Neural Network. Basic to application
(painting style transfer)

Kim Woo Hyun

September 4, 2019

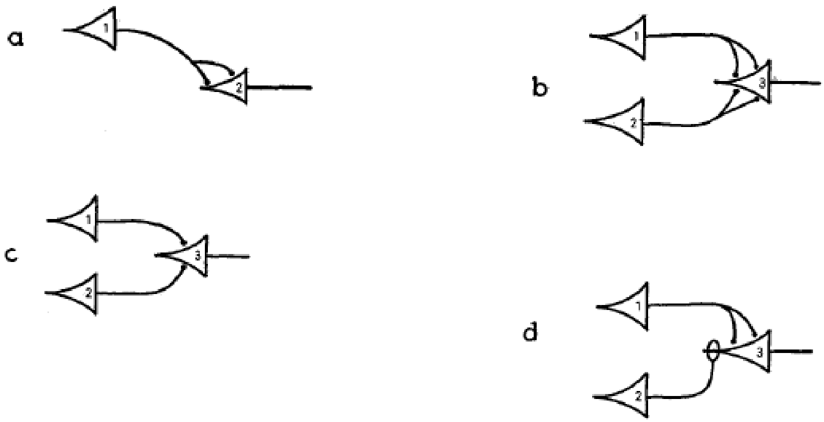
Outline

1. Neural Network
   - First Generation (ANN, Perceptron)
   - Second Generation (MLP, Back-propagation)
   - Thrid Generation (ReLU)
2. Convolutional Neural Network
   - Convolution layer
   - ReLU layer
   - Pooling layer
   - Fully Connected layer
3. Painting Style Transfer
   - VGGnet
   - Algorithm and Loss function
   - Result

First Generation

Artificial Neural Network : ANN

At 1943 ***McCulloch, Warren S.***, and ***Walter Pitts*** suggested

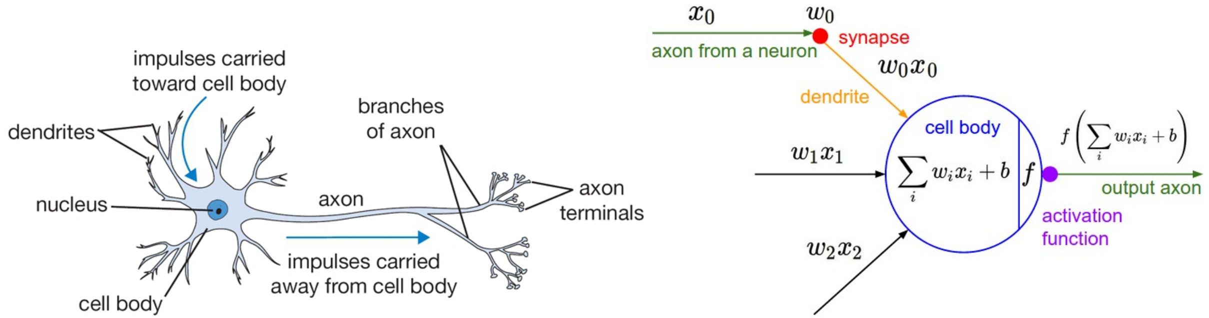


- Mimic the human neural structure by connecting switches

First Generation

Perceptron

In 1958 ***Frank Rosenblatt*** suggested Linear Classifier.

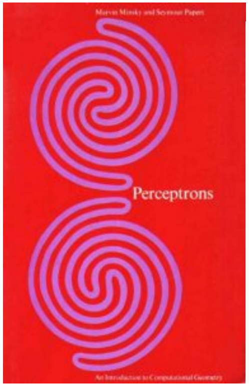


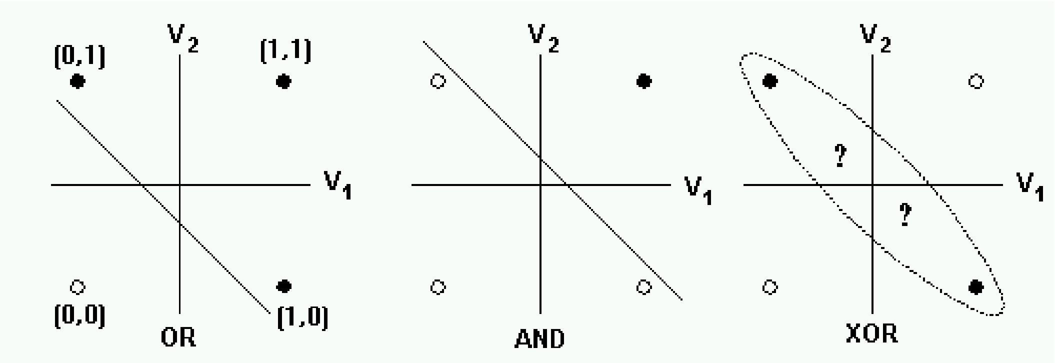
- Expected computer can do things human can do better at that time.
- Basic structure is not changed until now.
- Using sigmoid with **Activation function**. (Make output $\in [0,1]$)

First Generation

Problem

In 1969 ***Marvin Minsky, Seymour Papert*** proved limitations of perceptron.



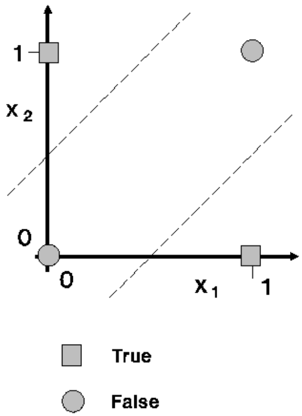


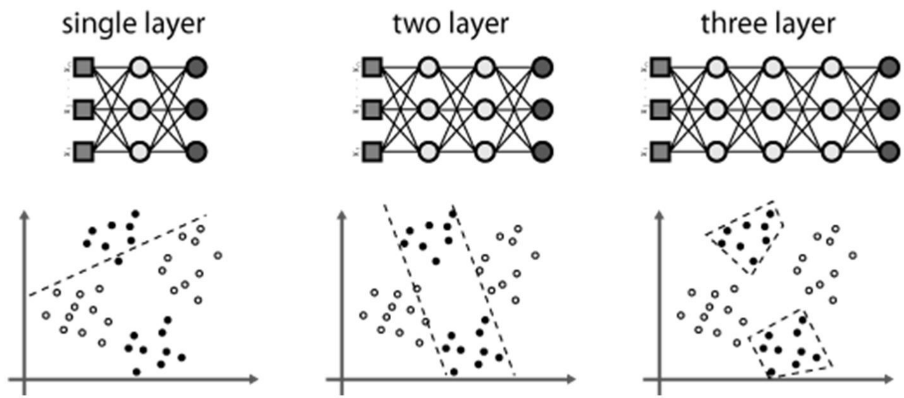
It can't solve XOR problem even.

Second Generation

Multi-Layer Perception : MLP

Make neurons deeper by make **hidden layers** of perception



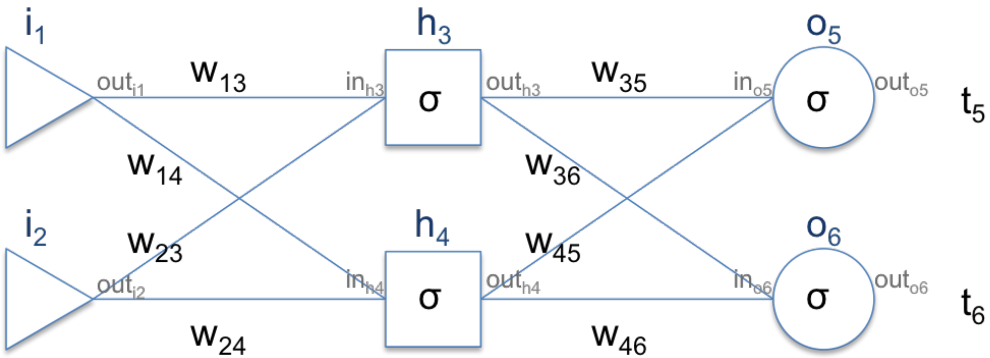


- Solve the Non-Linear problems with multiple linear classifier.
- **Too many parameters!!**
- Needs parameter controller.

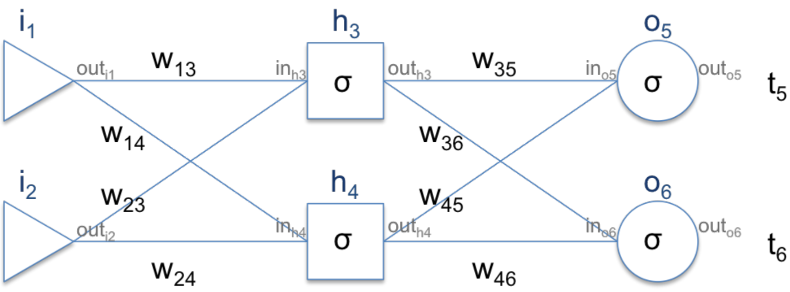
Second Generation

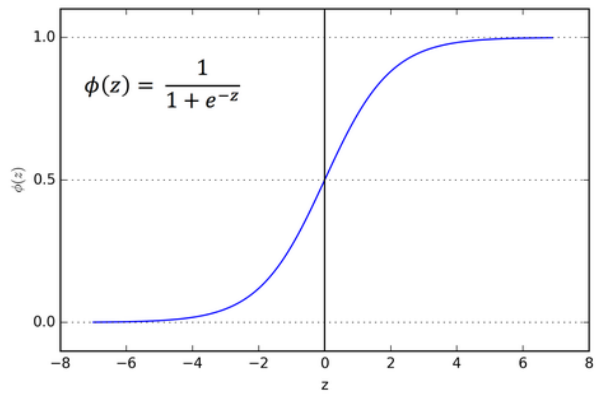
Back-propagation

Feedback algorithm controls the weights of neural network.



- i : input layer
- h : hidden layer
- o : output layer
- w_{ij} : weight connected to the neuron i to j.





- *out* : Output value of a neuron.
- *in* : sum of weighted output of connected neurons.
  $(in = \sum w * out)$
- t : Target value (Choose yourself!)
- **Sigmoid** activation function. Ex) $out_{h3} = \sigma(in_{h3}) = \frac{1}{1+e^{-in_{h3}}}$

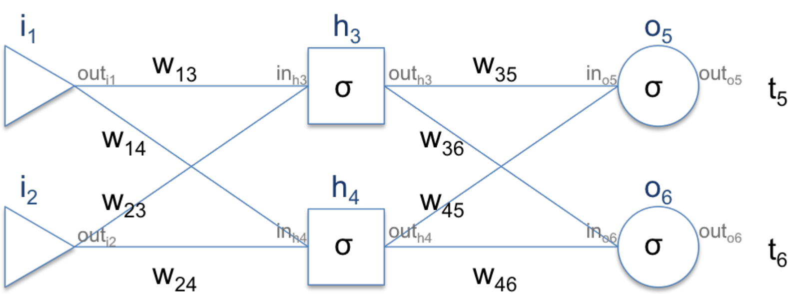
Error with Sum of square (Euclidean Distance)

$$E = \frac{1}{2}(t_5 - out_{o5})^2 + \frac{1}{2}(t_6 - out_{o6})^2$$

We want to see how much each weights influence to $E \Rightarrow$ Calculate $\frac{\partial E}{\partial w_{ij}}$

Example) Calculate $\frac{\partial E}{\partial w_{35}}$ with **Chain-rule**

$$\frac{\partial E}{\partial w_{35}} = \frac{\partial E}{\partial out_{o5}} * \frac{\partial out_{o5}}{\partial in_{o5}} * \frac{\partial in_{o5}}{\partial w_{35}}$$



Second Generation

First,

$$\frac{\partial E}{\partial out_{o5}} = \frac{\partial}{\partial out_{o5}}\left[\frac{1}{2}(t_5 - out_{o5})^2 + \frac{1}{2}(t_6 - out_{o6})^2\right] = out_{o5} - t_5$$

Second,

$$\frac{\partial out_{05}}{\partial in_{o5}} = \frac{\partial \sigma(in_{o5})}{\partial in_{o5}}$$

Second Generation

The sigmoid function $\sigma(x)$ is

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

The differential of sigmoid $\sigma(x)$

$$\begin{aligned}
\sigma'(x) &= \frac{ae^{-ax}}{(1 + e^{-ax})^2} \\
&= a\frac{1}{(1 + e^{-ax})}\frac{e^{-ax}}{(1 + e^{-ax})} \\
&= a\frac{1}{(1 + e^{-ax})}\left(1 - \frac{1}{(1 + e^{-ax})}\right) \\
&= a\sigma(x)(1 - \sigma(x))
\end{aligned}$$

Second Generation

First,

$$\frac{\partial E}{\partial out_{o5}} = \frac{\partial}{\partial out_{o5}} \left[\frac{1}{2}(t_5 - out_{o5})^2 + \frac{1}{2}(t_6 - out_{o6})^2 \right] = out_{o5} - t_5$$

Second,

$$\frac{\partial out_{05}}{\partial in_{o5}} = \frac{\partial \sigma(in_{o5})}{\partial in_{o5}} = \sigma(in_{o5})(1 - \sigma(in_{o5})) = out_{o5}(1 - out_{o5})$$

Second Generation

First,

$$\frac{\partial E}{\partial out_{o5}} = \frac{\partial}{\partial out_{o5}}\left[\frac{1}{2}(t_5 - out_{o5})^2 + \frac{1}{2}(t_6 - out_{o6})^2\right] = out_{o5} - t_5$$

Second,

$$\frac{\partial out_{05}}{\partial in_{o5}} = \frac{\partial \sigma(in_{o5})}{\partial in_{o5}} = \sigma(in_{o5})(1 - \sigma(in_{o5})) = out_{o5}(1 - out_{o5})$$

Third,

$$\frac{\partial in_{o5}}{\partial w_{35}} = \frac{\partial (out_{h3} * w_{35})}{\partial w_{35}} = out_{h3}$$

Finally,

$$\frac{\partial E}{\partial w_{35}} = (out_{o5} - t_5)(1 - out_{o5})out_{o5}out_{h3}$$

Beautifully, all parameters are already calculated and what we have to do is easy math.

Second Generation

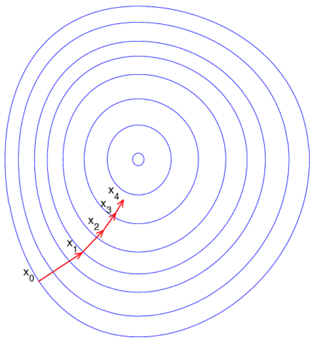
Then, how to update weights?

$$w := w - r\frac{\partial E}{\partial w}, \text{ r is constant called learning rate.}$$

So, updated w_{35} is

$$w_{35} := w_{35} - r(out_{o5} - t_5)(1 - out_{o5})out_{o5}out_{h3}$$

This method called **Gradient descent.**



Second Generation

Gradient descent

Simply, moving to orthogonal direction from contour line.

Why the direction to orthogonal? At minimum point of f(x,y),

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = 0$$

Assume direction of contour line is (a, b). Then using **Tayler series**, derive orthogonal direction by linearize the contour line.

$$f(x_1 + a, y_1 + b) \simeq f(x_1, y_2) + \frac{\partial f}{\partial x} a + \frac{\partial f}{\partial y} b + \dots$$

The condition of (a, b) that minimize error is

$$\frac{\partial f}{\partial x} a + \frac{\partial f}{\partial y} b = 0$$

Second Generation

If $a = \frac{\partial f}{\partial y}$ and $b = -\frac{\partial f}{\partial x}$.

$$\frac{\partial f}{\partial x}a + \frac{\partial f}{\partial y}b = \frac{\partial f}{\partial x}\frac{\partial f}{\partial y} + \frac{\partial f}{\partial y}(-\frac{\partial f}{\partial x}) = 0$$

In addition, the inner product of gradient and (a,b) is

$$(\nabla f(x,y)) \cdot (a,b) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \cdot \left(\frac{\partial f}{\partial y}, -\frac{\partial f}{\partial x}\right) = 0$$

It means the vector orthogonal to contour line is gradient itself. And if we track the gradient until it is 0, we can find minimum point.

*Caution it can be a saddle point not minimum but I don't want to discuss in this time because I don't know.

Second Generation

Problems

- Gradient descent is bad at non-convex function, but sigmoid is non-convex function.

$$\sigma''(x) = a^2\sigma(x)(1-\sigma(x))(1-2\sigma(x))$$

$$a^2\sigma(x)(1-\sigma(x)) \geq 0 \text{ but } -1 \leq 1-2\sigma(x) \leq 1$$

- Cost of back-propagation is Big.
- Vanishing Gradient Problem.

Second Generation

Cost of back-propagation.

Cost is big at shallow layer.

For example,

$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial out_{h3}} * \frac{\partial out_{h3}}{\partial in_{h3}} * \frac{\partial in_{h3}}{\partial w_{13}}$$

$$\vdots$$

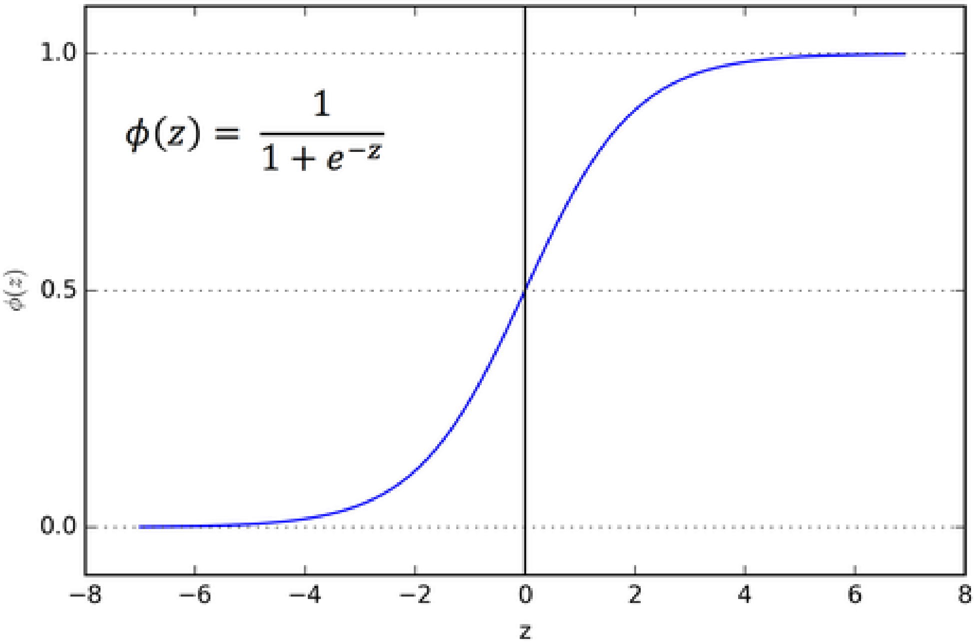
$$= [(out_{o5} - t_5)\{out_{o5}(1 - out_{o5})\}w_{35} + (out_{o5} - t_5)\{out_{o6}(1 - out_{o6})\}w_{36}]$$

$$*(1 - out_{h3}) * out_{h3} * out_{i1}$$

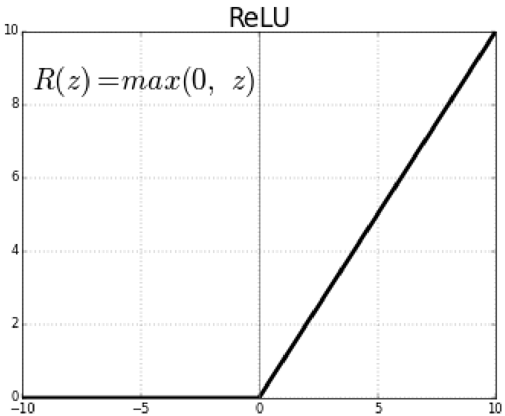
Of course! since it is chain-rule algorithm, it is easier than looks like. However if we have very big network?

Vanishing Gradient Problem

Because of sigmoid function, gradient is going to 0 while repeat Back-propagation.



Thrid Generation



Rectified Linear Unit : ReLU

- Convex : good at gradient descent.
- Cost of Back-propagation is decrease. (since $f'(x) = 1$ or 0 always)
- Safe from Vanishing Gradient Problem

All problems are from bad activation function.

Thrid Generation

Table 3: Non-linearities tested.

Name	Formula	Year
none	y = x	-
sigmoid	$y = \frac{1}{1+e^{-x}}$	1986
tanh	$y = \frac{e^{2x}-1}{e^{2x}+1}$	1986
ReLU	y = max(x, 0)	2010
(centered) SoftPlus	$y = \ln(e^x + 1) - \ln 2$	2011
LReLU	y = max(x, αx), $\alpha \approx 0.01$	2011
maxout	y = max(W_1x + b_1, W_2x + b_2)	2013
APL	y = max(x,0) + $\sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$	2014
VLReLU	y = max(x, αx), $\alpha \in 0.1, 0.5$	2014
RReLU	y = max(x, αx), α = random(0.1, 0.5)	2015
PReLU	y = max(x, αx), α is learnable	2015
ELU	y = x, if x $\geq$ 0, else $\alpha(e^x - 1)$	2015

Notice at gap between tanh and ReLU.

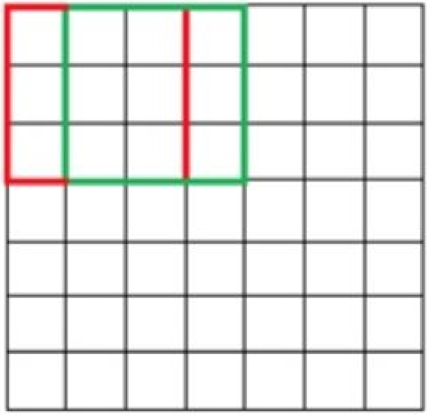
Section 2. Convolutional Neural Network

- Convolution layer
- ReLU layer
- Pooling layer
- Fully Connected layer

Convolution layer

2D Convolution

Nothing specially different from 1D convolution.



- Input size = 7x7x1
- Filter size = 3x3
- The number of filter = 1
- Stride = 1

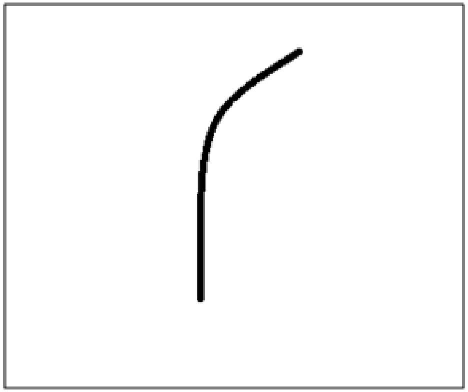
Convolution layer

What is the filter do?

Assume weights are already trained.

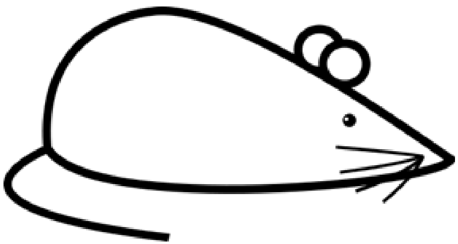
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

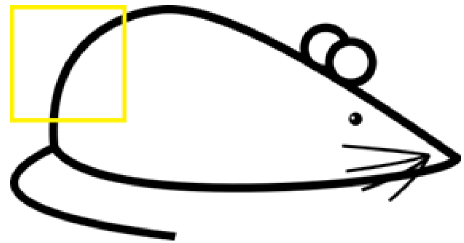


Visualization of a curve detector filter

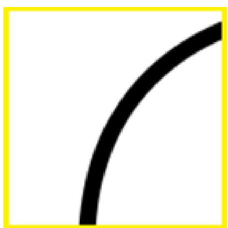
Curve detection filter and its visualization.



Original image



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

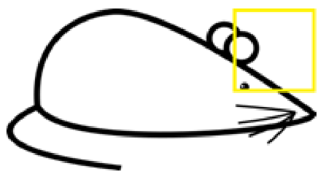
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

If Original image has similar shape at part, the result of Mult and Sum has a large number.

Filter



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

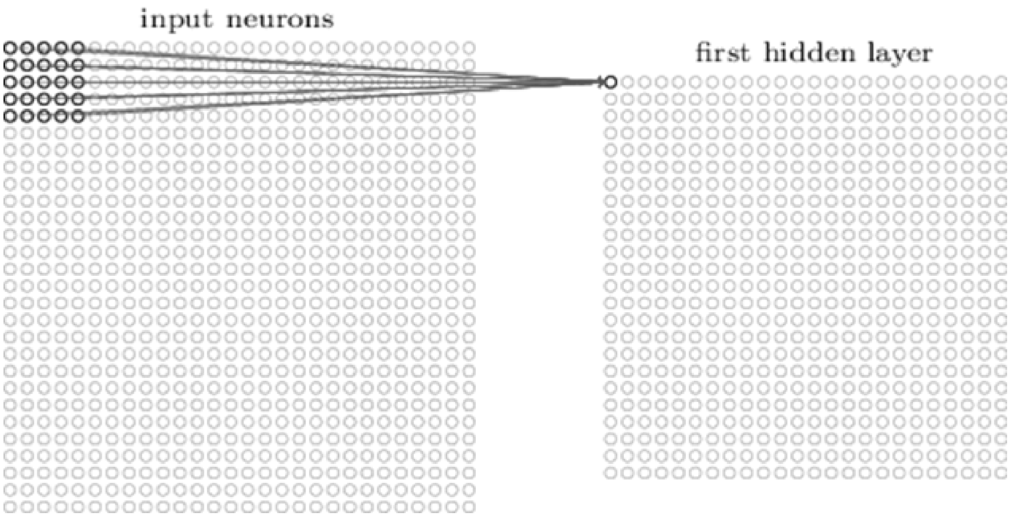
Pixel representation of filter

Multiplication and Summation = 0

In contrast, If not, the result has a small number.

Trained filter can **give a score** for which feature exist or not!!

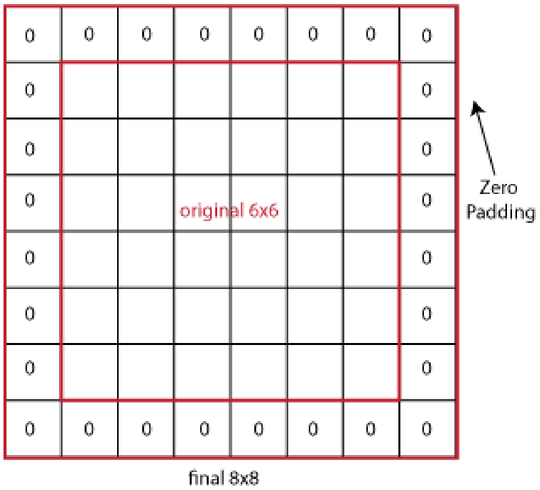
Filter



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Each score is grouped together and forms layer by convolution.

Padding

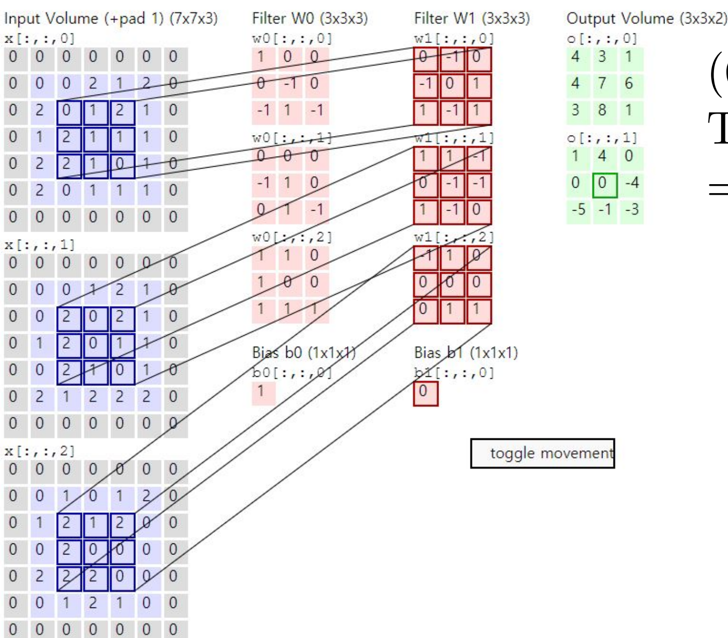


- Attach zeros around the layer. (Zero-padding)
- Prevent from size decreasing while convolution.
- To catch the features at edge more detail.

Convolution layer

Convolution

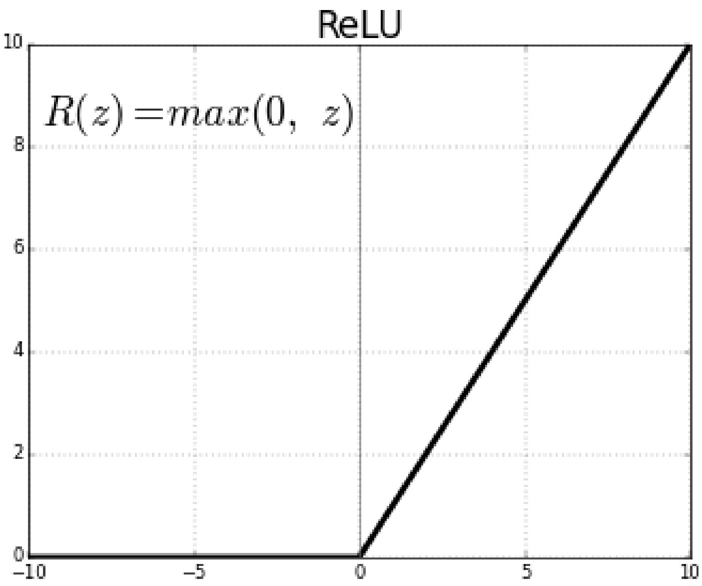
W = width, H = Height, D = Depth, P = Padding, S = stride.
F = Filters W and H, N = Number of filters.



(6+1)x(6+1)x3 input
Two 3x3x3 filters
$\Rightarrow$ Two output with 3x3x2

- $W_2 = \frac{W-F+2P}{S} + 1 = \frac{6-3+2*1}{2} + 1 = 3$
- $H_2 = \frac{H-F+2P}{S} + 1 = \frac{6-3+2*1}{2} + 1 = 3$
- $D_2 = N = 2$ (Depth is same with Number of filters)

ReLU layer



- Zero OR Itself.
- Used to give Non-linearity and threshold.
- No parameter. No size change.

ReLU layer

Why we have to give a Non-linearity.

Experimental result is given.

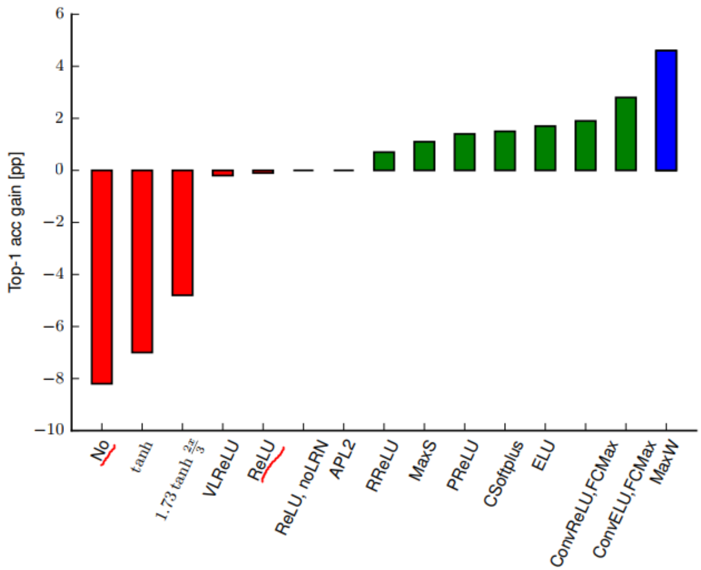
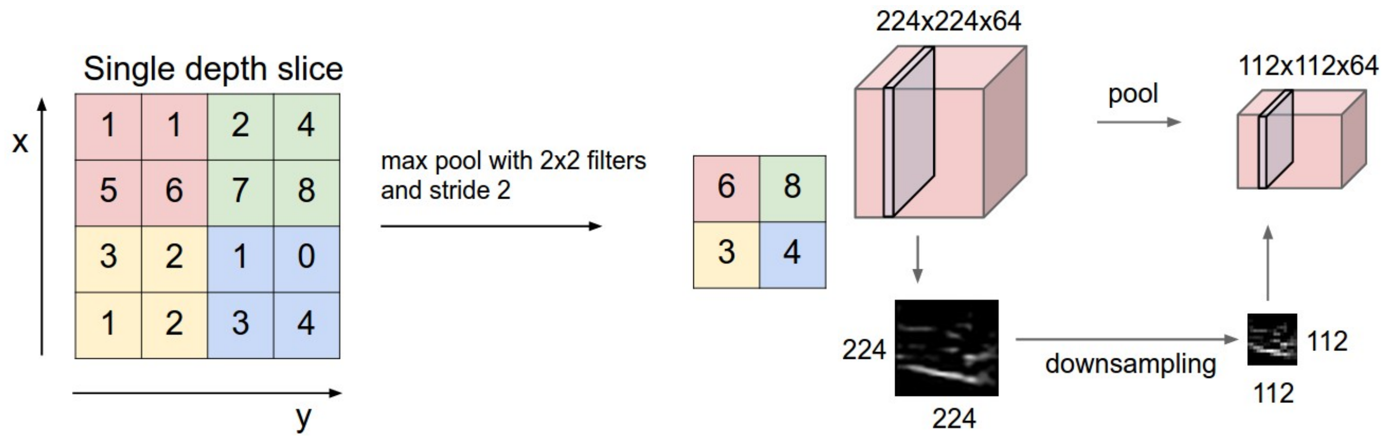


Figure 2: Top-1 accuracy gain over ReLU in the CaffeNet-128 architecture. MaxS stands for "maxout, same compexity", MaxW – maxout, same width, CSoftplus – centered softplus. The baseline, i.e. ReLU, accuracy is 47.1%.

With Image.net classification test.

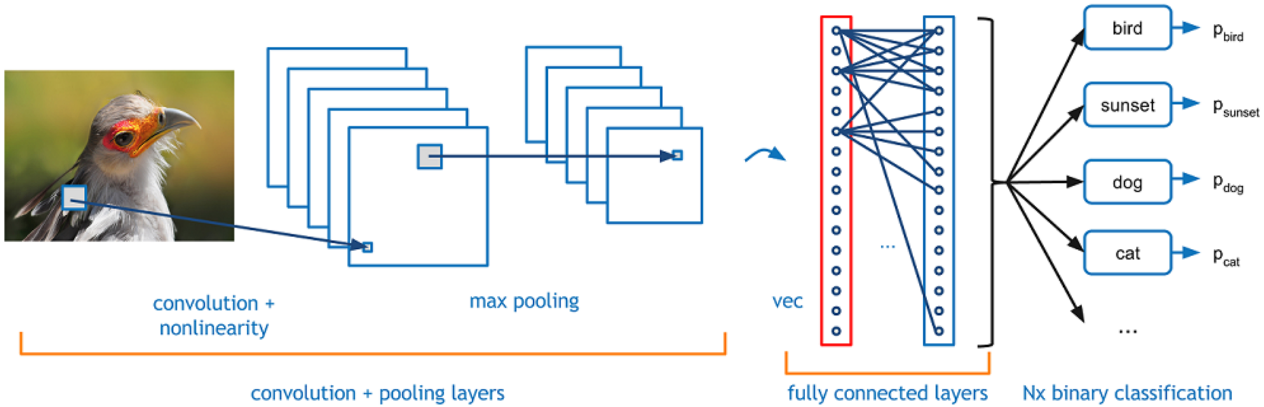
Pooling layer



- Usually, using **Max-Pooling.** (If higher value is important)
- No depth change.
- ***Reduce Complexity!!!!!!(Down-sampling)*** $\frac{1}{4} = 75\%$ reduced.
- Not Recessary. (But Recommended)

$$W_2 = \frac{W - F}{S} + 1 = \frac{224 - 2}{2} + 1 = 112$$

Fully Connected layer

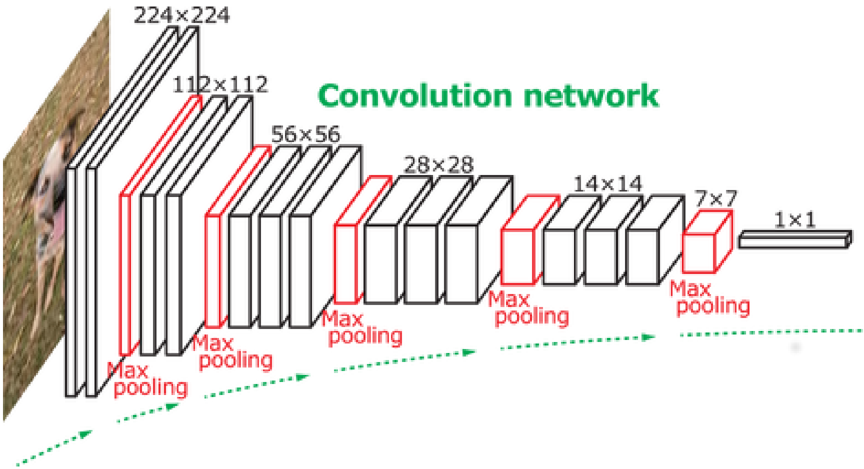


- Make 2D layer to 1D line layer (Make layer to vector.)
- Used to compare with target.
- Making method is not only one.

Section 3. Painting Style Transfer

- VGGnet
- Algorithm and Loss function
- Result

VGGnet

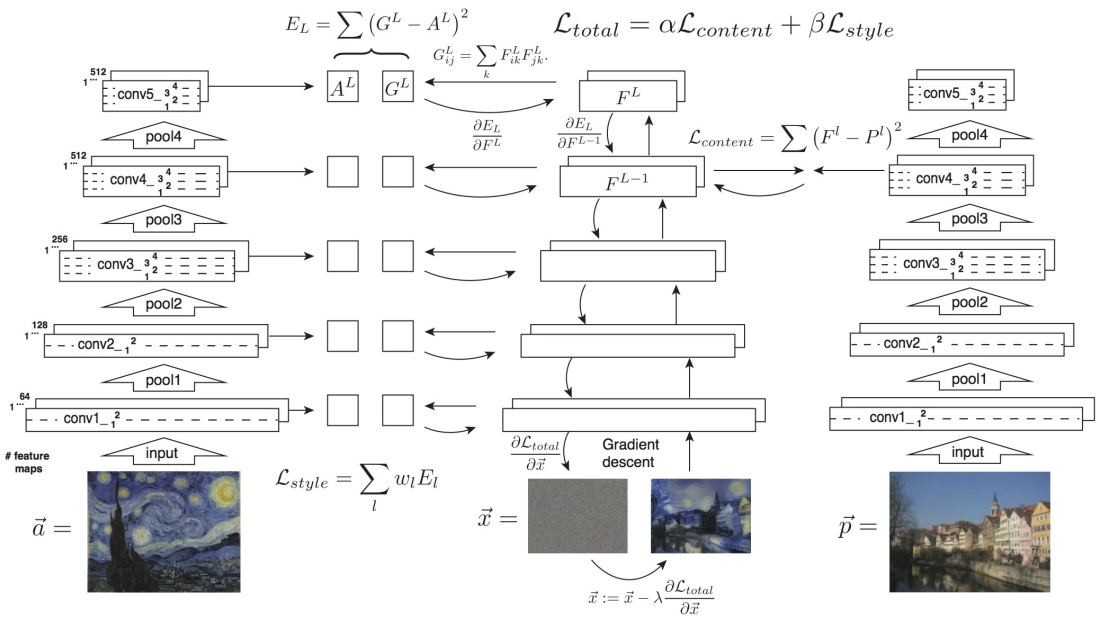


- $F_{conv} = 3\ (3 * 3 * D), S_{conv} = 1, Padding = 1$
- $F_{Pool} = 2\ (2 * 2 * D), S_{pool} = 2$

$$\frac{W - F_{conv} + 2P}{S_{conv}} + 1 = \frac{224 - 3 + 2 * 1}{1} + 1 = 224$$

$$\frac{W - F_{conv}}{S_{pool}} + 1 = \frac{224 - 2}{2} + 1 = 112$$

Painting style transfer



- Weights must be trained already.
- a = style image, p = content image
- x = generated image.

Painting style transfer

- N_l = Number of feature maps of lth layer
- M_l = Size of feature map of lth layer
- $F^l \in \mathcal{R}^{N_l * M_l}$
- F^l_{ij} is the activation of the i^{th} filter at position j in layer l
- P^l_{ij} is same with F^l_{ij} but it is from content image.(conv4_2)

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2.$$

So this loss function want to minimize distance of each value of same position between content layer and generate layer.

- $G^l \in \mathcal{R}^{N_l * N_l}$
- G^l_{ij} is the inner product between the vectorized feature maps i and j in layer l (Gram matrix of style layer)
- $$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$
- A^l_{ij} is same with G^l_{ij} but it is from content image.

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2$$

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

They have thought the style information is hide on correlation but I can't understand.

Painting style transfer

The differential of each loss function are

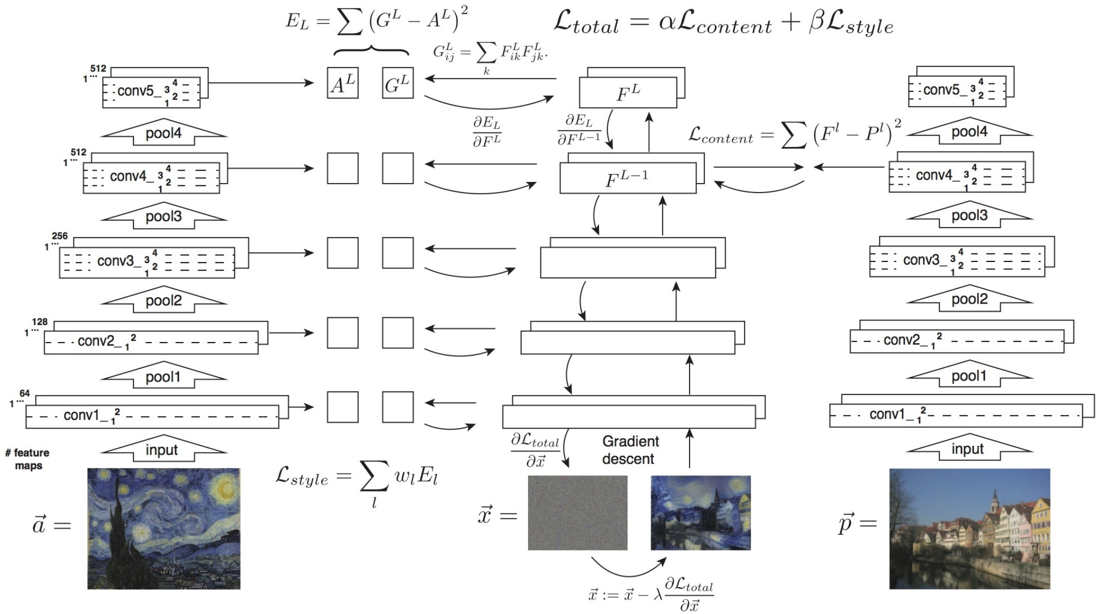
$$\frac{\partial \mathcal{L}_{\text{content}}}{\partial F^l_{ij}} = \begin{cases} (F^l - P^l)_{ij} & \text{if } F^l_{ij} > 0 \\ 0 & \text{if } F^l_{ij} < 0, \end{cases}$$

$$\frac{\partial E_l}{\partial F^l_{ij}} = \begin{cases} \frac{1}{N_l^2 M_l^2}((F^l)^{\mathrm{T}}(G^l - A^l))_{ji} & \text{if } F^l_{ij} > 0 \\ 0 & \text{if } F^l_{ij} < 0. \end{cases}$$

And the total loss is

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$

- α and β are learning rate.

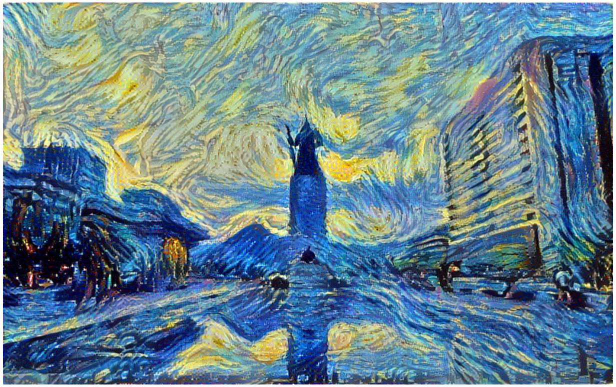


$$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$$

- λ is learning rate.
- At first, $\vec{x}$ is white noise image.
- **Not learning weights, learning $\vec{x}$!!!!**

Result

+



Bonus

Thank you!