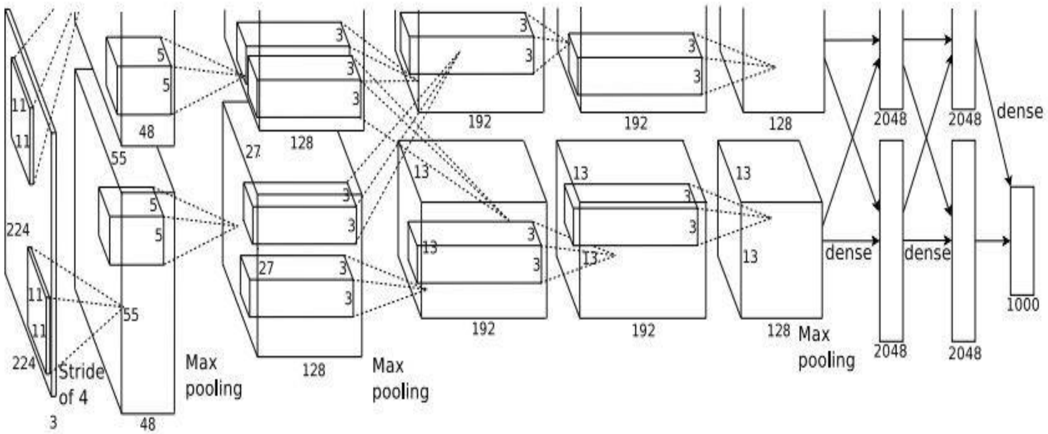


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

[Krizhevsky et al. 2012]



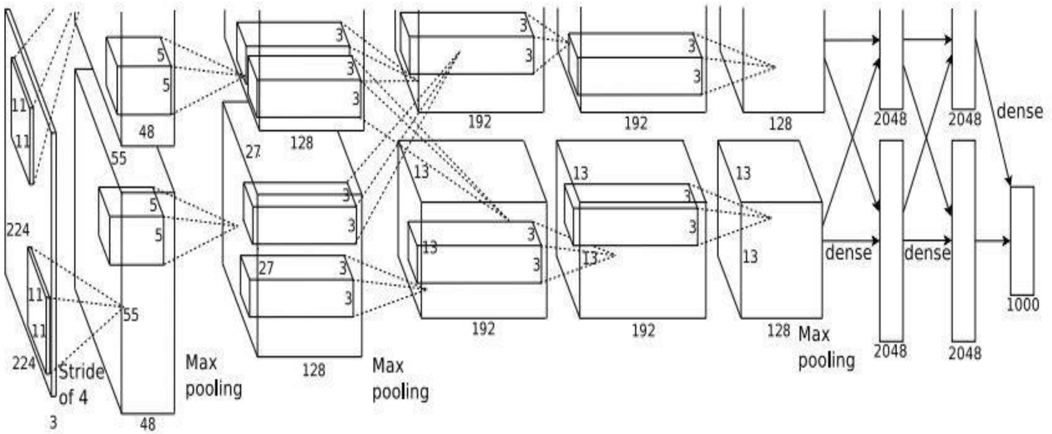
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet

[Krizhevsky et al. 2012]



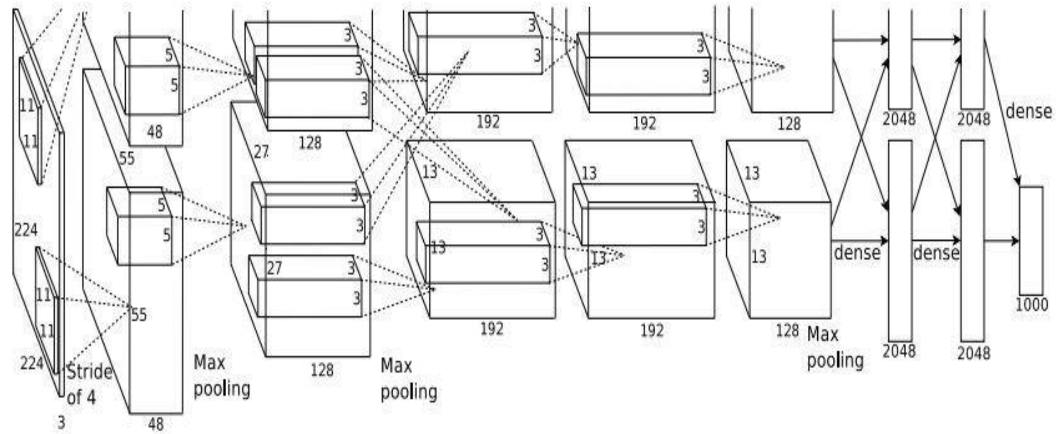
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

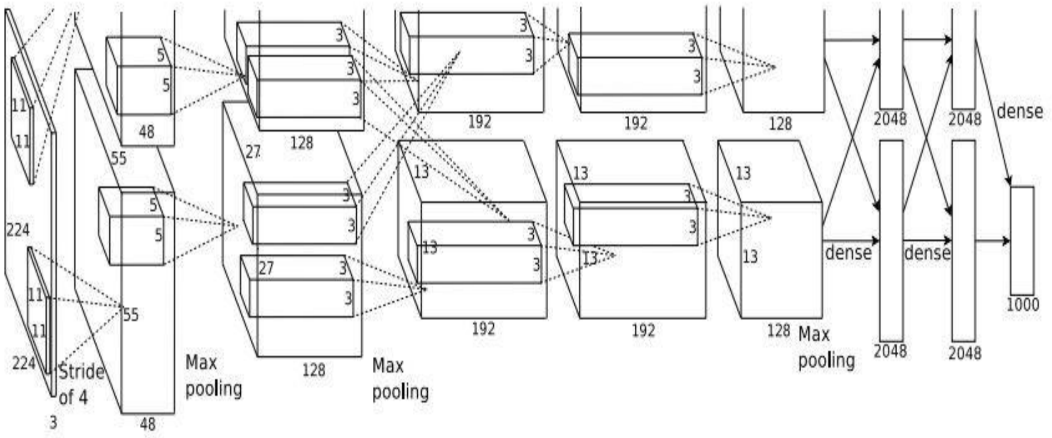


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

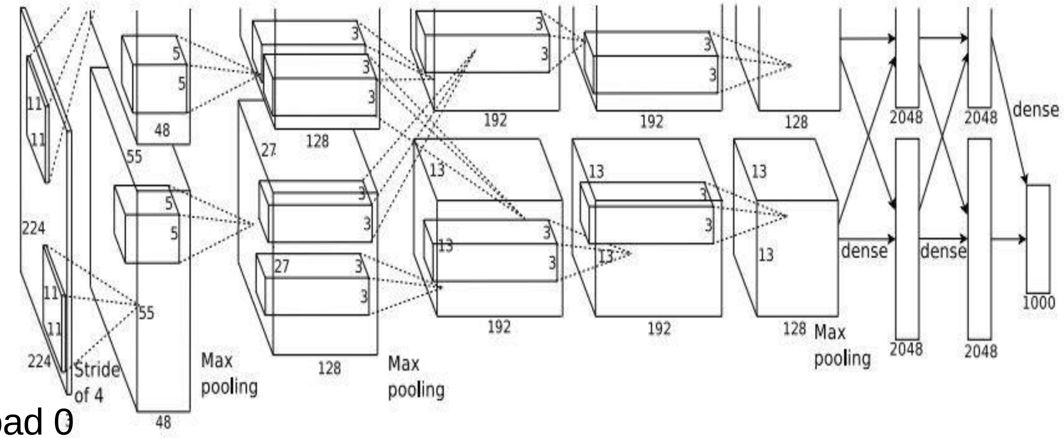
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Case Study: AlexNet

[Krizhevsky et al. 2012]

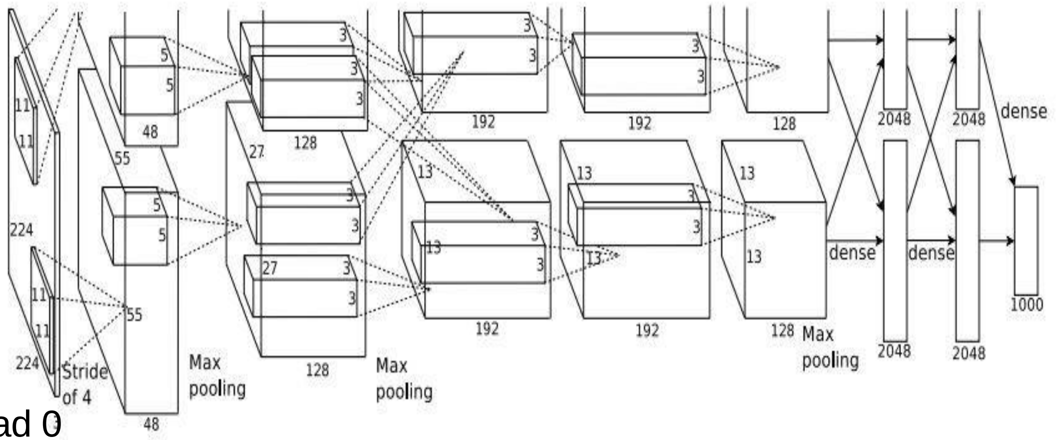


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
-first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
-Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

slide from: Fei-Fei Li & Andrej Karpathy & Justin Johnson