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

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

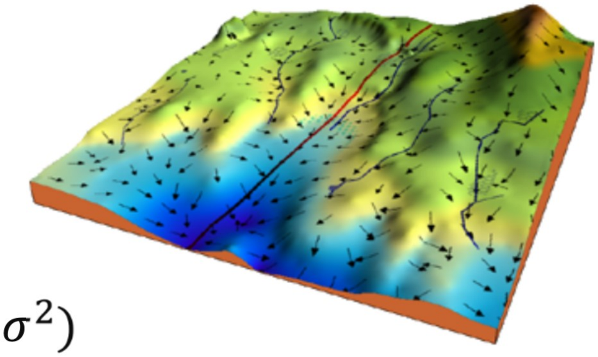
$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & 5 & 1 \\ 1 & 0 & 8 \end{bmatrix} * \begin{array}{c|ccc} {}_{n}\backslash^{m} & -1 & 0 & 1 \\ \hline -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 \end{array} = ?$$

2D Convolution

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

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & 5 & 1 \\ 1 & 0 & 8 \end{bmatrix} * \begin{array}{c c} {}_{n}\diagdown^{m} & \begin{array}{ccc} -1 & 0 & 1 \end{array} \\ \begin{array}{c} -1 \\ 0 \\ 1 \end{array} & \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \end{array} = ?$$

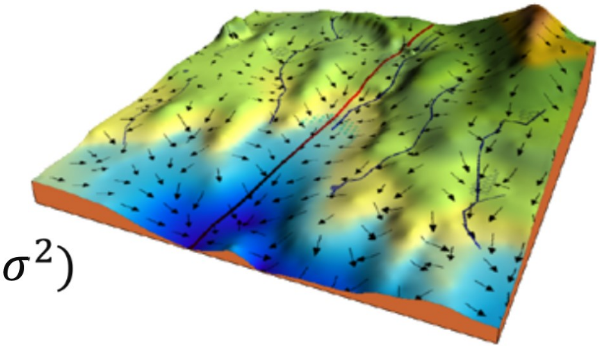
Gradient Descent



Algorithm

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

Gradient Descent



Algorithm

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

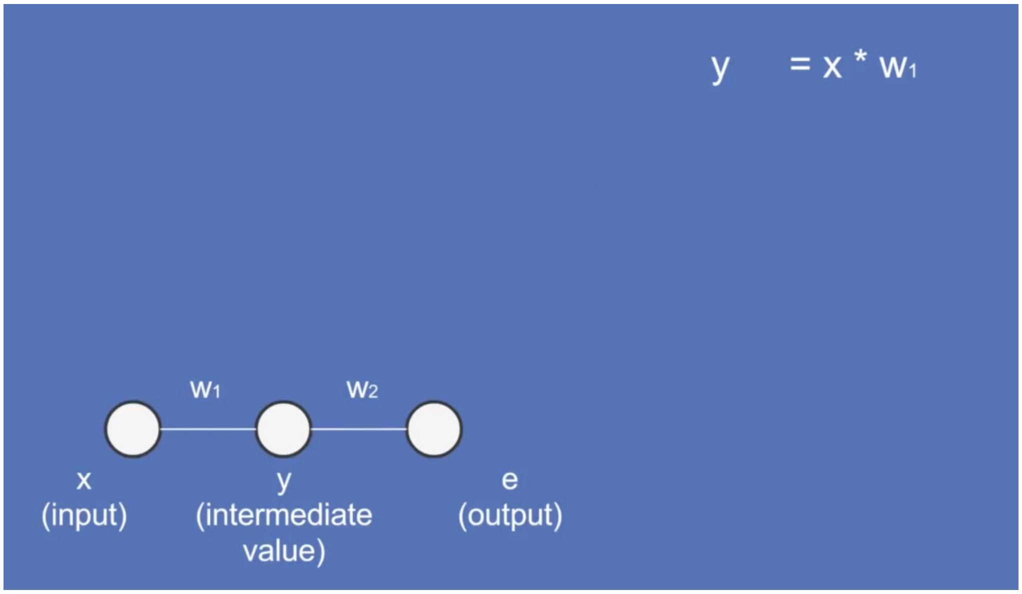
$$(h(x))' = (g(f(x)))' = g'(f(x)) \cdot f'(x)$$

derive the outside, keep the inside

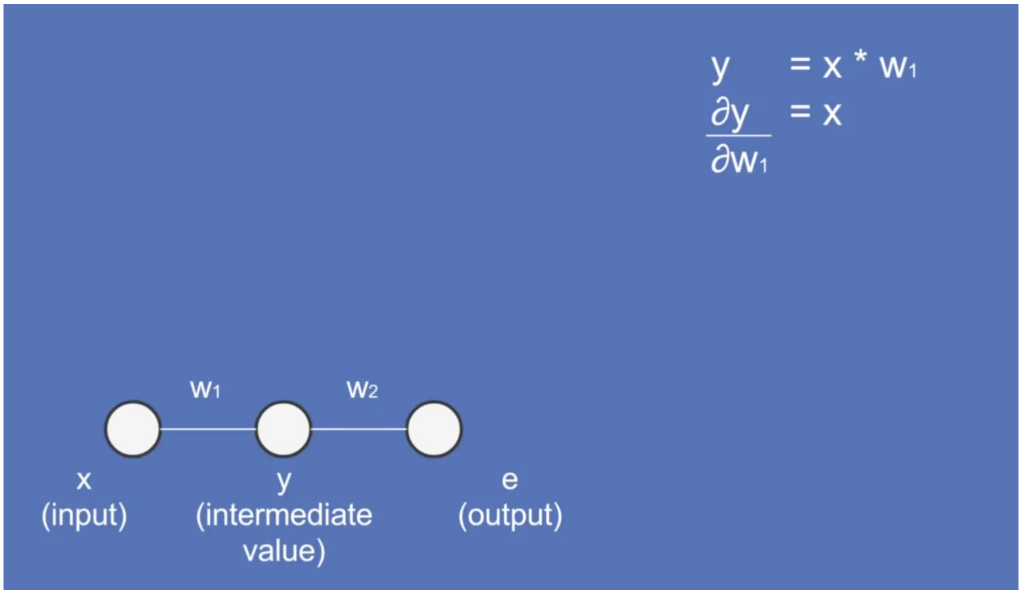
derive the inside

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

Chain rule

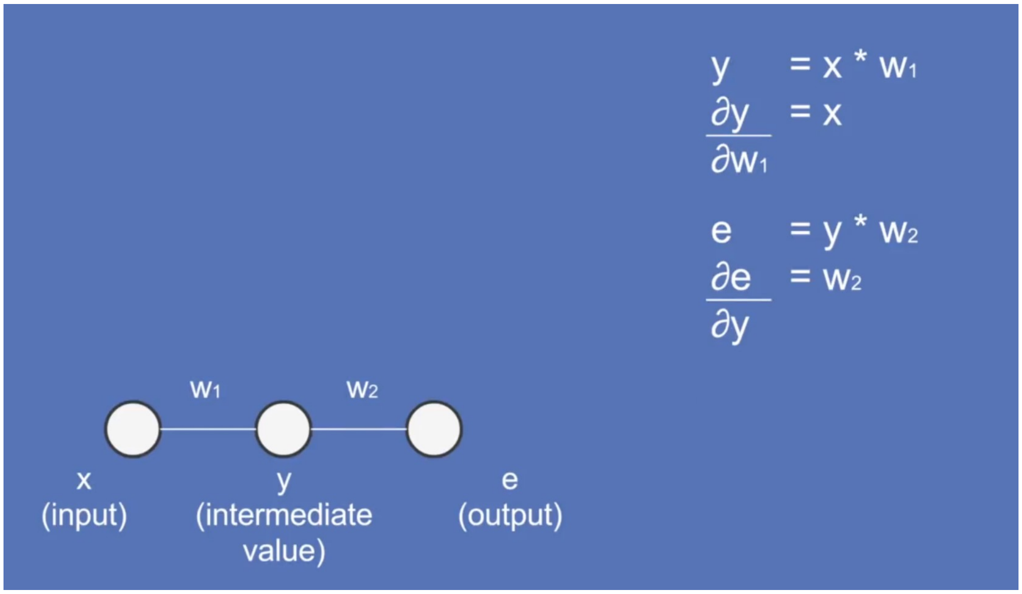


Chain rule

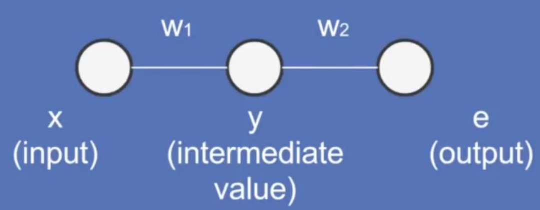


$y = x * w_1$
$\frac{\partial y}{\partial w_1} = x$
w_1
w_2
x
(input)
y
(intermediate
value)
e
(output)

Chain rule


$y = x * w_1$
$\frac{\partial y}{\partial w_1} = x$
$e = y * w_2$
$\frac{\partial e}{\partial y} = w_2$
w_1
w_2
x
(input)
y
(intermediate value)
e
(output)

Chain rule



$$y = x * w_1$$

$$\frac{\partial y}{\partial w_1} = x$$

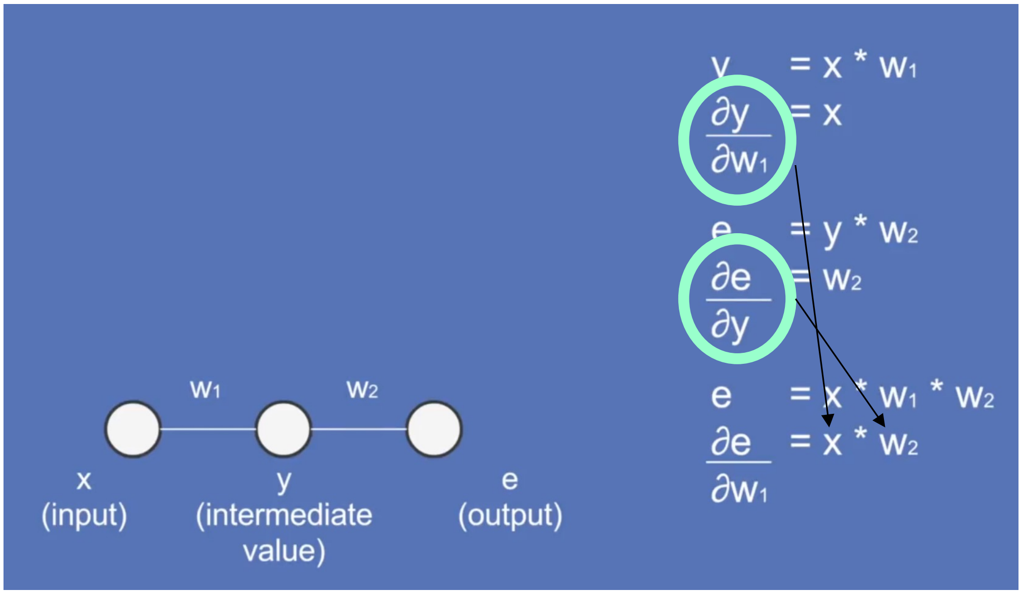
$$e = y * w_2$$

$$\frac{\partial e}{\partial y} = w_2$$

$$e = x * w_1 * w_2$$

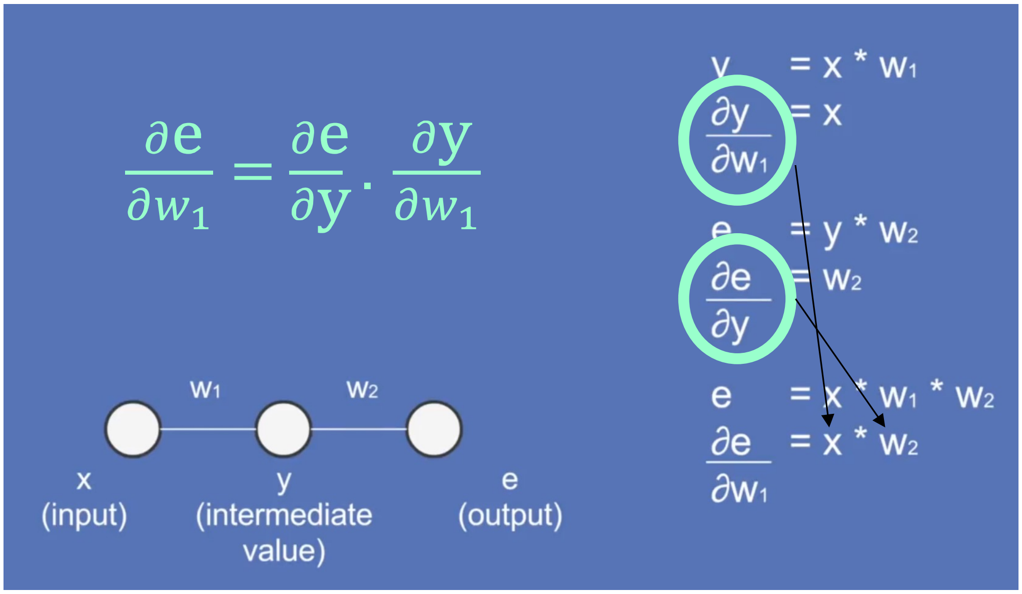
$$\frac{\partial e}{\partial w_1} = x * w_2$$

Chain rule



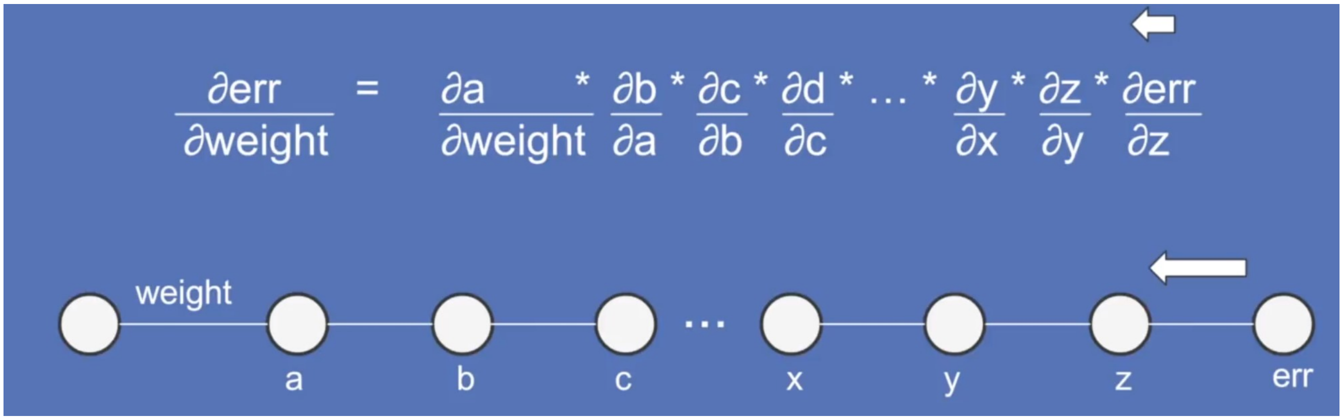
y = x * w1
∂y/∂w1 = x
e = y * w2
∂e/∂y = w2
e = x * w1 * w2
∂e/∂w1 = x * w2
w1
w2
x
(input)
y
(intermediate value)
e
(output)

Chain rule

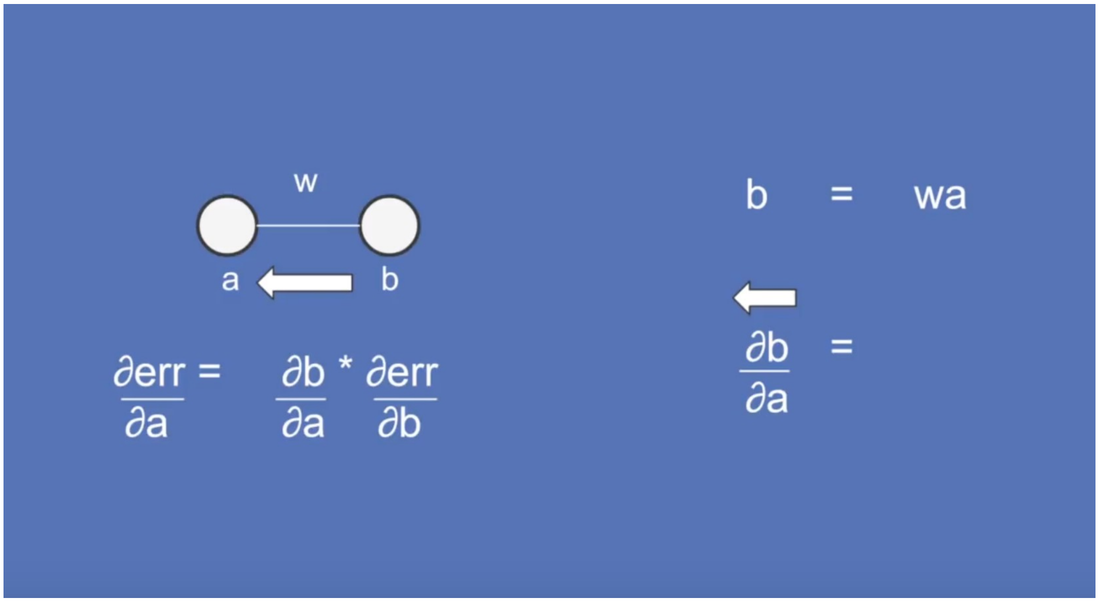


$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial w_1}$
$y = x * w_1$
$\frac{\partial y}{\partial w_1} = x$
$e = y * w_2$
$\frac{\partial e}{\partial y} = w_2$
$e = x * w_1 * w_2$
$\frac{\partial e}{\partial w_1} = x * w_2$
w_1
w_2
x
(input)
y
(intermediate
value)
e
(output)

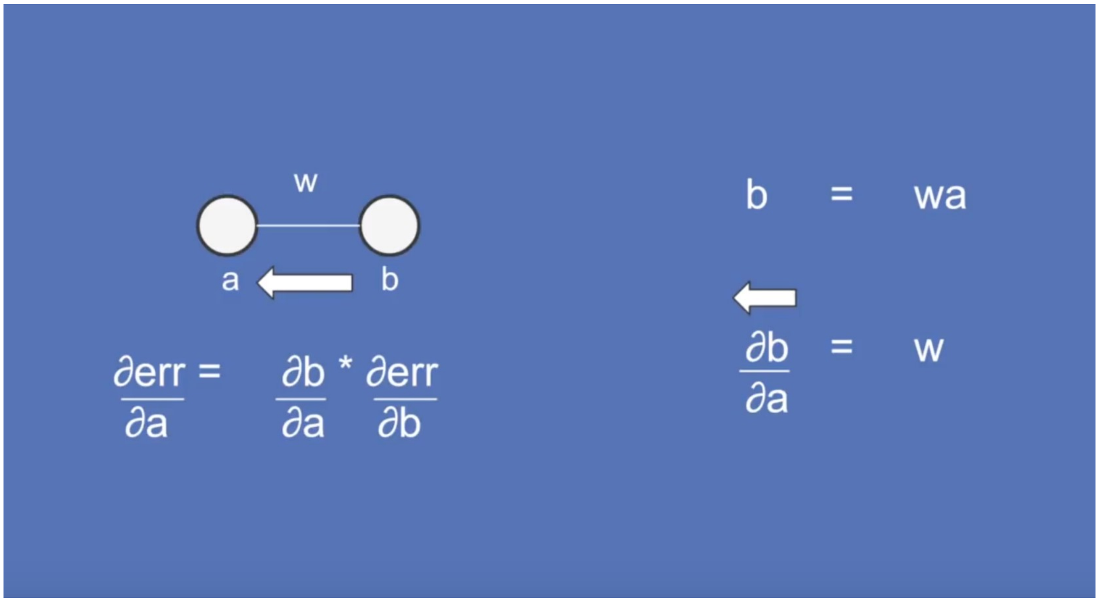
Chain Rule



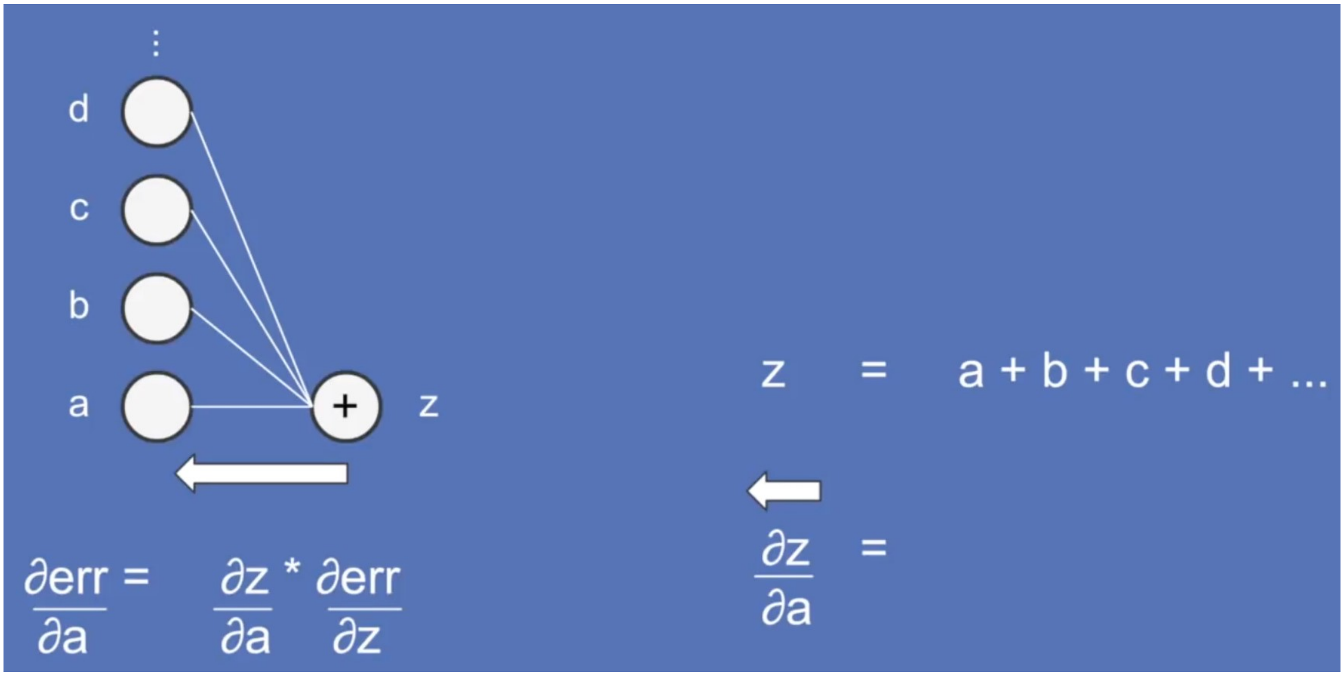
Chain rule



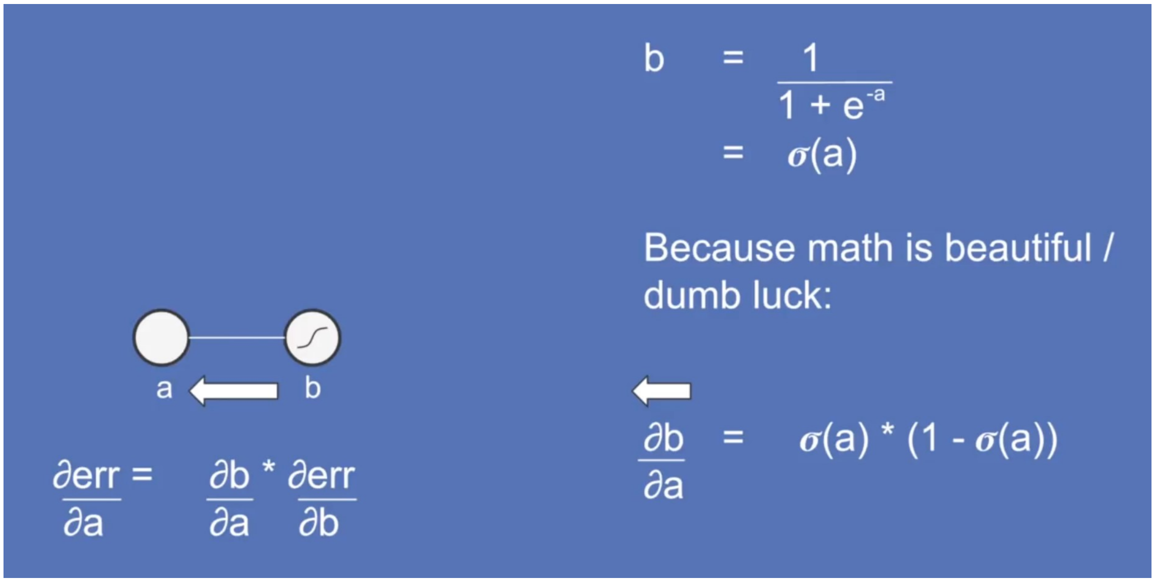
Chain rule


w
a
b
$\frac{\partial err}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial err}{\partial b}$
b = wa
$\frac{\partial b}{\partial a}$ = w

Chain rule

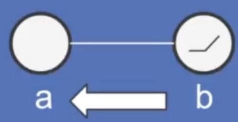


Chain rule



$b = \frac{1}{1 + e^{-a}}$
$= \sigma(a)$
Because math is beautiful / dumb luck:
$\frac{\partial b}{\partial a} = \sigma(a) * (1 - \sigma(a))$
a
b
$\frac{\partial err}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial err}{\partial b}$

Chain rule



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

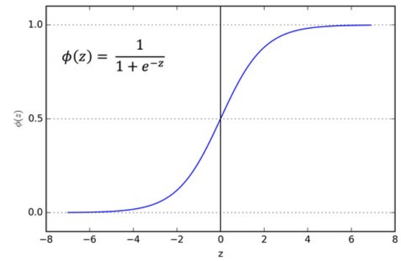
$$b = \begin{cases} a, & a > 0 \\ 0, & \text{otherwise} \end{cases}$$

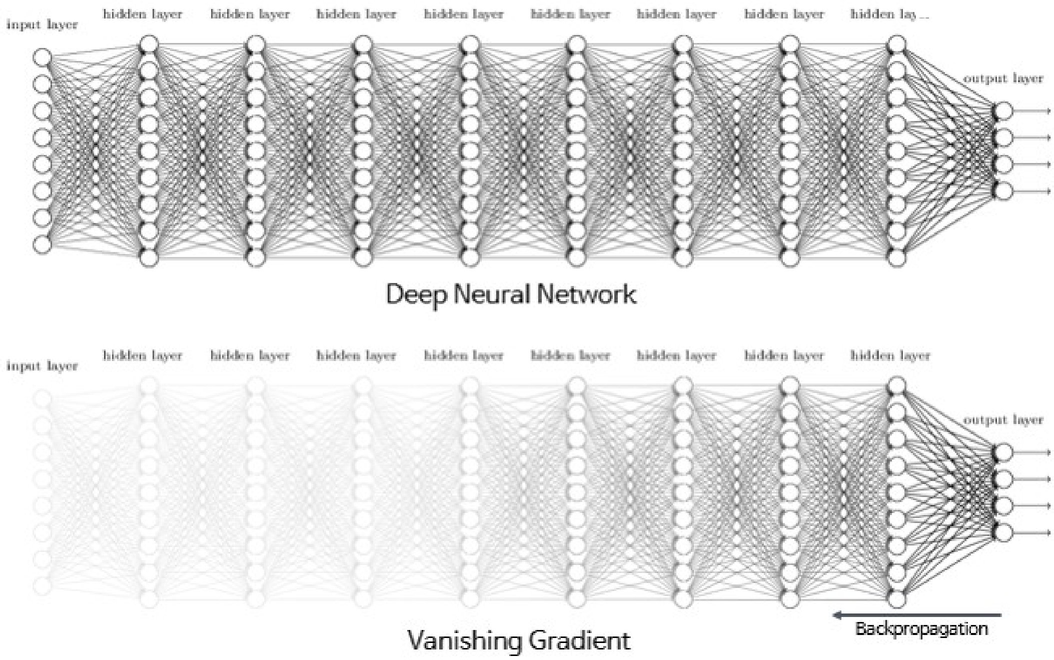
$$\leftarrow$$

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

Limitations of Gradient Descent

- Activation function: Sigmoid





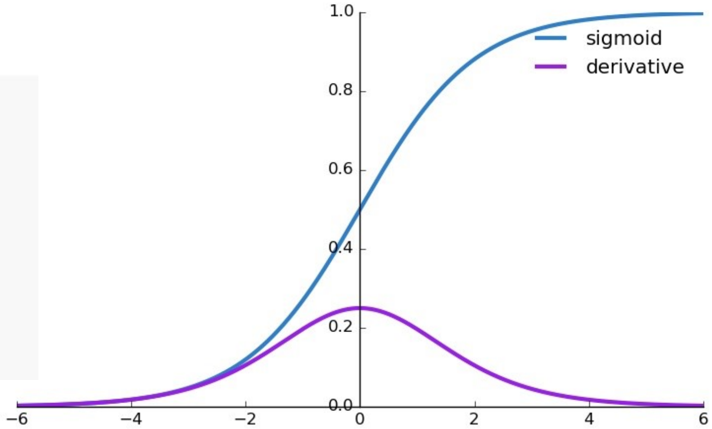
Vanishing gradient

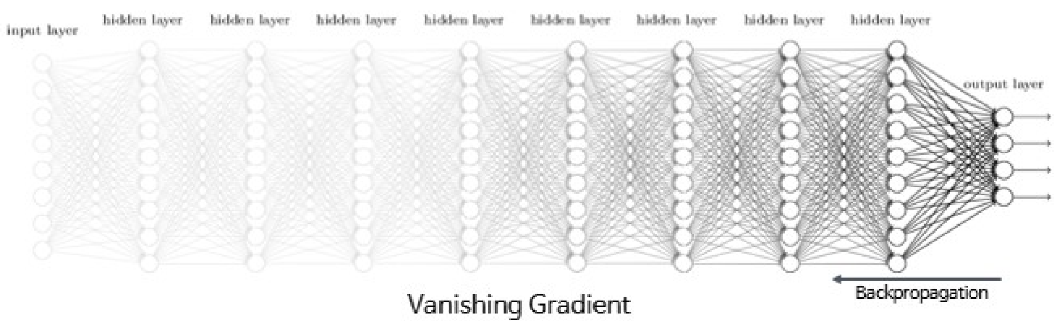
The sigmoid function is defined as follows

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

This function is easy to differentiate because

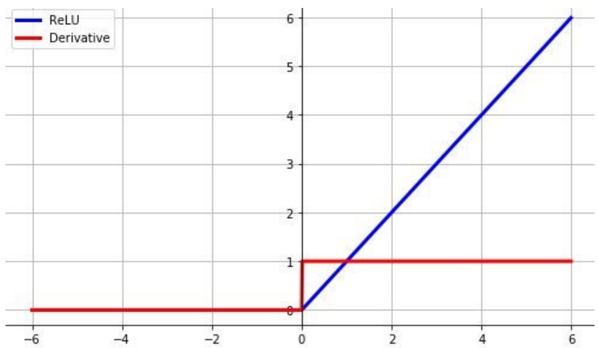
$$\frac{d\sigma(x)}{d(x)} = \sigma(x) \cdot (1 - \sigma(x)).$$





Vanishing gradient : Solution 1

ReLU activation function



Vanishing gradient : Solution 2

(mini-)batch normalization (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

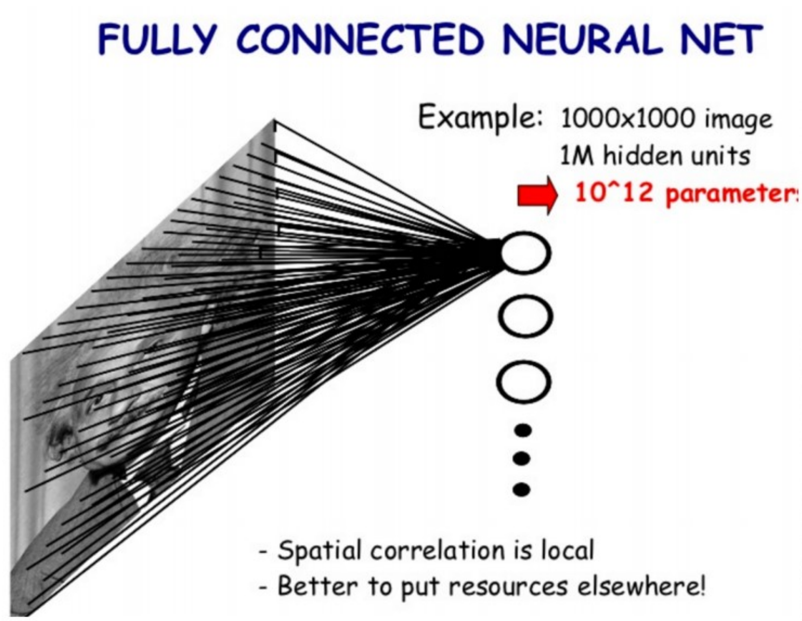
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

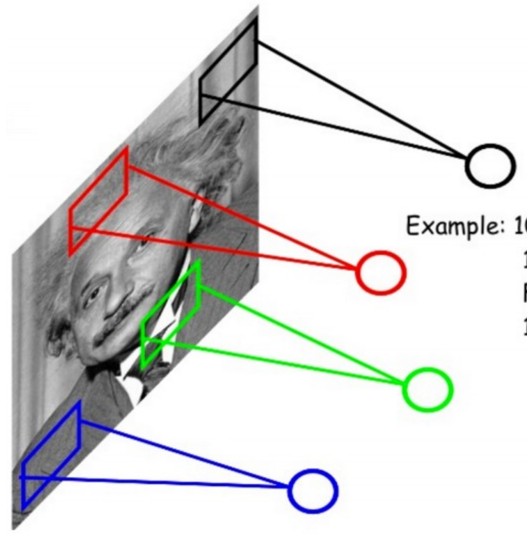
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

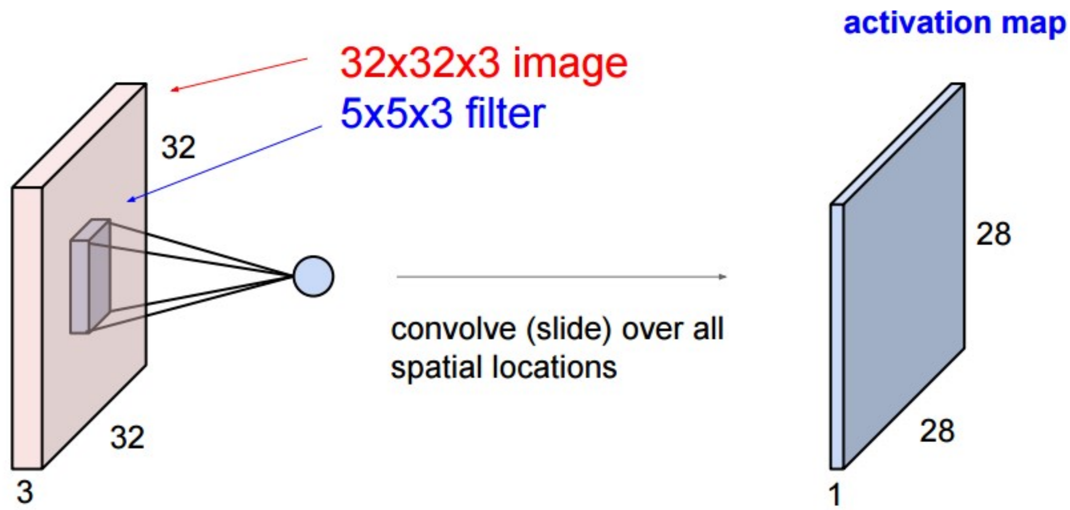

FULLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
10^12 parameter
- Spatial correlation is local
- Better to put resources elsewhere!

LOCALLY CONNECTED NEURAL NET

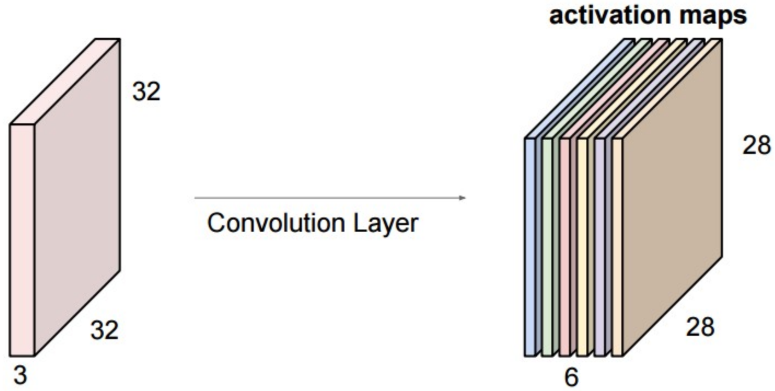


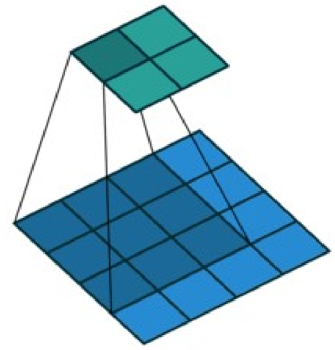
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

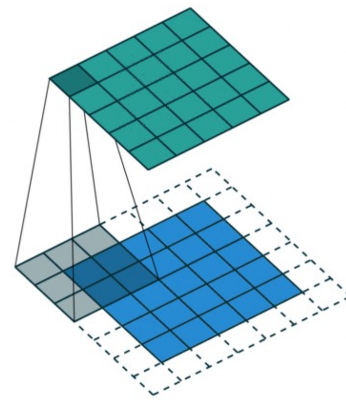
Convolution Layer



Convolution layer

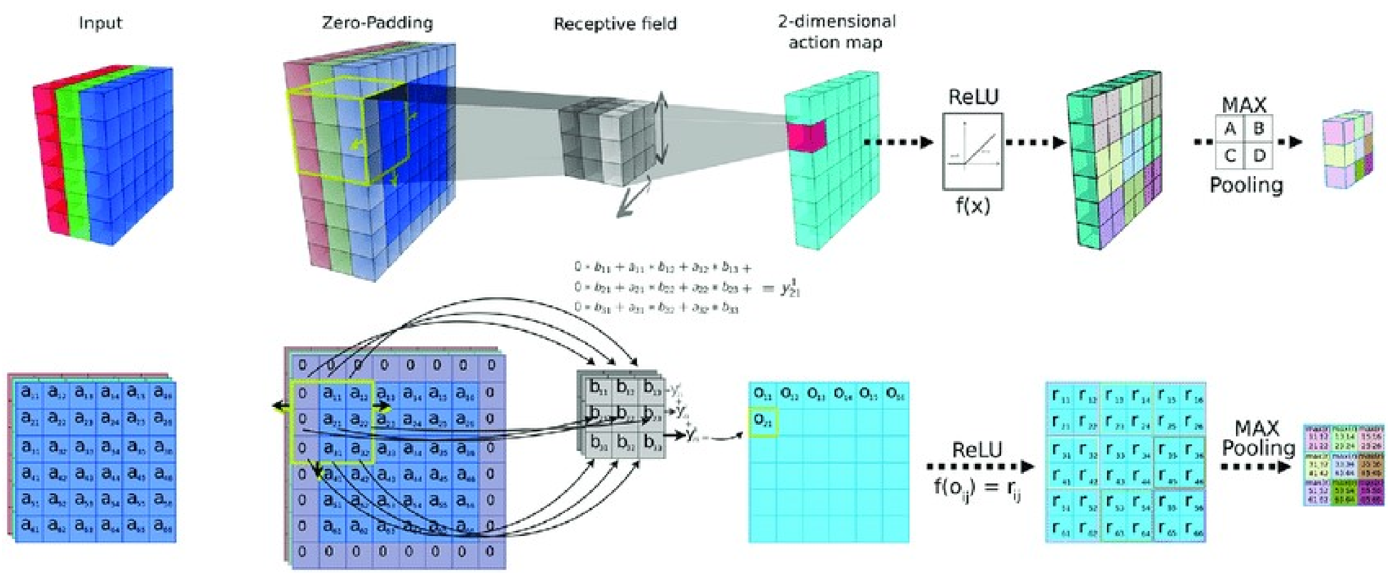




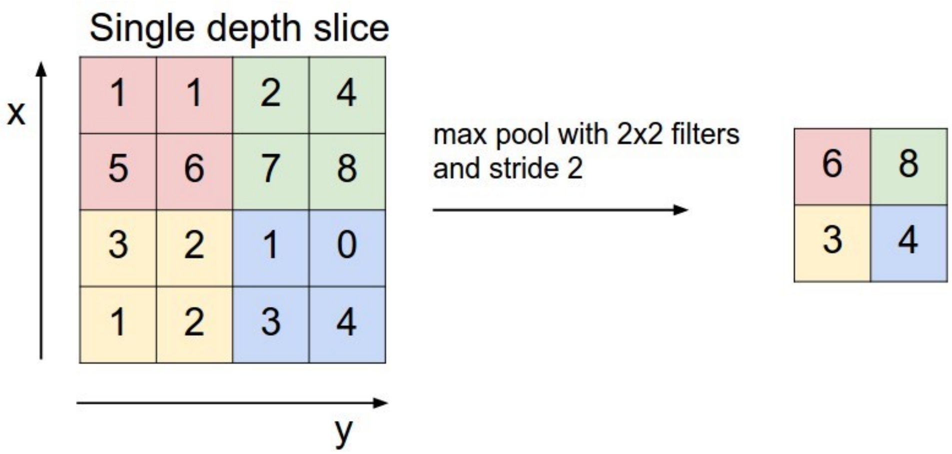


Convolution layer

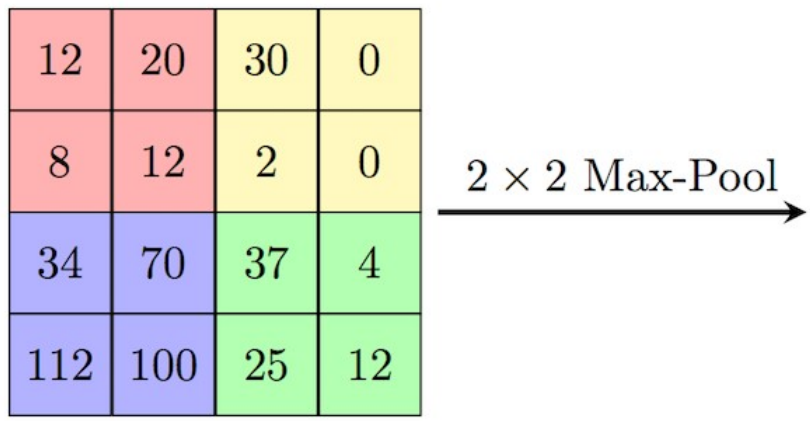
- Parametrized by: height, width, depth, stride, padding, number of filters, type of activation function



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.

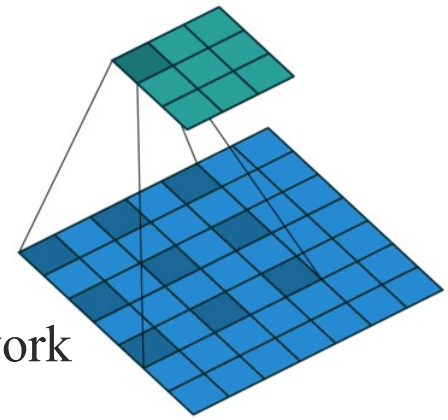
Flatten layer

- Convert the input to a 1D array.

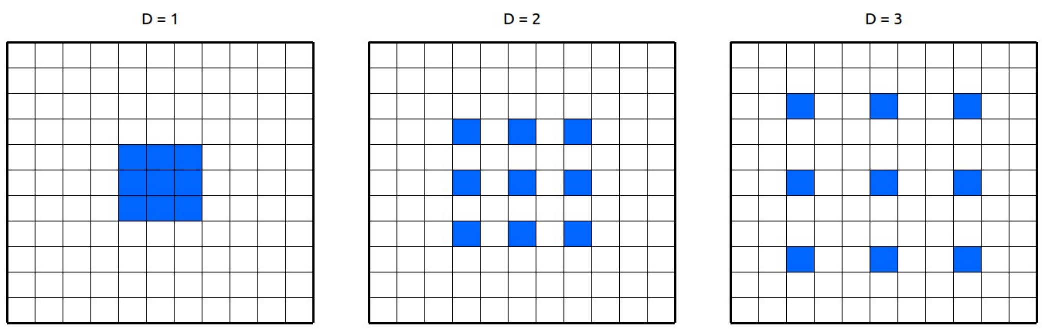
Batch Normalization (BN), Noise Layer

Dilated Convolution

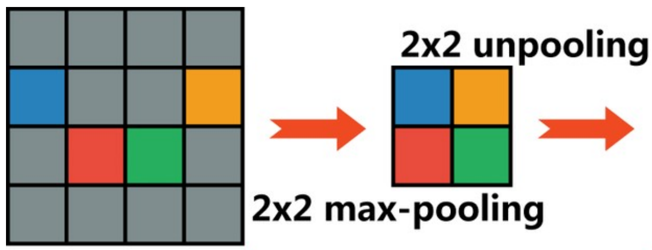
Dilating the filter means expanding its size filling the empty positions with 0.



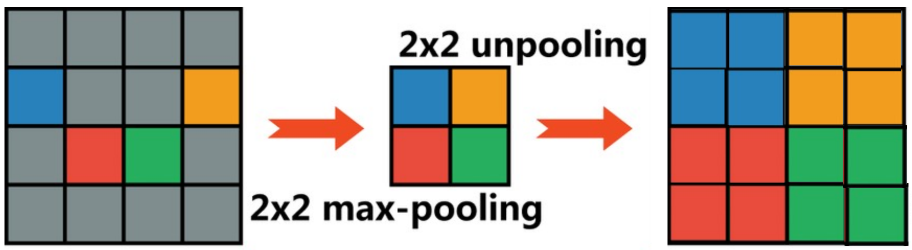
A way of increasing receptive view of the network while keeping the number of weights constant.



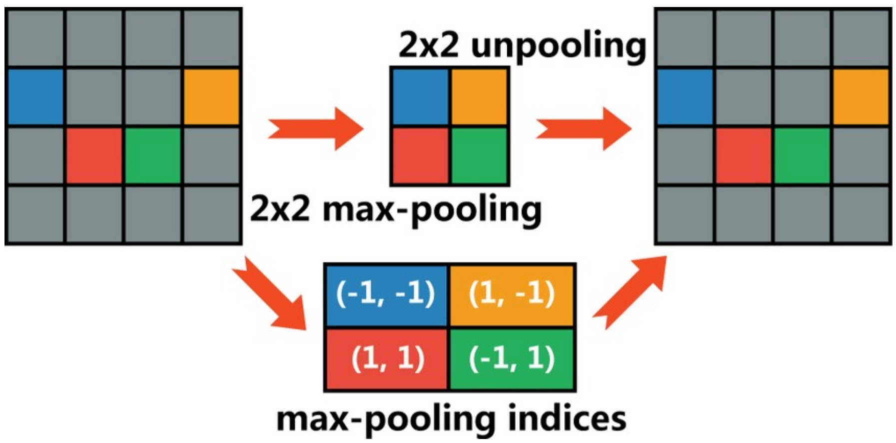
Unpooling

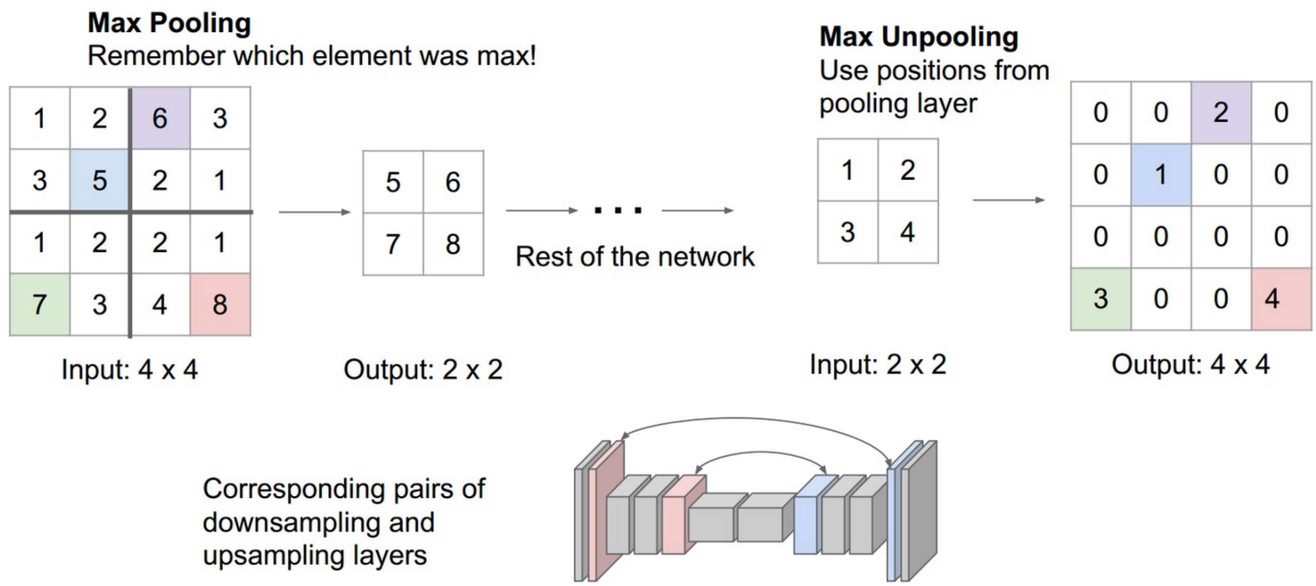


Unpooling (Nearest Neighbor)



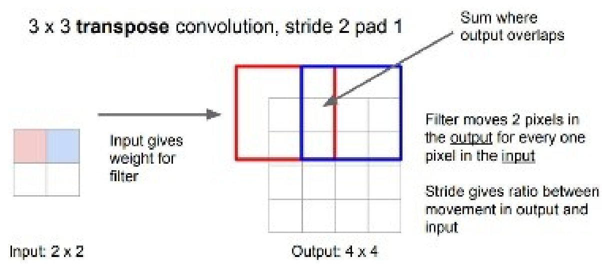
Max-Unpooling

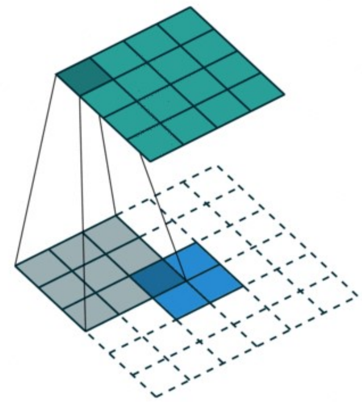


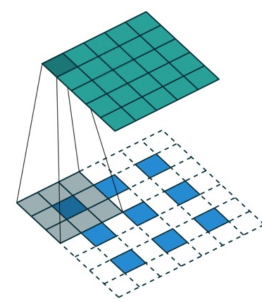

Max Pooling
Remember which element was max!
1 2 6 3
3 5 2 1
1 2 2 1
7 3 4 8
Input: 4 x 4
5 6
7 8
Output: 2 x 2
...
Rest of the network
Max Unpooling
Use positions from pooling layer
1 2
3 4
Input: 2 x 2
0 0 2 0
0 1 0 0
0 0 0 0
3 0 0 4
Output: 4 x 4
Corresponding pairs of downsampling and upsampling layers

Learned Up-Sampling (~~Deconvolution~~)

- Also called transposed convolution







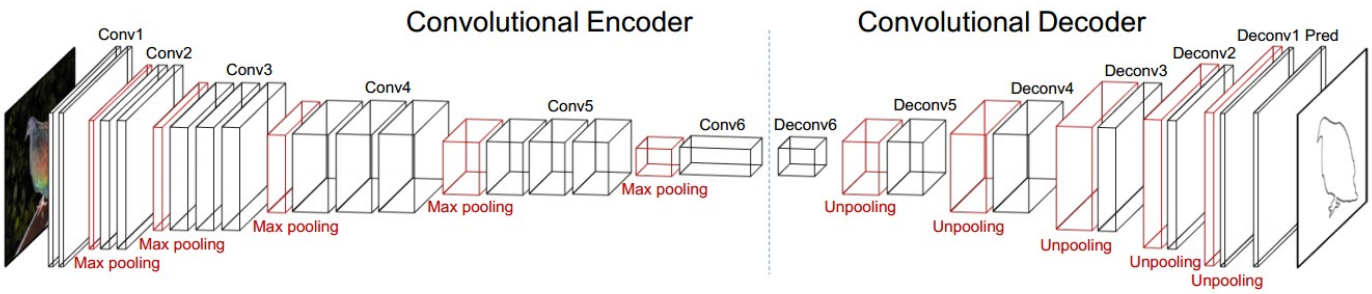
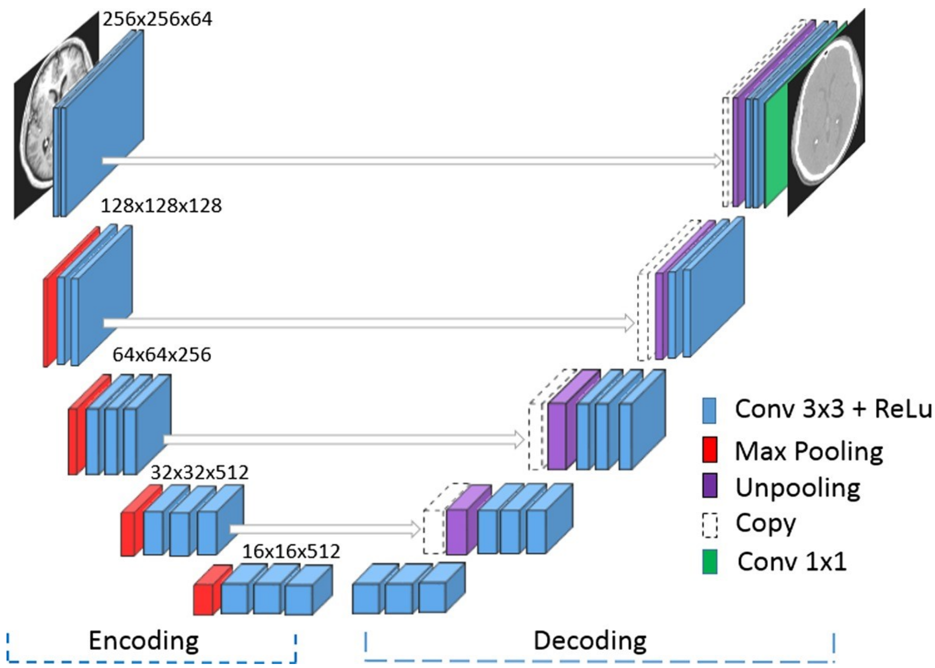
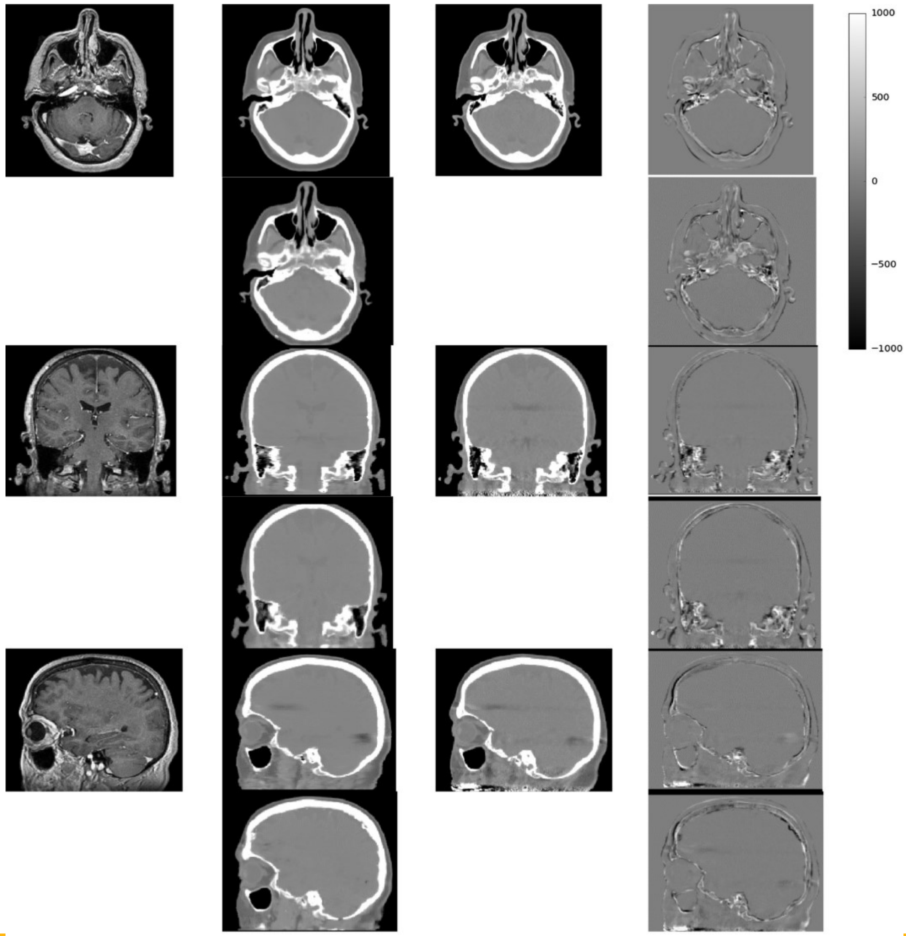


Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

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)



MR sCT Real CT Difference Map