

Introduction to Computer Vision

13. Deep Learning

UCLA – CS 188 – Fall 2019

Fabien Scalzo, Ph.D.

Week 1			26-Sep	Introduction
Week 2	1-Oct	Basic Image Processing	3-Oct	Feature Extraction and Classification
Week 3	8-Oct	Feature Tracking/Optical Flow	10-Oct	SVD, 2D camera model, projective plane
Week 4	15-Oct	2D Image transformations, RANSAC	17-Oct	Euclidean geometry, rigid body motion
Week 5	22-Oct	Epipolar Geometry	24-Oct	3D Cameras and processing
Week 6	29-Oct	Midterm	31-Oct	3D Cameras and processing
Week 7	5-Nov	Learning from data	7-Nov	Neural Networks
Week 8	12-Nov	Deep Learning	14-Nov	Deep Learning
Week 9	19-Nov	Object Detection	21-Nov	Generative Models
Week 10	26-Nov	Guest Lecture (Nikhil Naik)	28-Nov	
Week 11	3-Dec	Applications	5-Dec	Recap
Week 12	10-Dec		12-Dec	Final

-
- Convolution
 - Back-propagation
 - Limitations of Gradient Descent
 - Limitations of Neural Networks in Computer Vision
 - Deep Learning
 - Convolutional Neural Networks
 - Convolutional Encoder-decoder
-

2D Convolution

$$I'(x, y) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(x - i, y - j) \cdot filter(i, j)$$

The diagram illustrates the computation of a 2D convolution. On the left is a 3x3 input matrix:

2	3	1
0	5	1
1	0	8

On the right is a 3x3 filter matrix:

-1	0	1
0	0	0
0	-1	0

A blue asterisk (*) symbol indicates the convolution operation between the input and the filter. Above the filter, the dimensions m and n are labeled, indicating the filter's height and width respectively. To the right of the filter is an equals sign (=) followed by a question mark (?), representing the result of the convolution.

2D Convolution

$$I'(x, y) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(x - i, y - j) \cdot filter(i, j)$$

The diagram illustrates the computation of a 2D convolution. On the left is a 3x3 input matrix:

2	3	1
0	5	1
1	0	8

On the right is a 3x3 filter matrix:

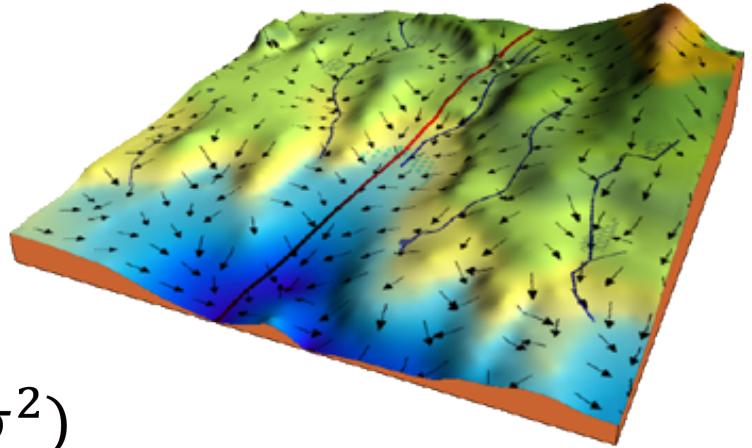
-1	0	1
0	-1	0
-1	0	-1

A blue asterisk (*) symbol indicates the convolution operation between the input and the filter. Above the filter, the dimensions m and n are labeled diagonally, indicating the filter's height and width respectively. To the right of the filter is an equals sign (=) followed by a question mark (?), representing the result of the convolution.

Gradient Descent

Algorithm

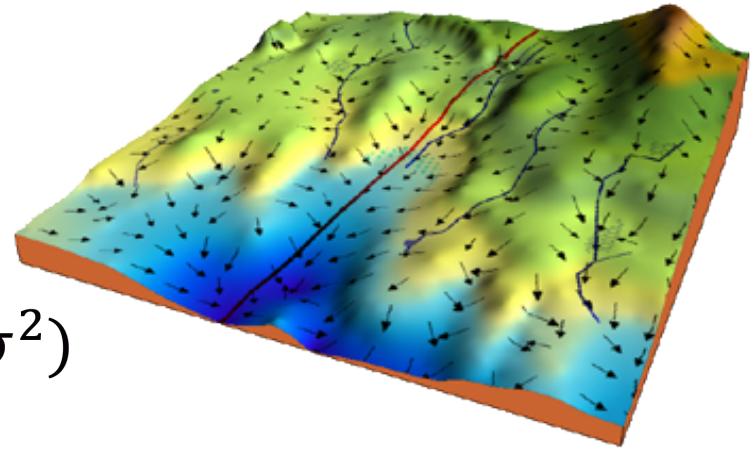
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
 - Propagate input forward
 - Backpropagate
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights



Chain rule

If $h(x)$ is a composite function defined by $h(x) = g(f(x))$, and f and g are differentiable, then $h(x)$ is differentiable and h' is given by the product:

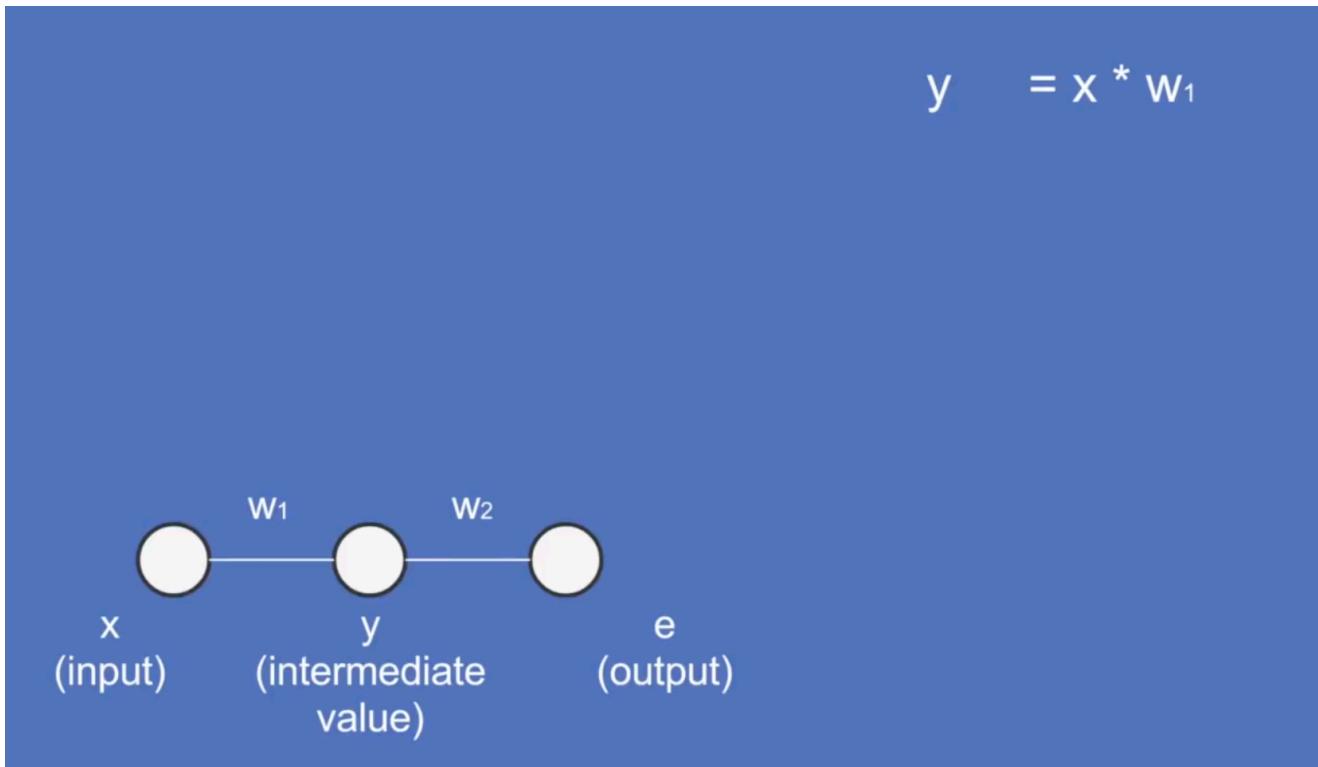
$$(h(x))' = (g(f(x)))' = g'(f(x)) \cdot f'(x)$$

derive the outside,
keep the inside

derive the
inside

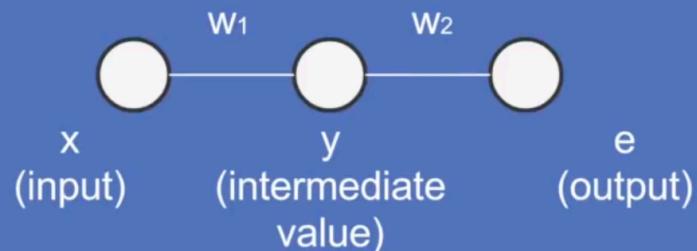
$$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x}$$

Chain rule



Chain rule

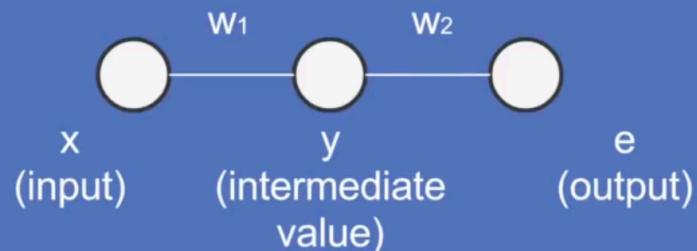
$$\begin{aligned}y &= x * w_1 \\ \frac{\partial y}{\partial w_1} &= x\end{aligned}$$



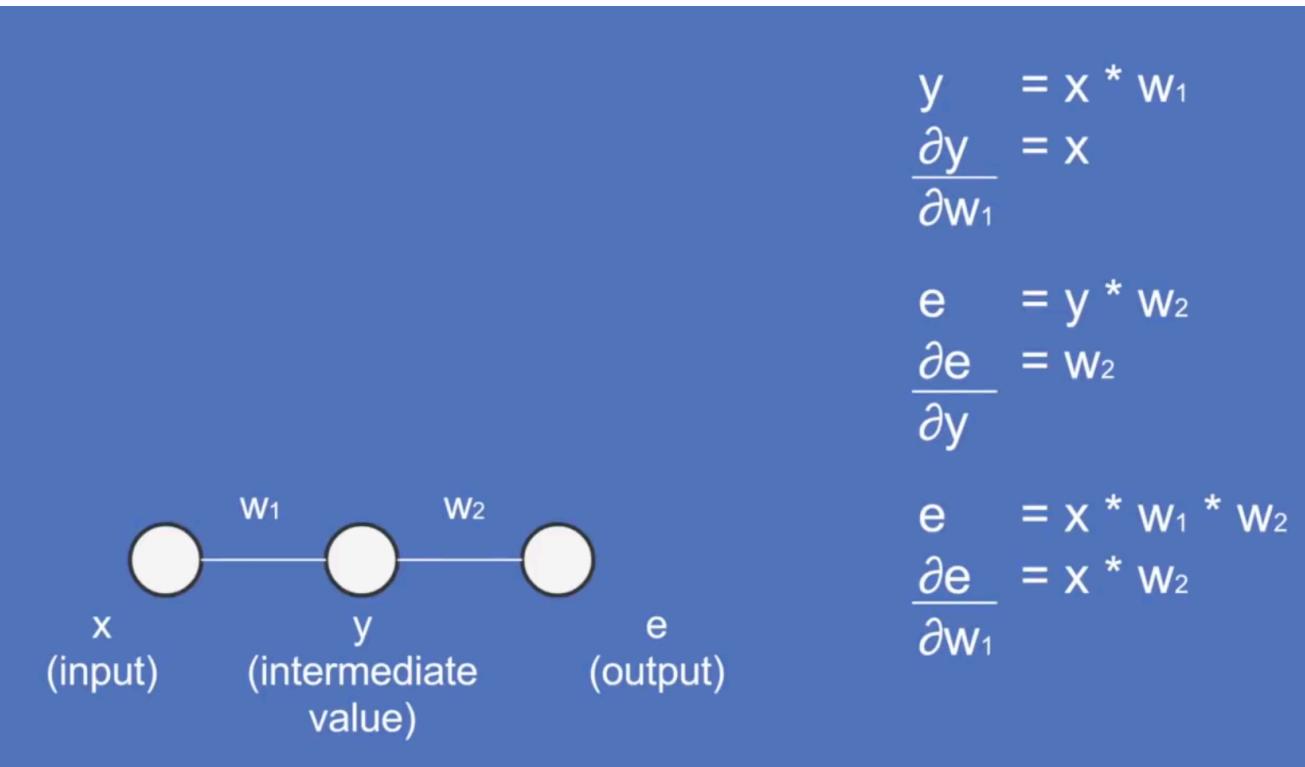
Chain rule

$$\begin{aligned}y &= x * w_1 \\ \frac{\partial y}{\partial w_1} &= x\end{aligned}$$

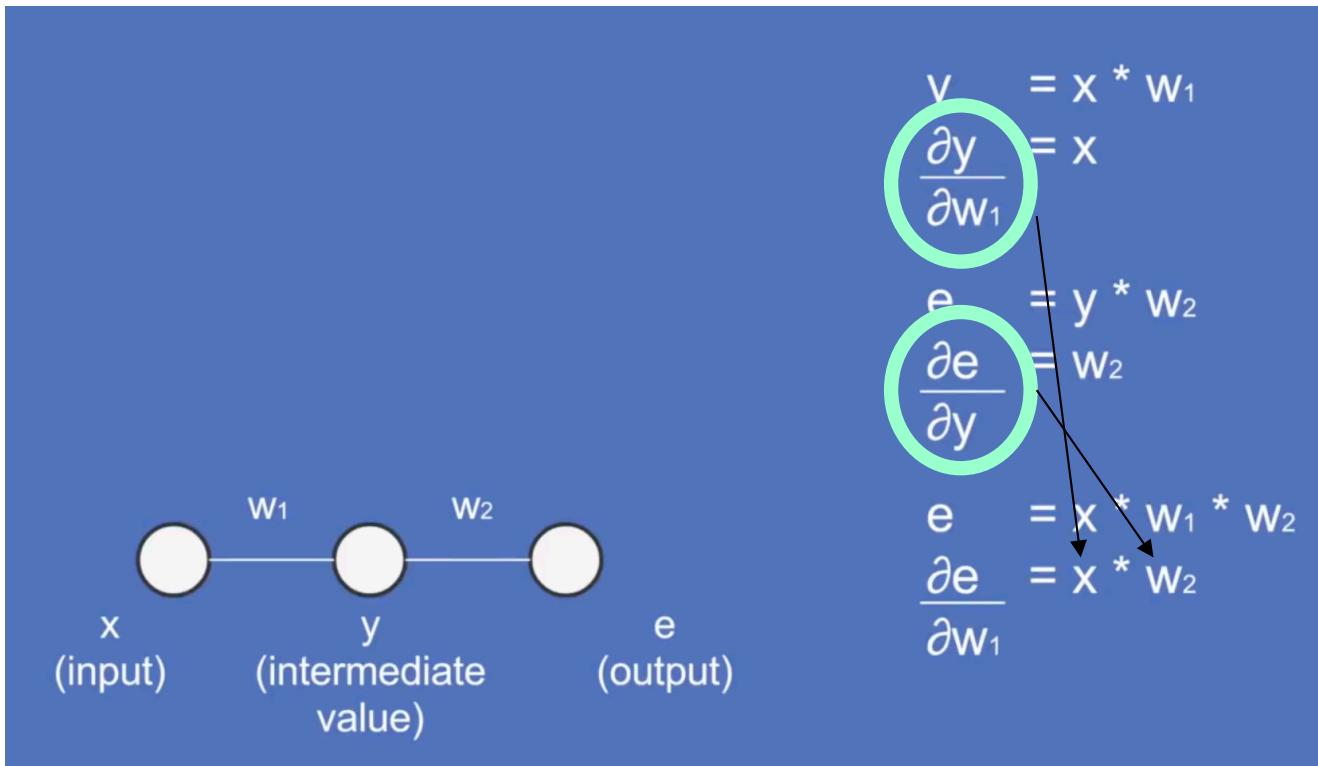
$$\begin{aligned}e &= y * w_2 \\ \frac{\partial e}{\partial y} &= w_2\end{aligned}$$



Chain rule



Chain rule



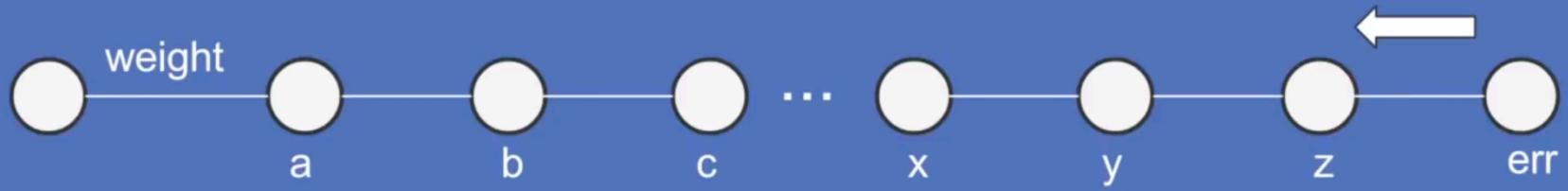
Chain rule

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial w_1}$$
$$y = x * w_1$$
$$\frac{\partial y}{\partial w_1} = x$$
$$e = y * w_2$$
$$\frac{\partial e}{\partial y} = w_2$$
$$e = x * w_1 * w_2$$
$$\frac{\partial e}{\partial w_1} = x * w_2$$

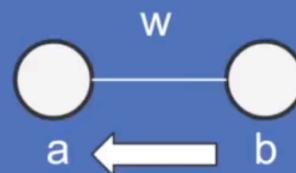
The diagram illustrates the chain rule for backpropagation through a simple neural network. The network consists of three nodes: an input node labeled 'x (input)', an intermediate node labeled 'y (intermediate value)', and an output node labeled 'e (output)'. Two weights, w_1 and w_2 , connect the nodes: w_1 connects 'x' to 'y', and w_2 connects 'y' to 'e'. To the right, the forward pass calculations are shown: $y = x * w_1$ and $e = y * w_2$. The partial derivatives are also calculated: $\frac{\partial y}{\partial w_1} = x$ and $\frac{\partial e}{\partial y} = w_2$. The final result of the chain rule application is $\frac{\partial e}{\partial w_1} = x * w_2$. Two arrows point from the intermediate equations to the final result, indicating the flow of gradients.

Chain Rule

$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



Chain rule



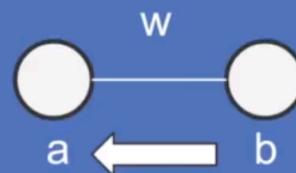
$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

$$b = wa$$



$$\frac{\partial b}{\partial a} =$$

Chain rule



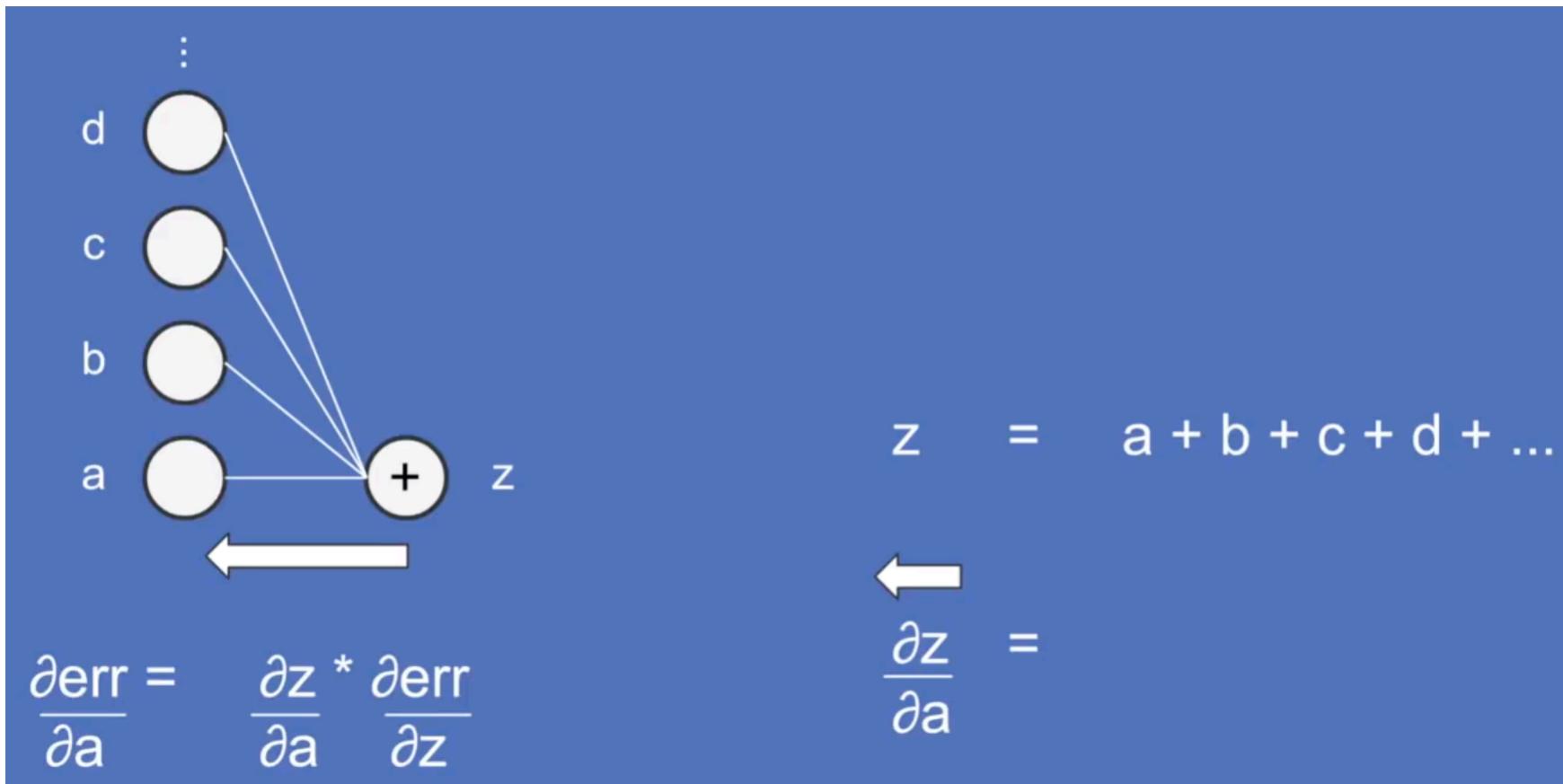
$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

$$b = wa$$



$$\frac{\partial b}{\partial a} = w$$

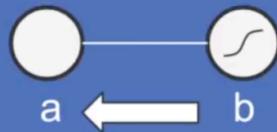
Chain rule



Chain rule

$$\begin{aligned} b &= \frac{1}{1 + e^{-a}} \\ &= \sigma(a) \end{aligned}$$

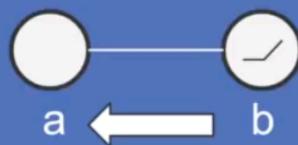
Because math is beautiful /
dumb luck:



$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

$$\frac{\partial b}{\partial a} = \sigma(a) * (1 - \sigma(a))$$

Chain rule



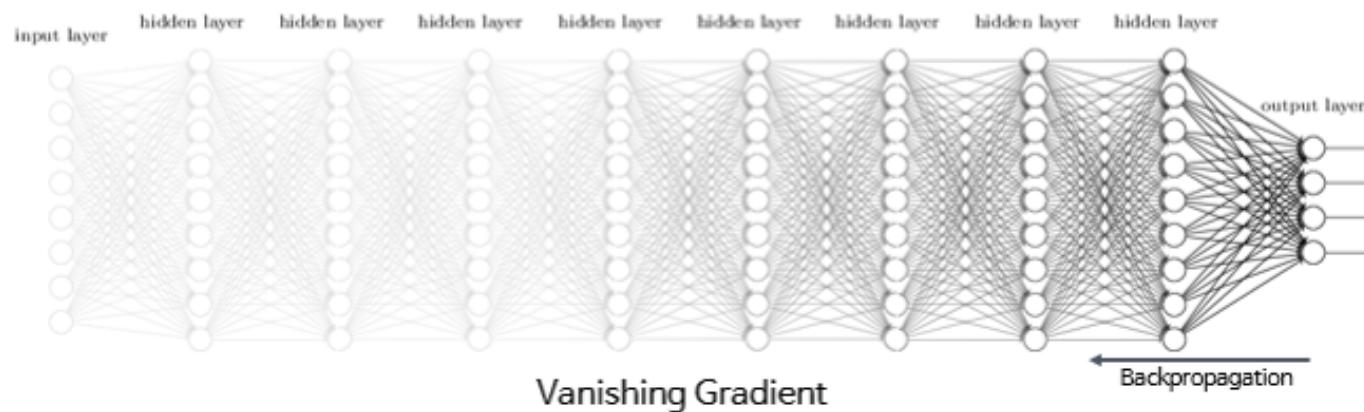
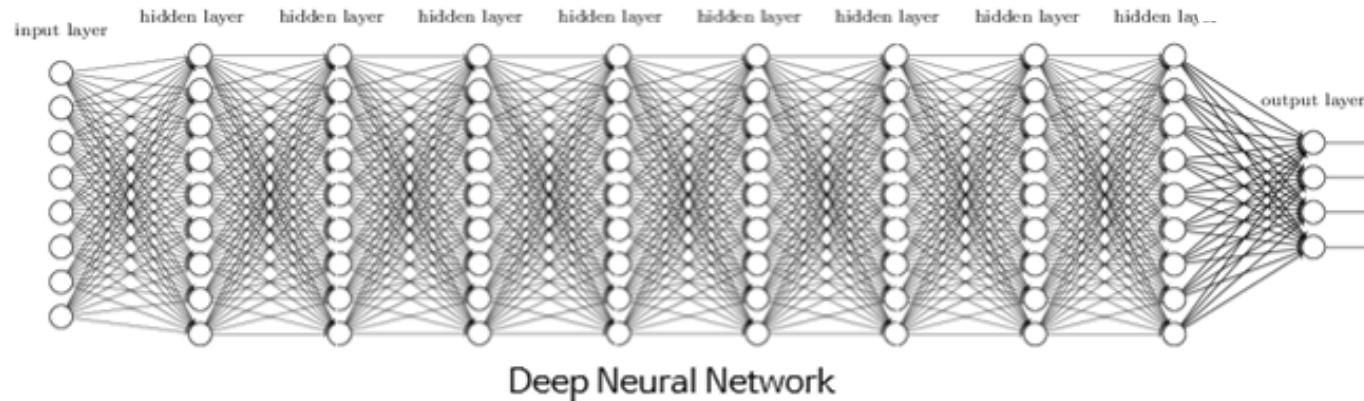
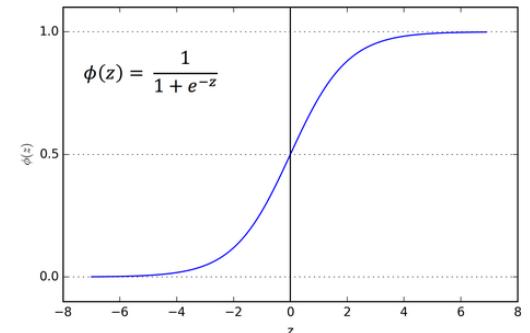
$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

$$\begin{aligned} b &= a, \quad a > 0 \\ &= 0, \quad \text{otherwise} \end{aligned}$$

$$\begin{array}{c} \leftarrow \\ \frac{\partial b}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{otherwise} \end{cases} \end{array}$$

Limitations of Gradient Descent

- Activation function: Sigmoid



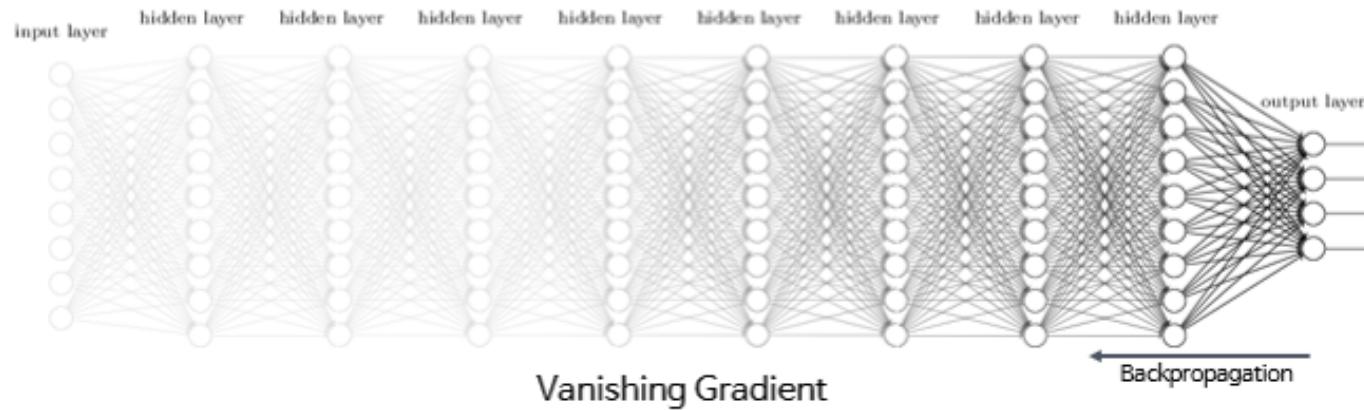
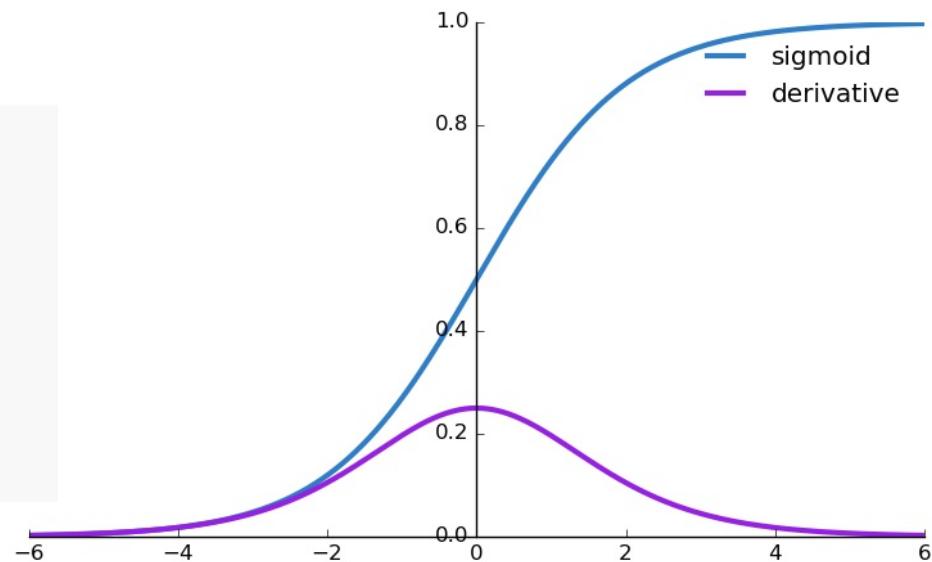
Vanishing gradient

The sigmoid function is defined as follows

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

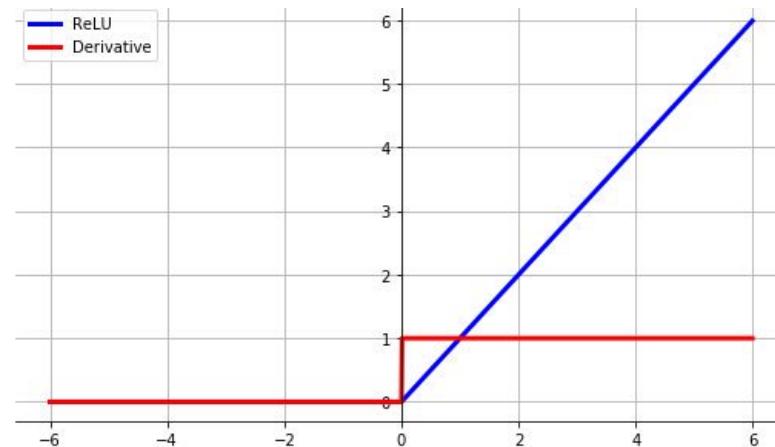
This function is easy to differentiate because

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x)).$$



Vanishing gradient : Solution 1

ReLU activation function



Vanishing gradient : Solution 2

(mini-)batch normalization (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

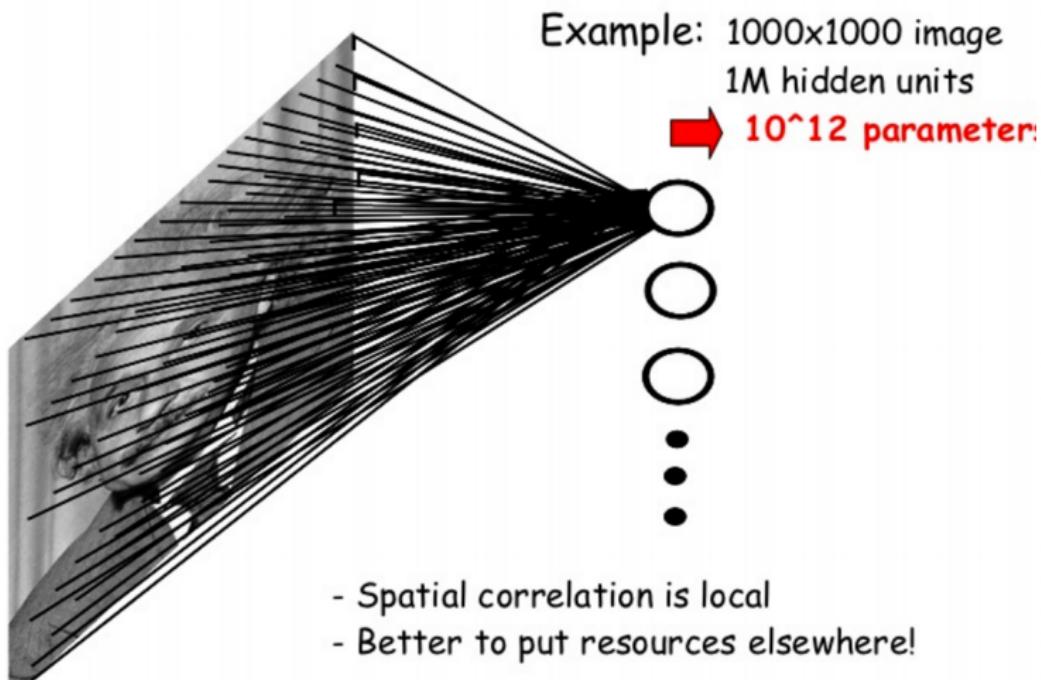
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

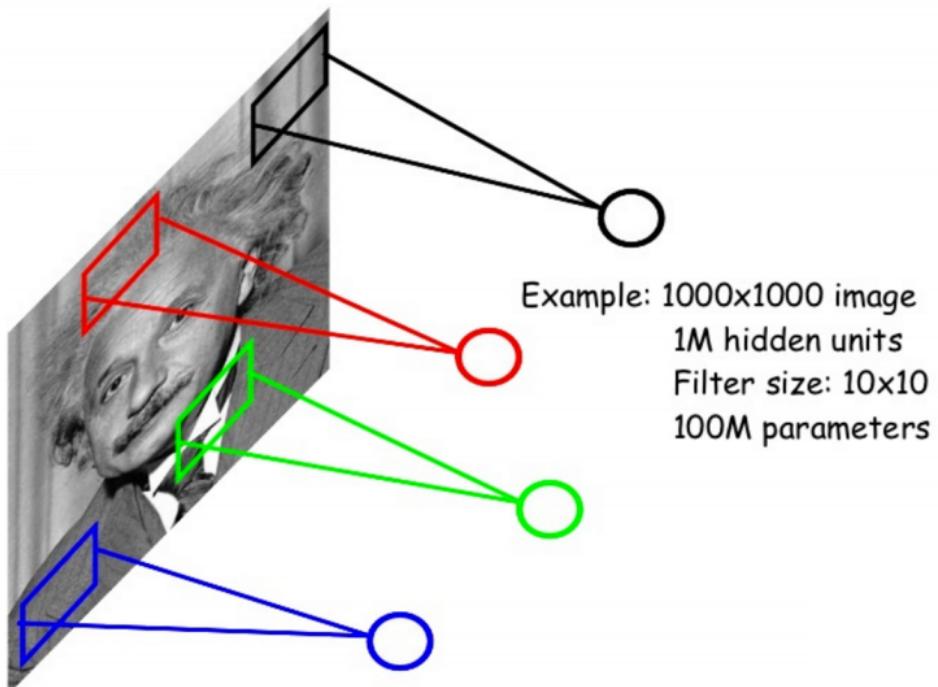
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

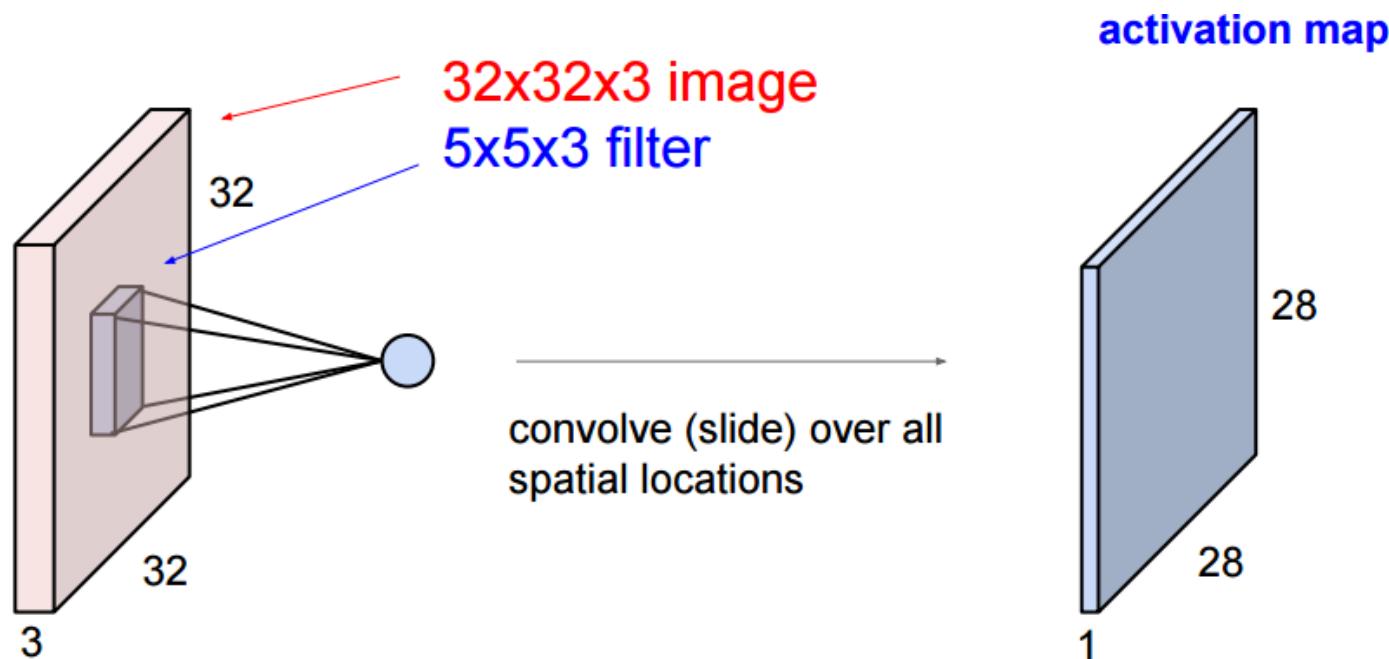
FULLY CONNECTED NEURAL NET



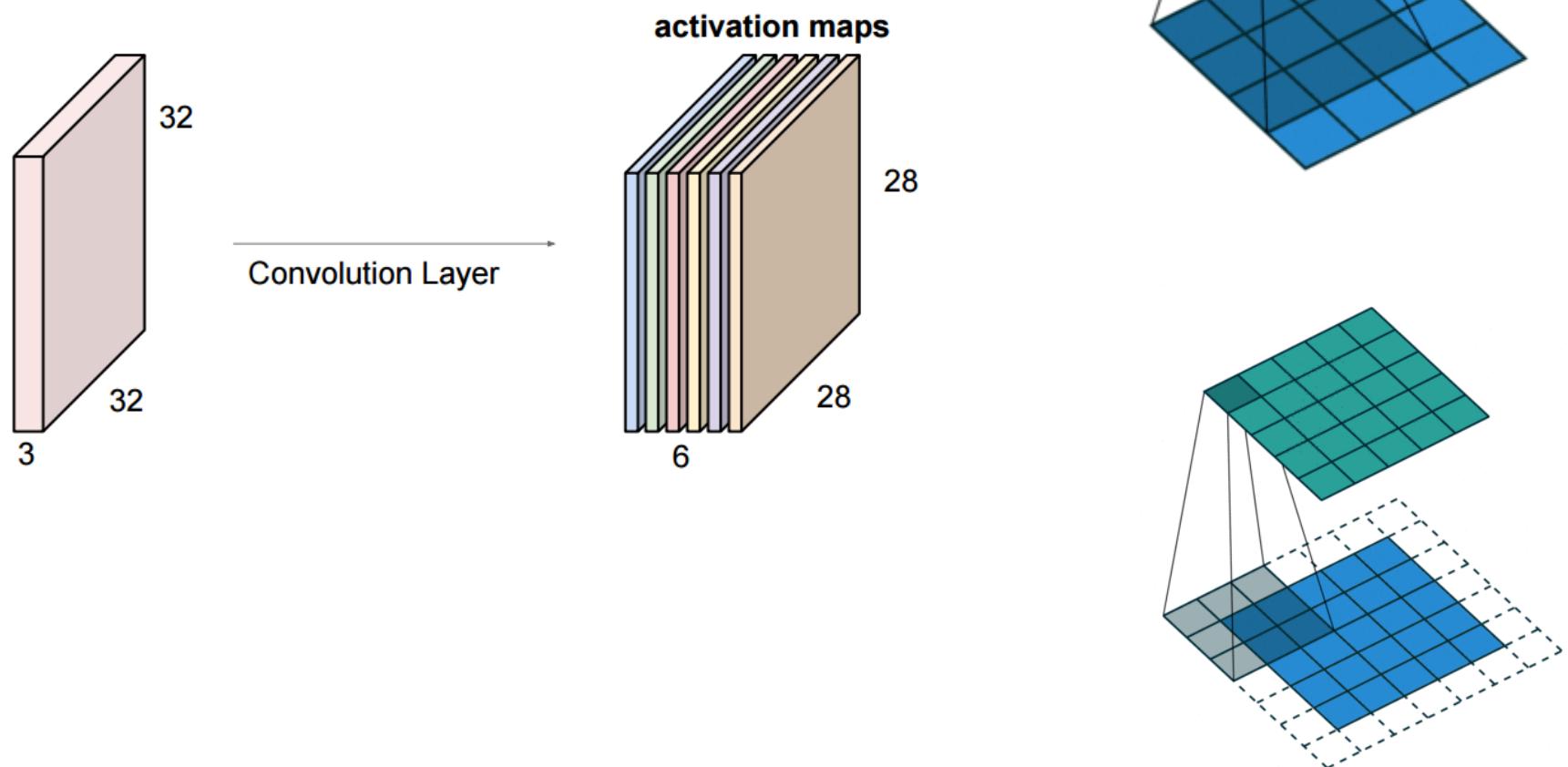
LOCALLY CONNECTED NEURAL NET



Convolution Layer

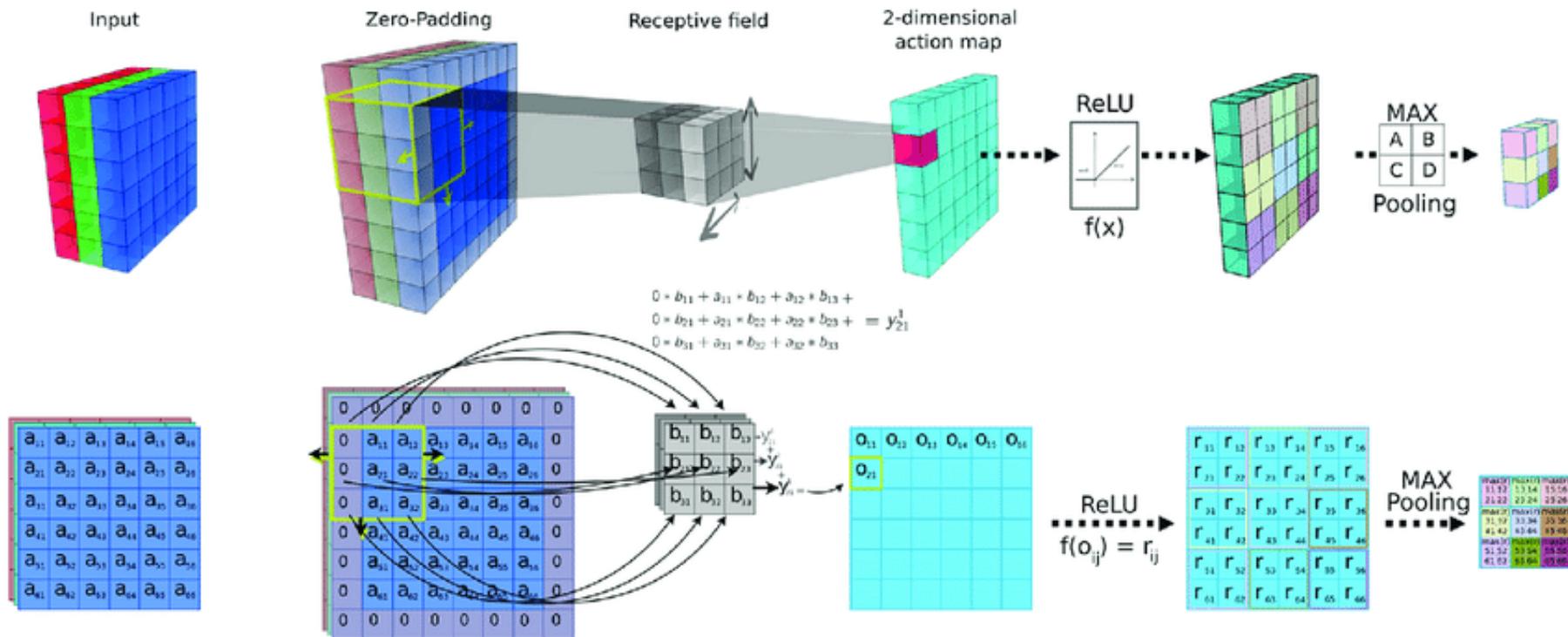


Convolution layer

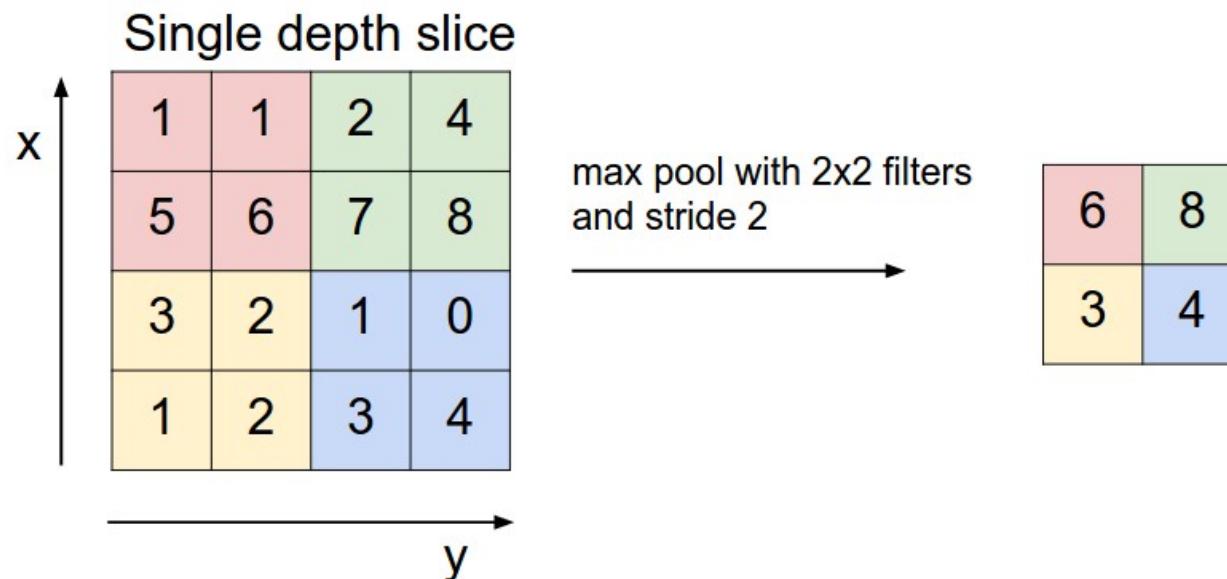


Convolution layer

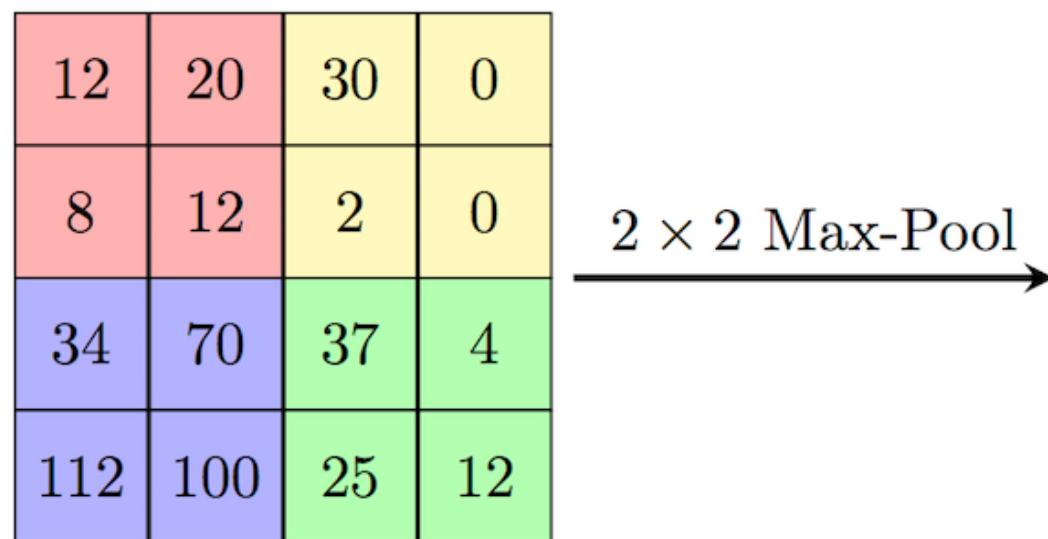
- Parametrized by: height, width, depth, stride, padding, number of filters, type of activation function



Pooling layer (i.e. sub-sampling)



Pooling layer (i.e. sub-sampling)



Dropout layer

- Dropout consists in randomly setting a fraction rate of input units to 0.

Flatten layer

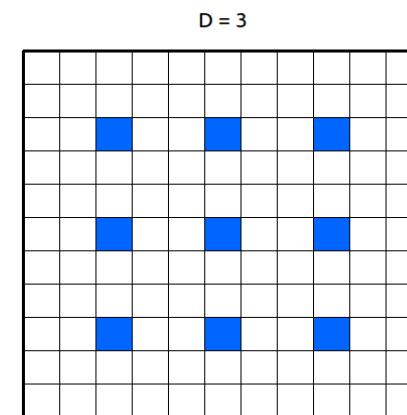
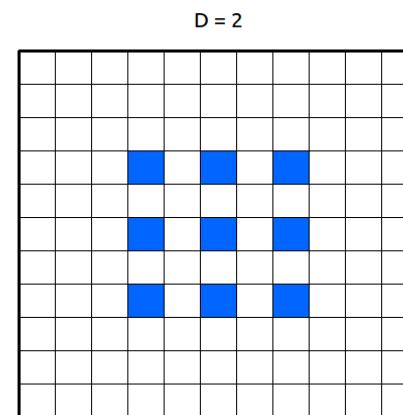
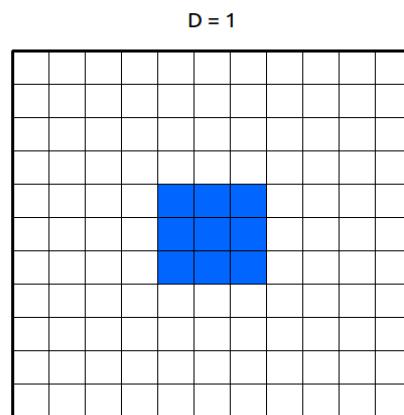
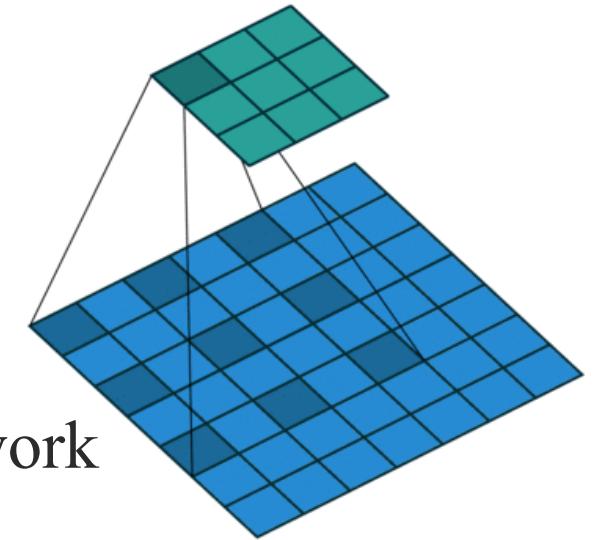
- Convert the input to a 1D array.

Batch Normalization (BN), Noise Layer

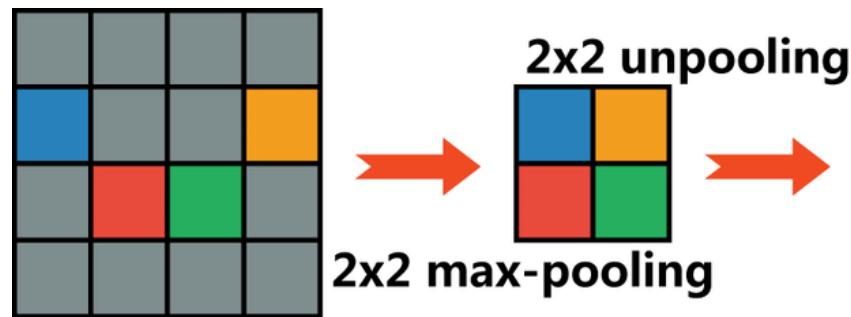
Dilated Convolution

Dilating the filter means expanding its size filling the empty positions with 0.

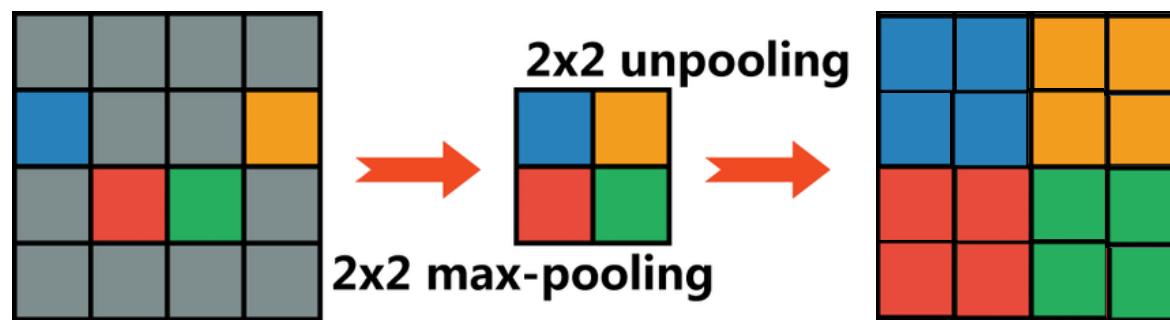
A way of increasing receptive view of the network while keeping the number of weights constant.



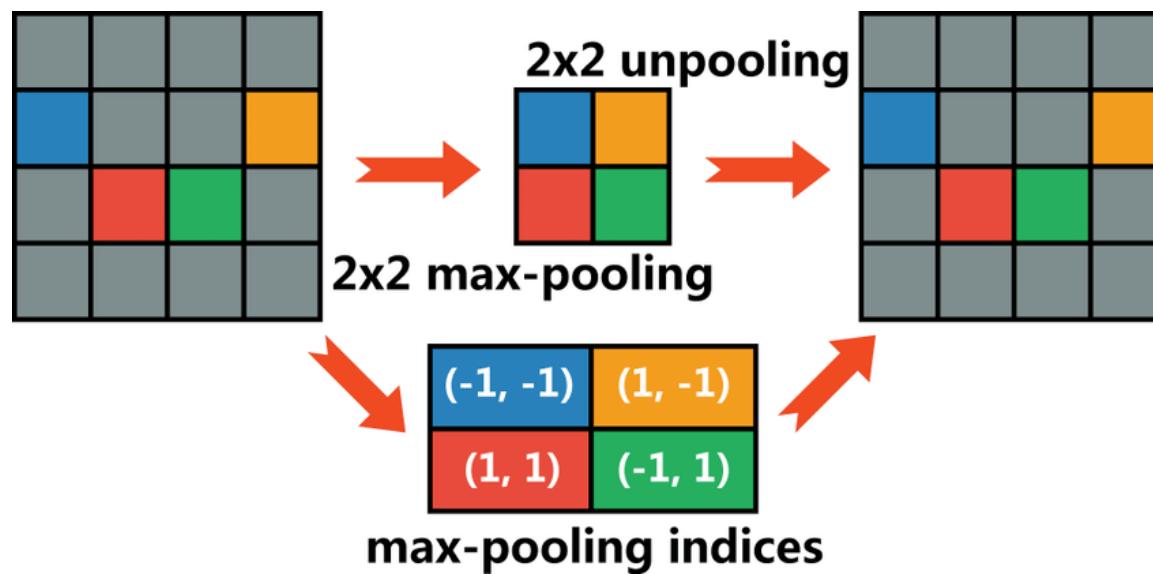
Unpooling



Unpooling (Nearest Neighbor)



Max-Unpooling



Max Pooling

Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Rest of the network

Max Unpooling

Use positions from pooling layer

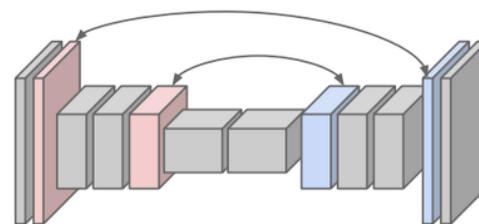
1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

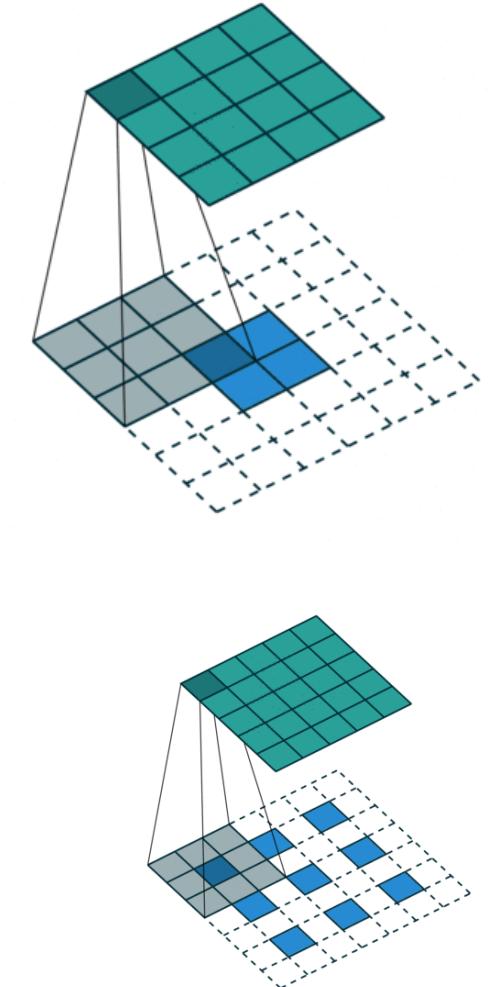
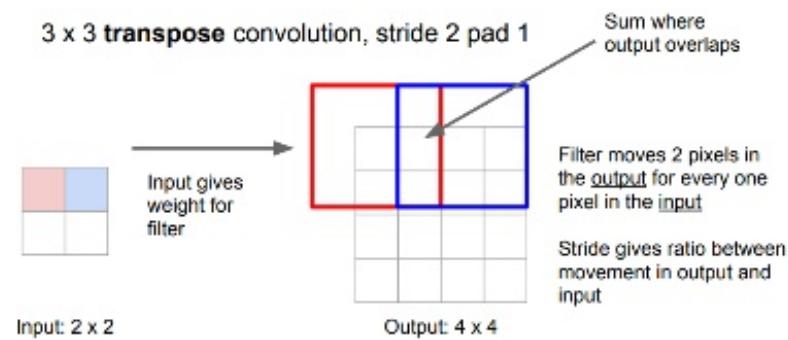
Output: 4 x 4

Corresponding pairs of
downsampling and
upsampling layers



Learned Up-Sampling (Deconvolution)

- Also called transposed convolution



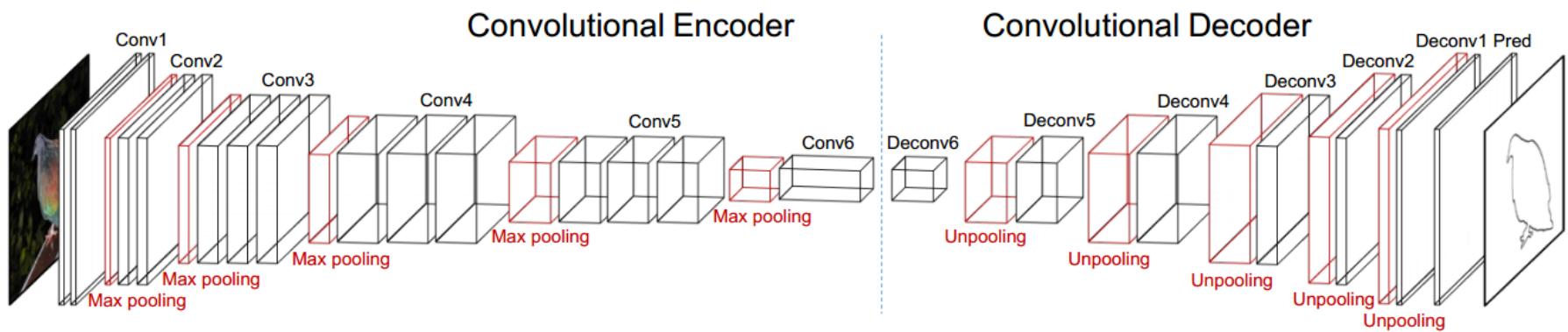
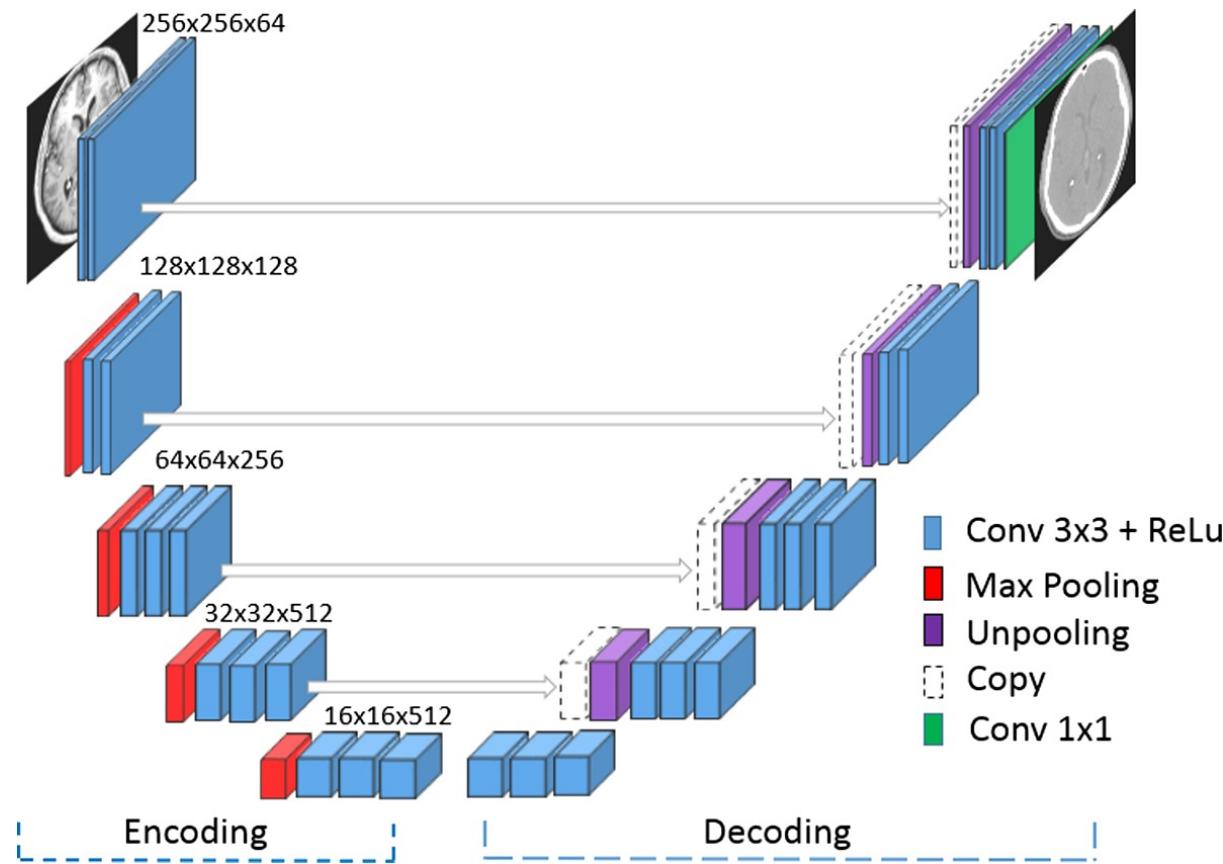


Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

MR-based synthetic CT generation using a deep convolutional neural network method



Medical Physics, Volume: 44, Issue: 4, Pages: 1408-1419, First published: 13 February 2017, DOI: (10.1002/mp.12155)

