Interfețe Vizuale Om-Mașină
IVOM

Cosmin Țoca	
Mail.	cosmin.toca@gmail.com

IVOM

Deep Learning - Convolutional Neural Networks

Overview

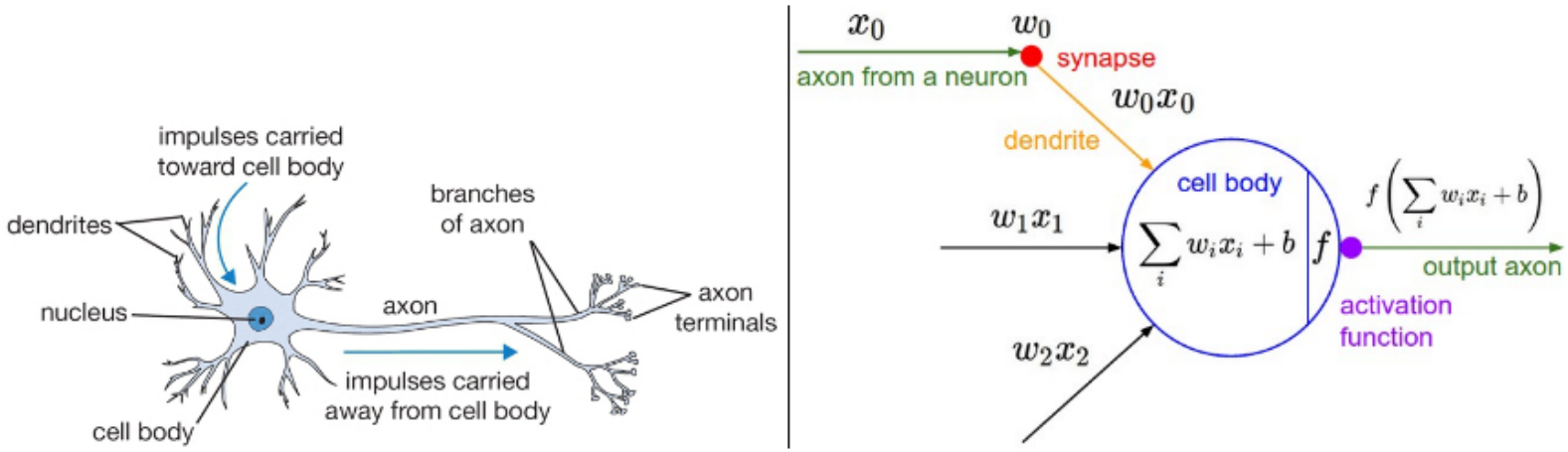
1. Regular Neural Networks (Multi-Layer Perceptron)
2. Deep Learning (Convolutional Neural Networks)
3. Open Source Tools
4. Applications

Tasks

Object Classification
Object Detection
Pixel-Wise Segmentation
Style Transfer
Generative Learning
Scene Understanding

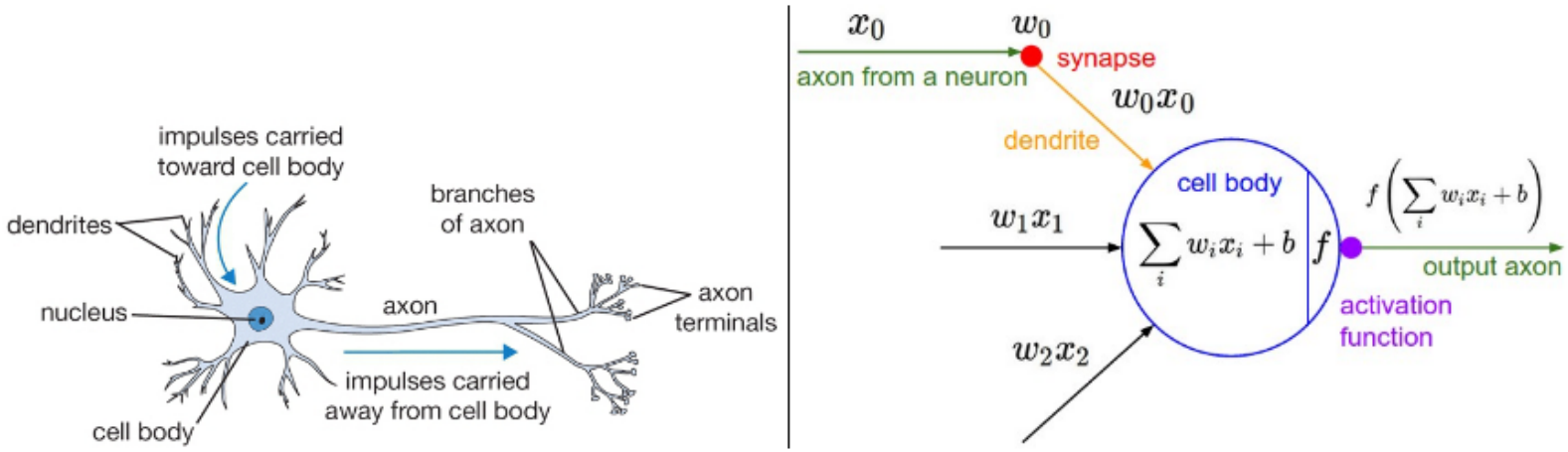
Regular Neural Networks – MLP (Multilayer Perceptron)

A cartoon drawing of a biological neuron (left) and its mathematical model (right).



Regular Neural Networks – MLP (Multilayer Perceptron)

A cartoon drawing of a biological neuron (left) and its mathematical model (right).



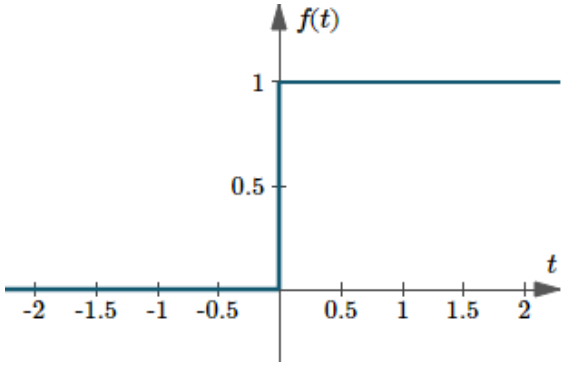
Inputs (X)	x_i
Weights (W)	w_i
Bias (B)	b
Transfer Function (NET)	$net = \sum_i w_i x_i + b$
Activation Function (OUT)	$out = f(net)$
Error (ERR)	$err = \frac{1}{2}(out - target)^2$

IVOM

Regular Neural Networks

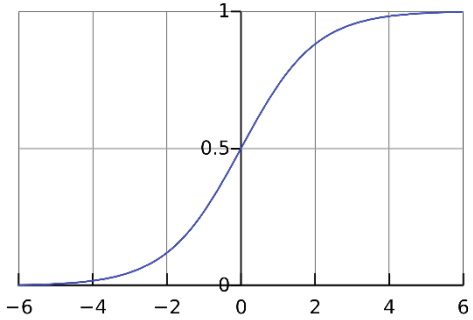
Activation Functions

Binary Step



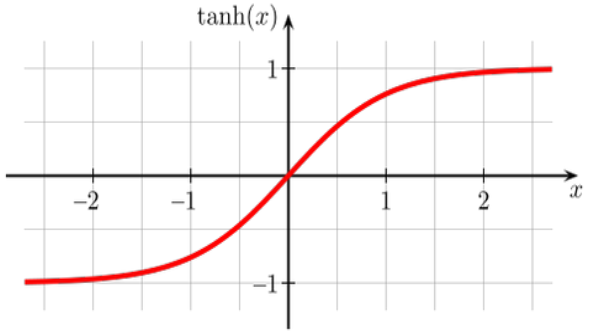
$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

Sigmoid



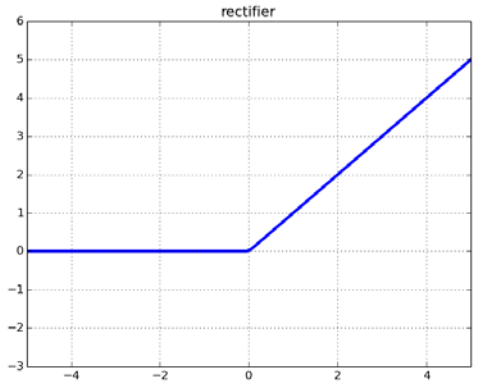
$$f(x) = \frac{1}{1 + e^{-x}}$$

TanH



$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

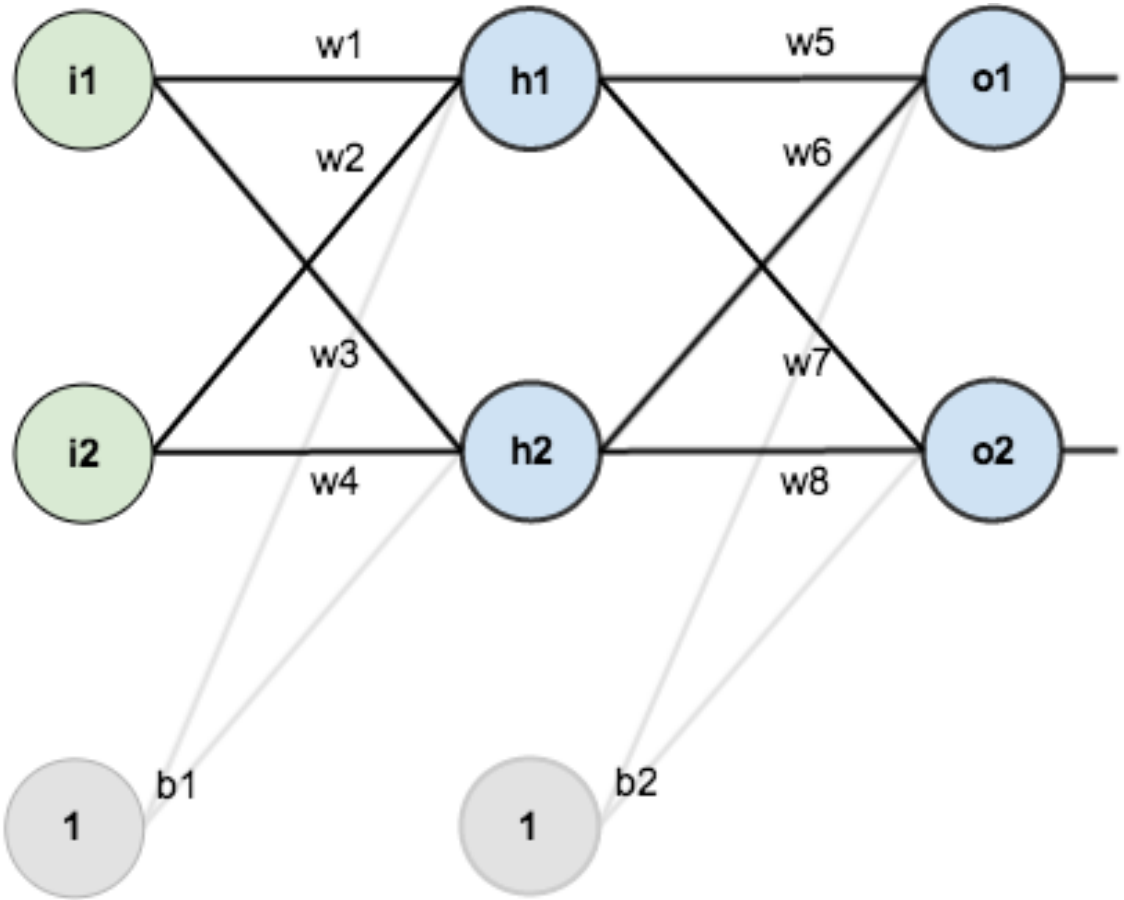
ReLU



$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

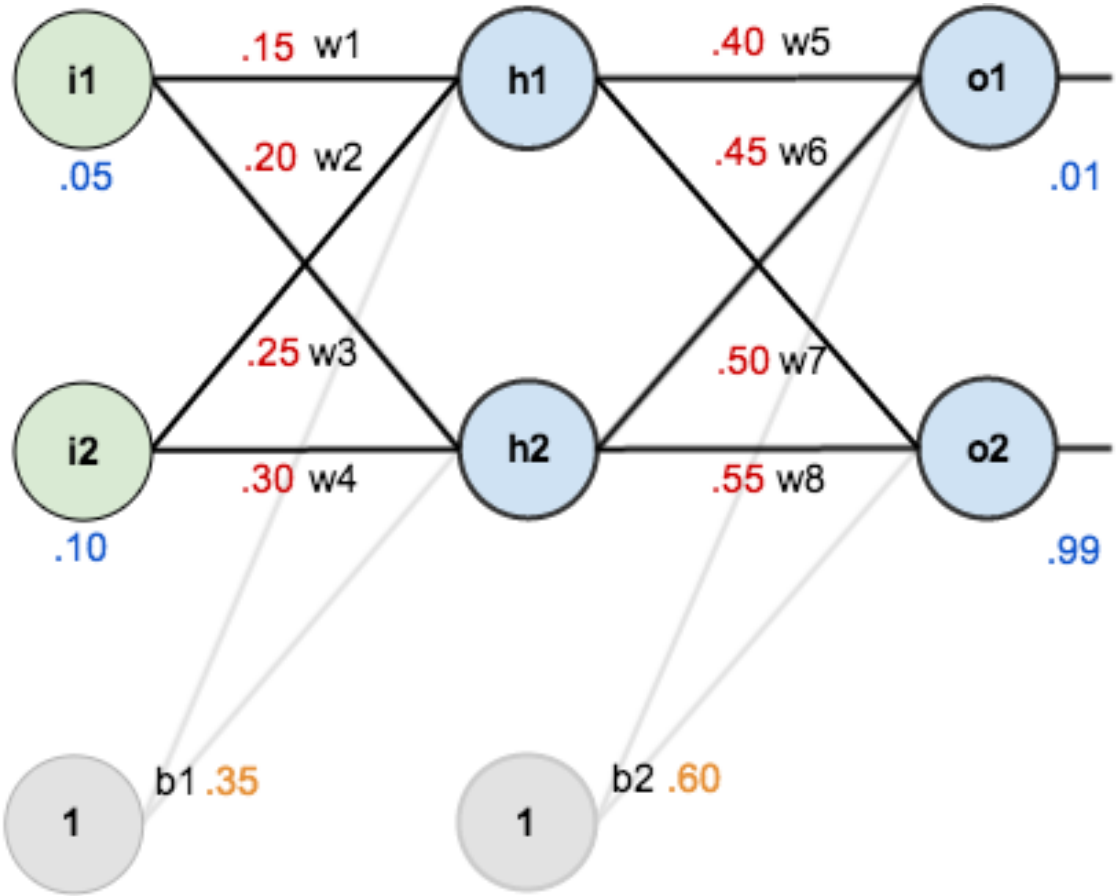
Regular Neural Networks

Multilayer Perceptron



Regular Neural Networks

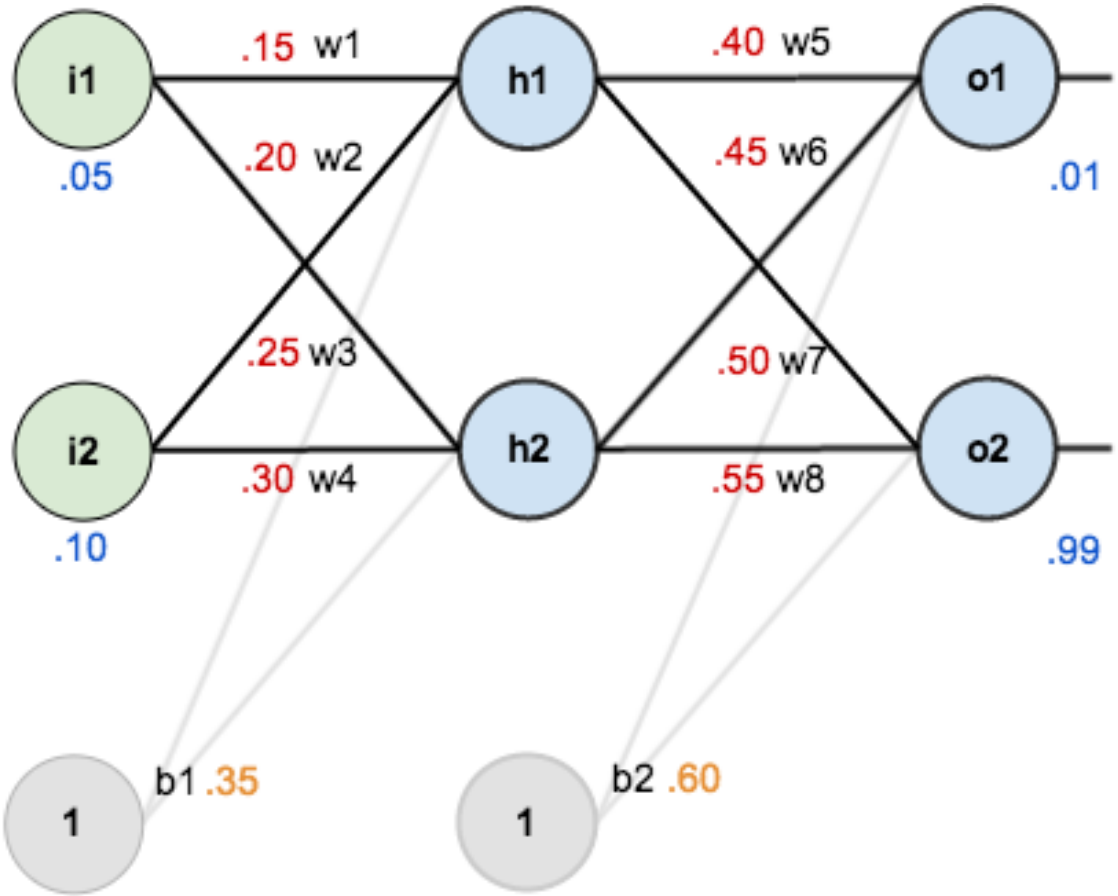
Weights Initialization



Regular Neural Networks

Forward Propagation

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

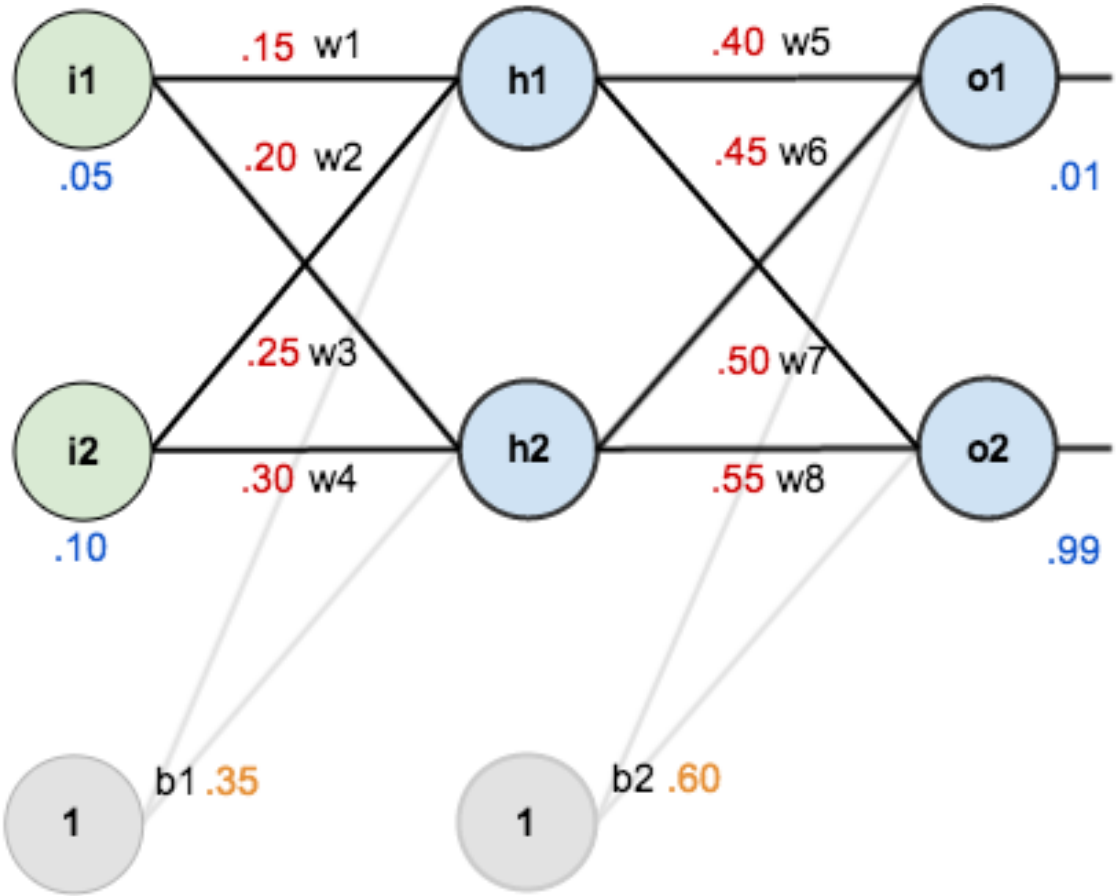


Regular Neural Networks

Forward Propagation

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$



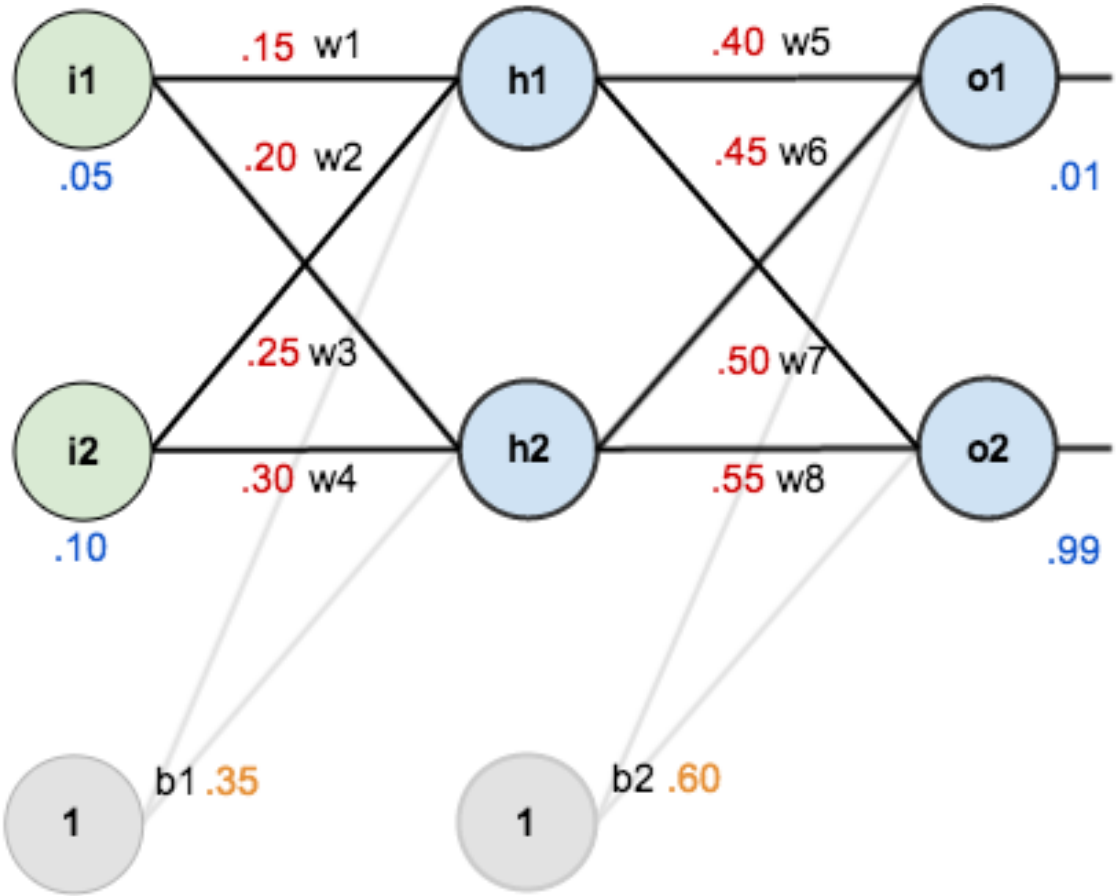
Regular Neural Networks

Forward Propagation

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$



Regular Neural Networks

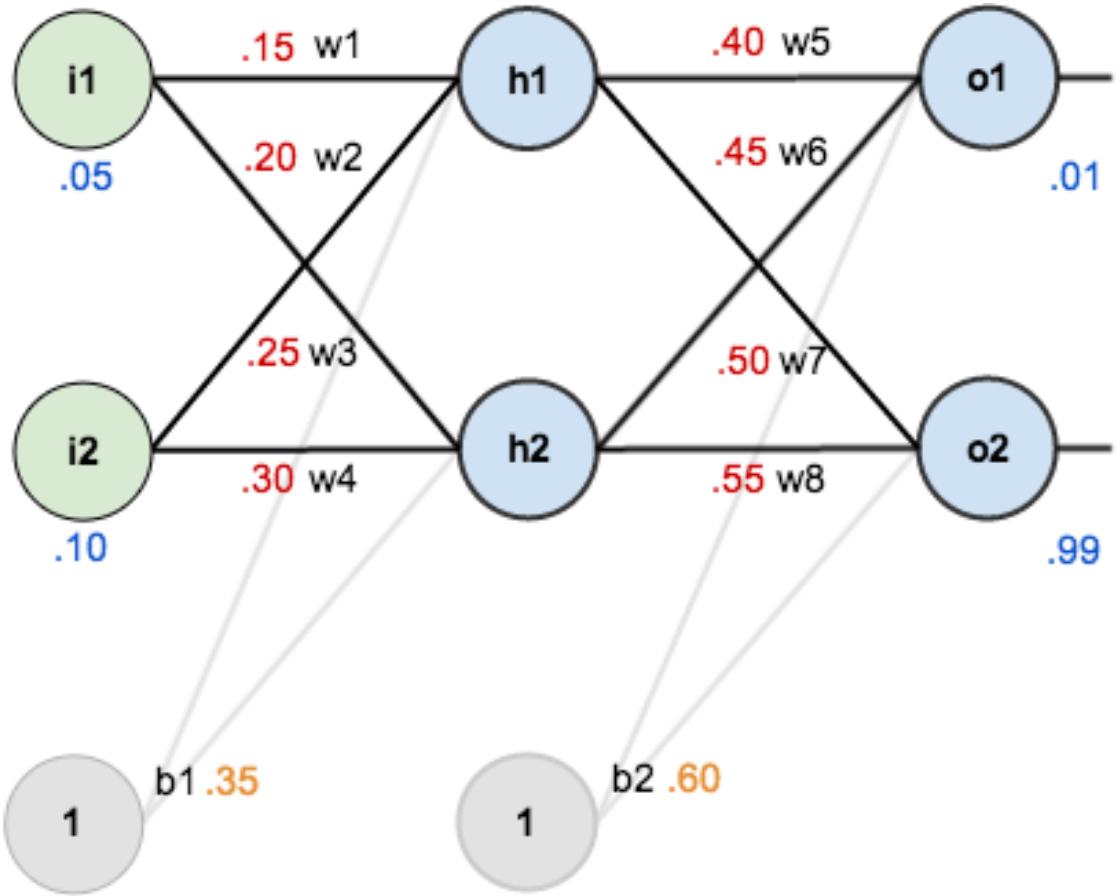
Forward Propagation

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$



Regular Neural Networks

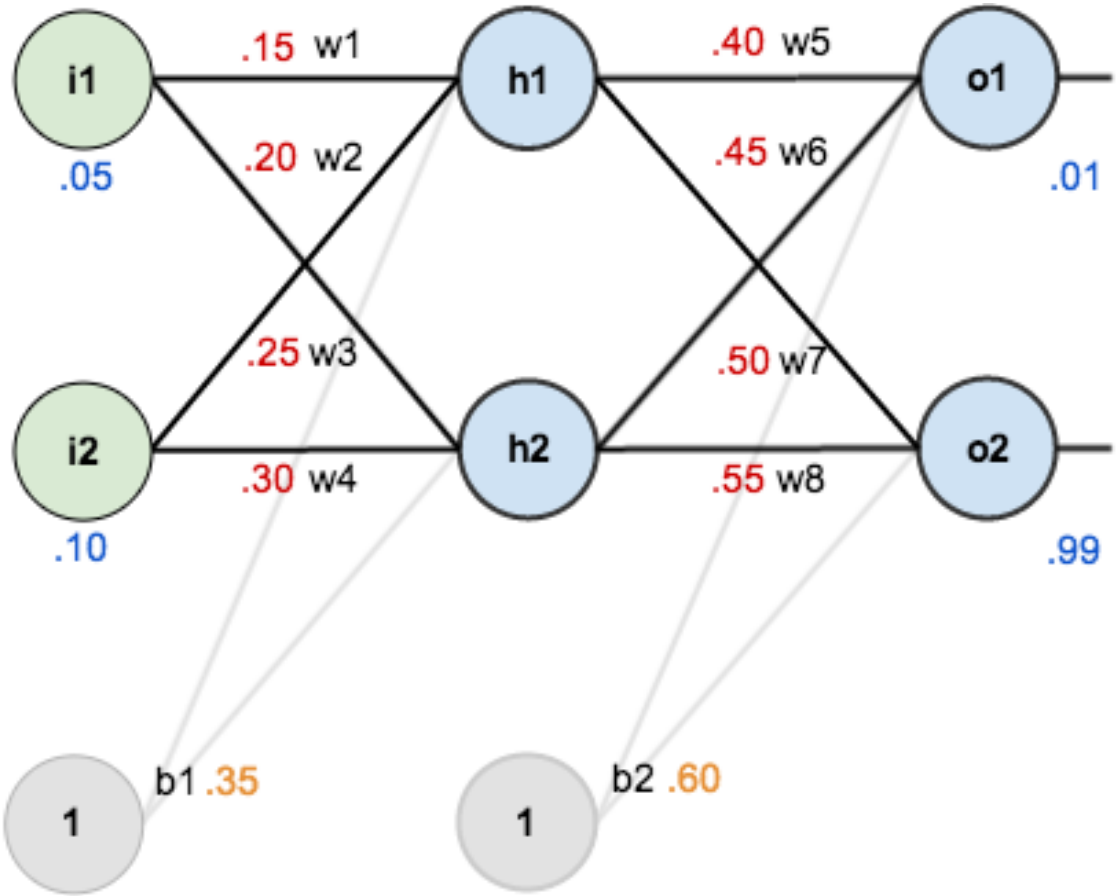
Forward Propagation

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

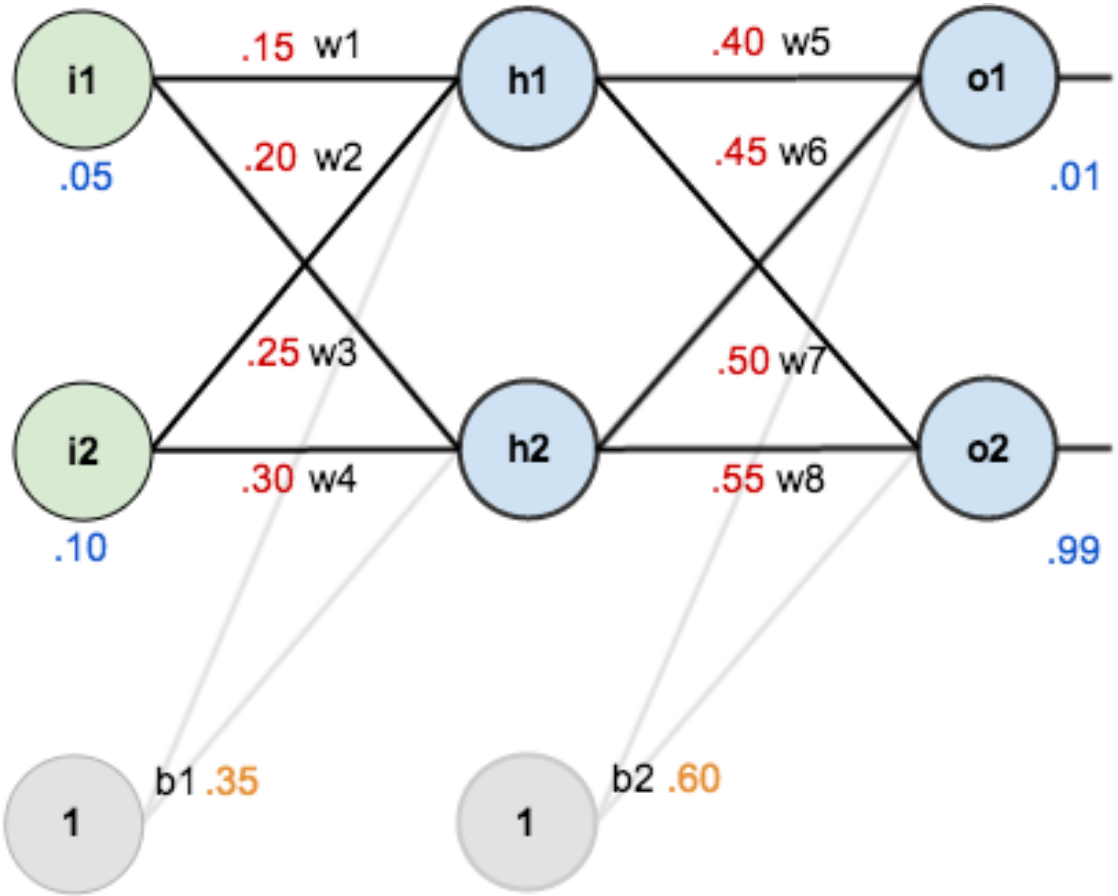
$$out_{o2} = 0.772928465$$

IVOM

Regular Neural Networks

Total Error

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

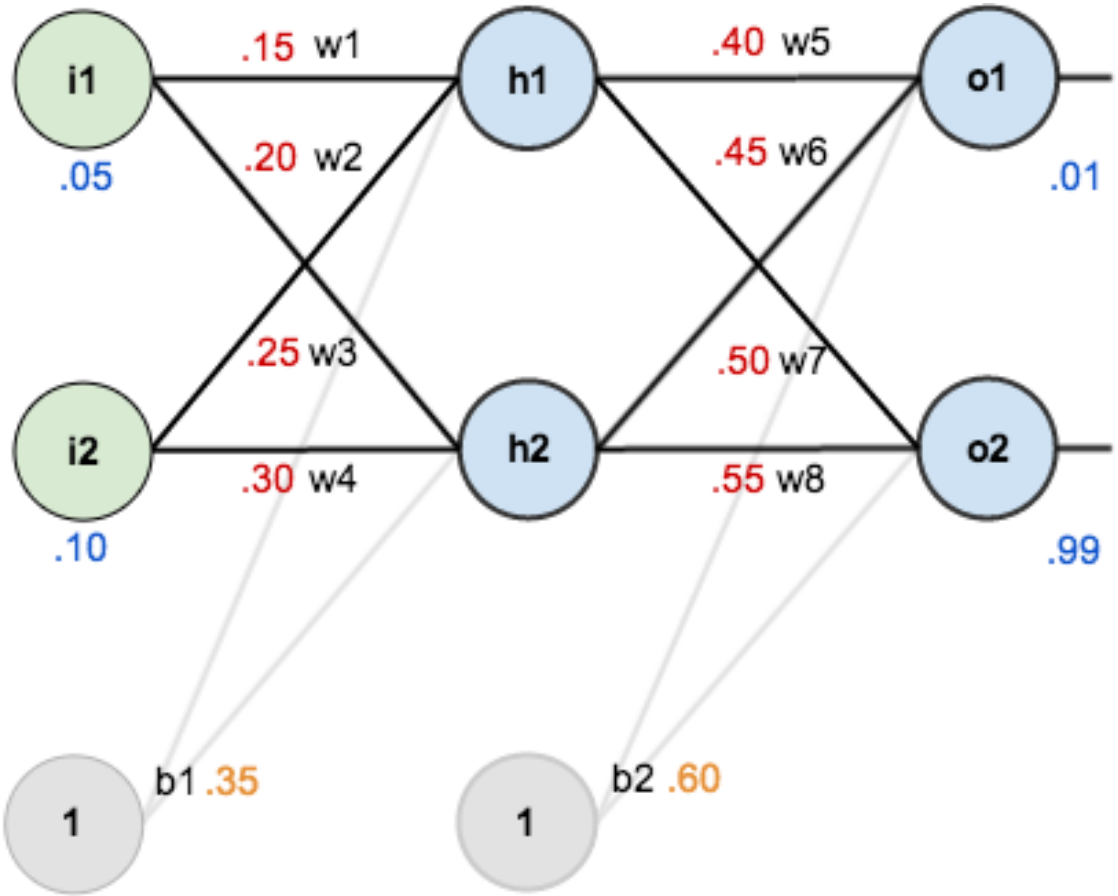


Regular Neural Networks

Total Error

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$



Regular Neural Networks

Total Error

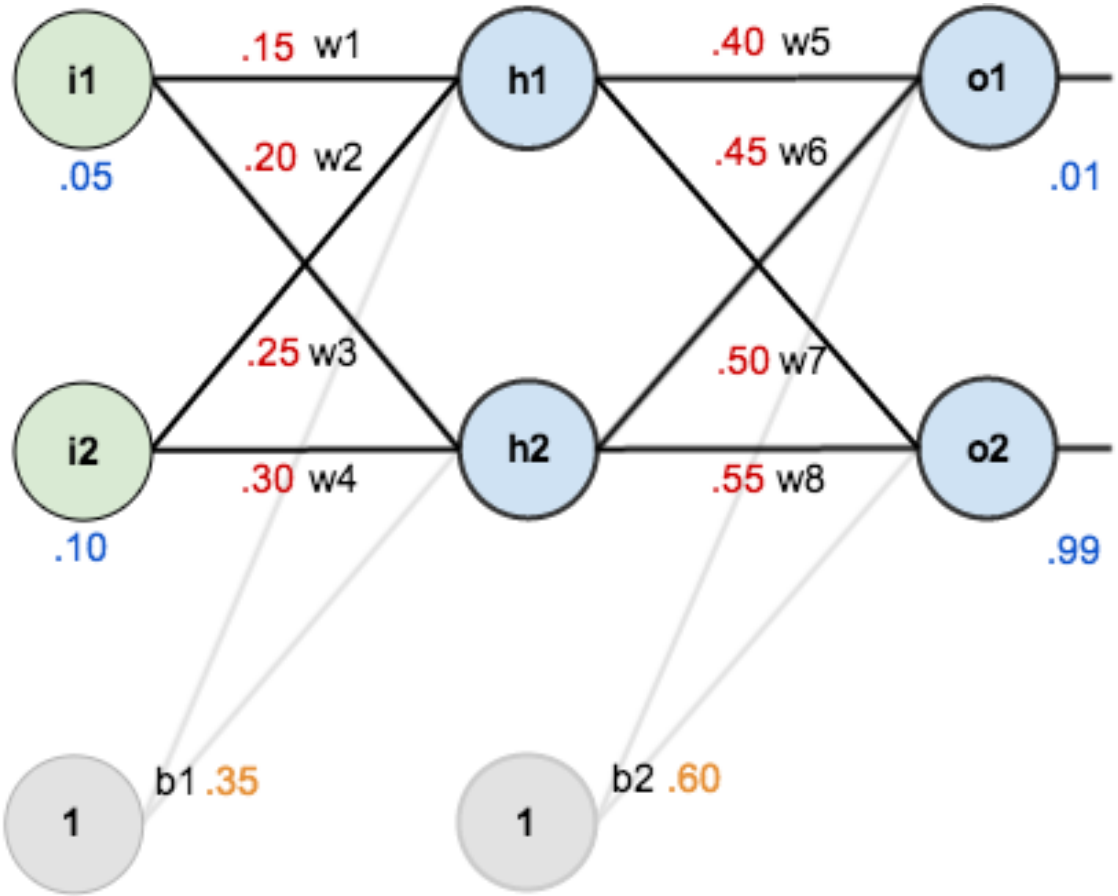
$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

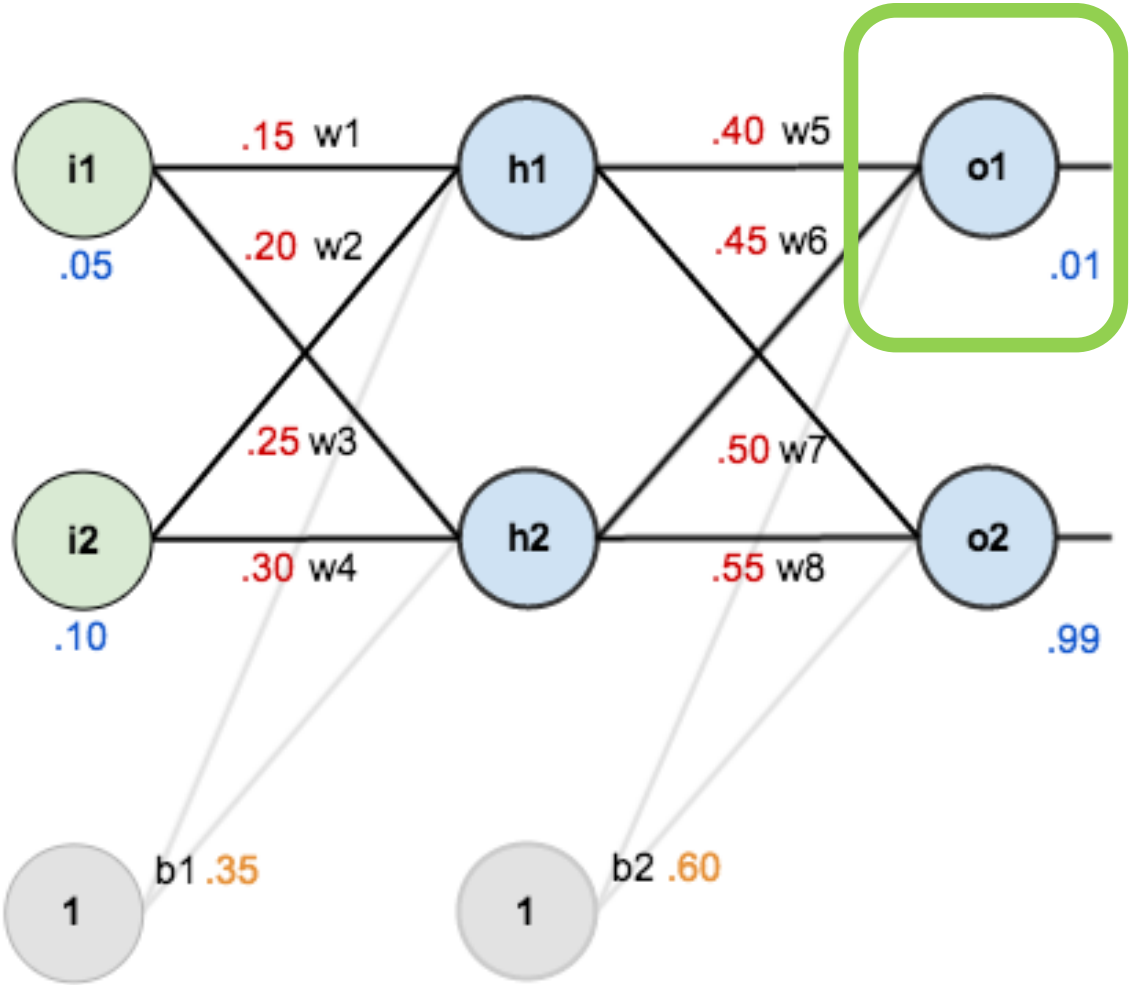
$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$



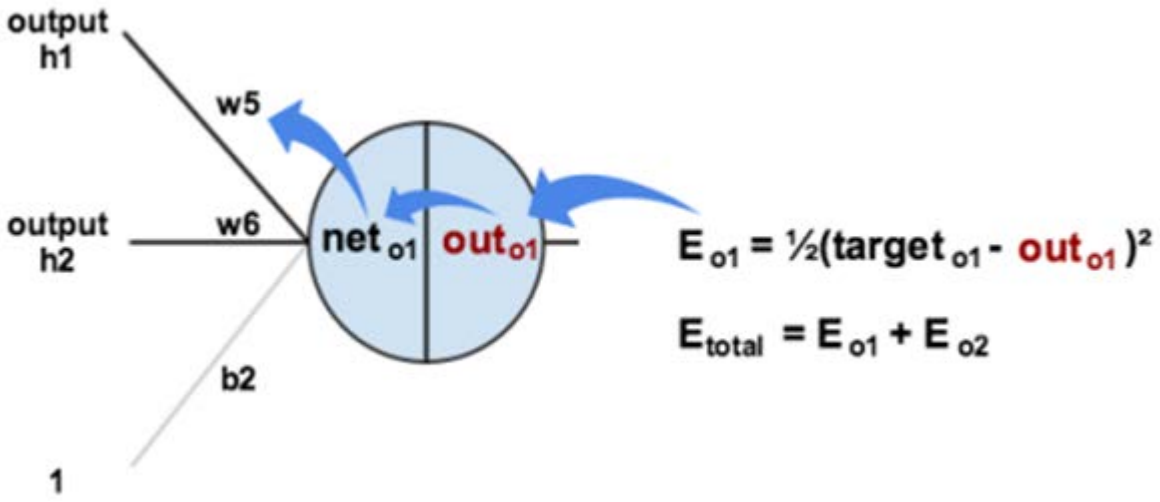
Regular Neural Networks

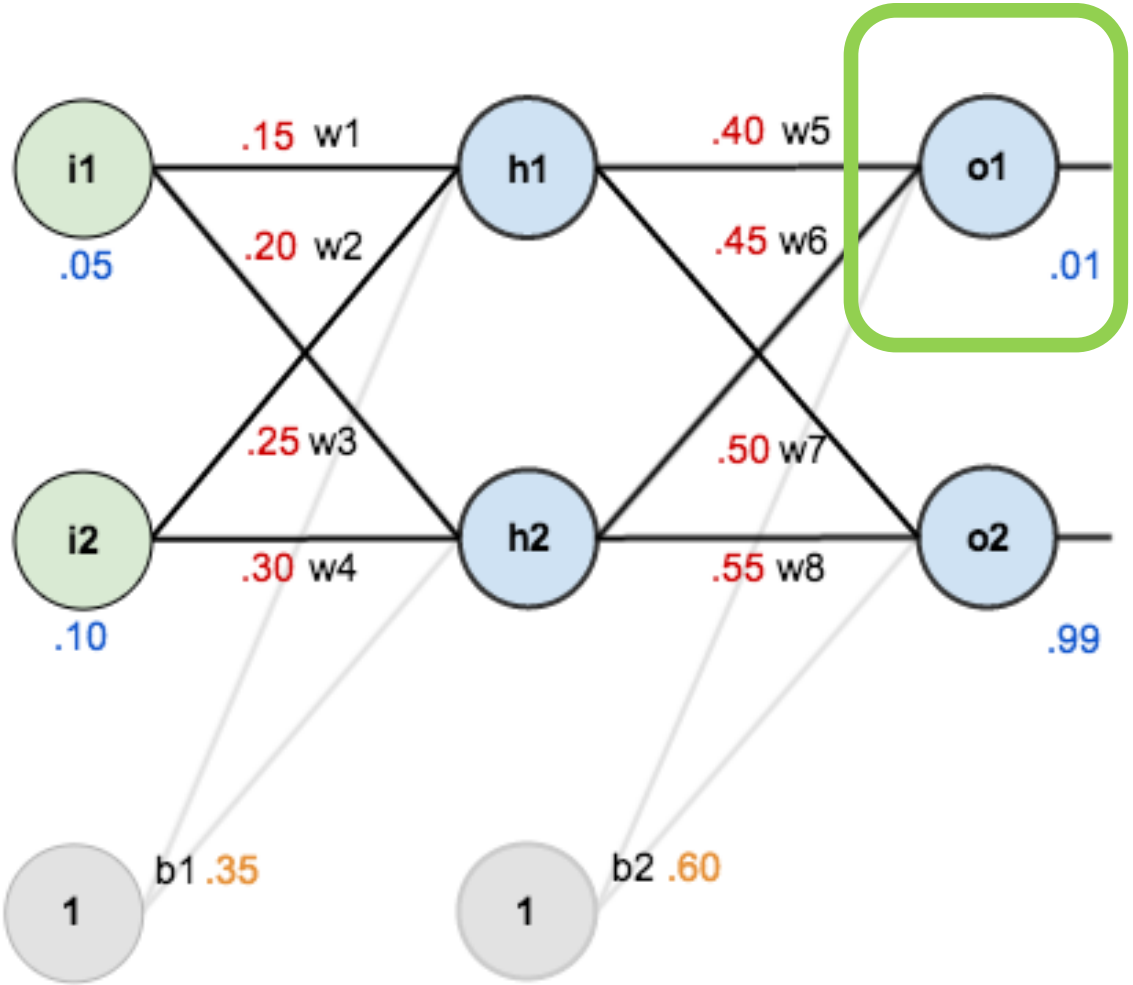
Backward Propagation



Regular Neural Networks

Backward Propagation - Chain Rule

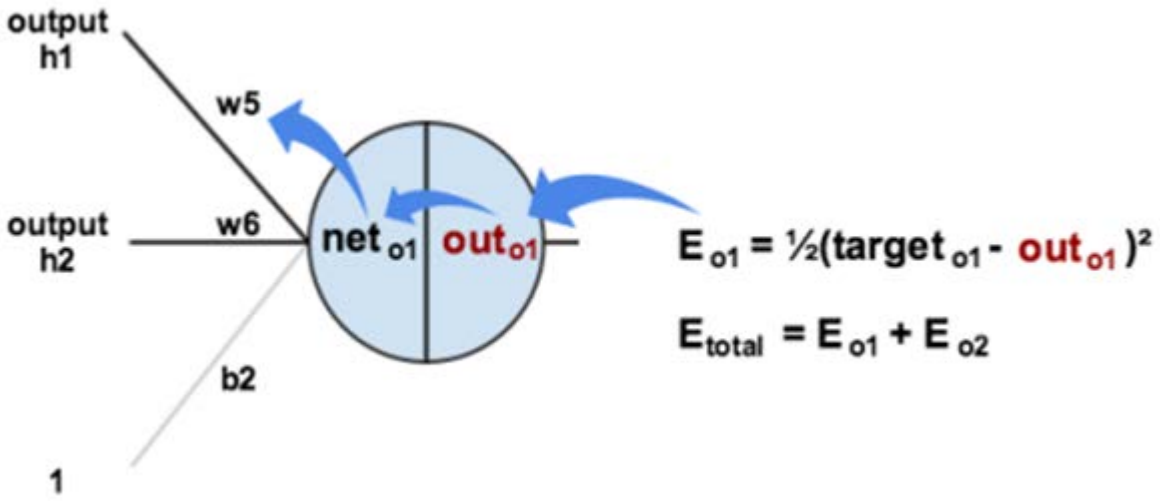




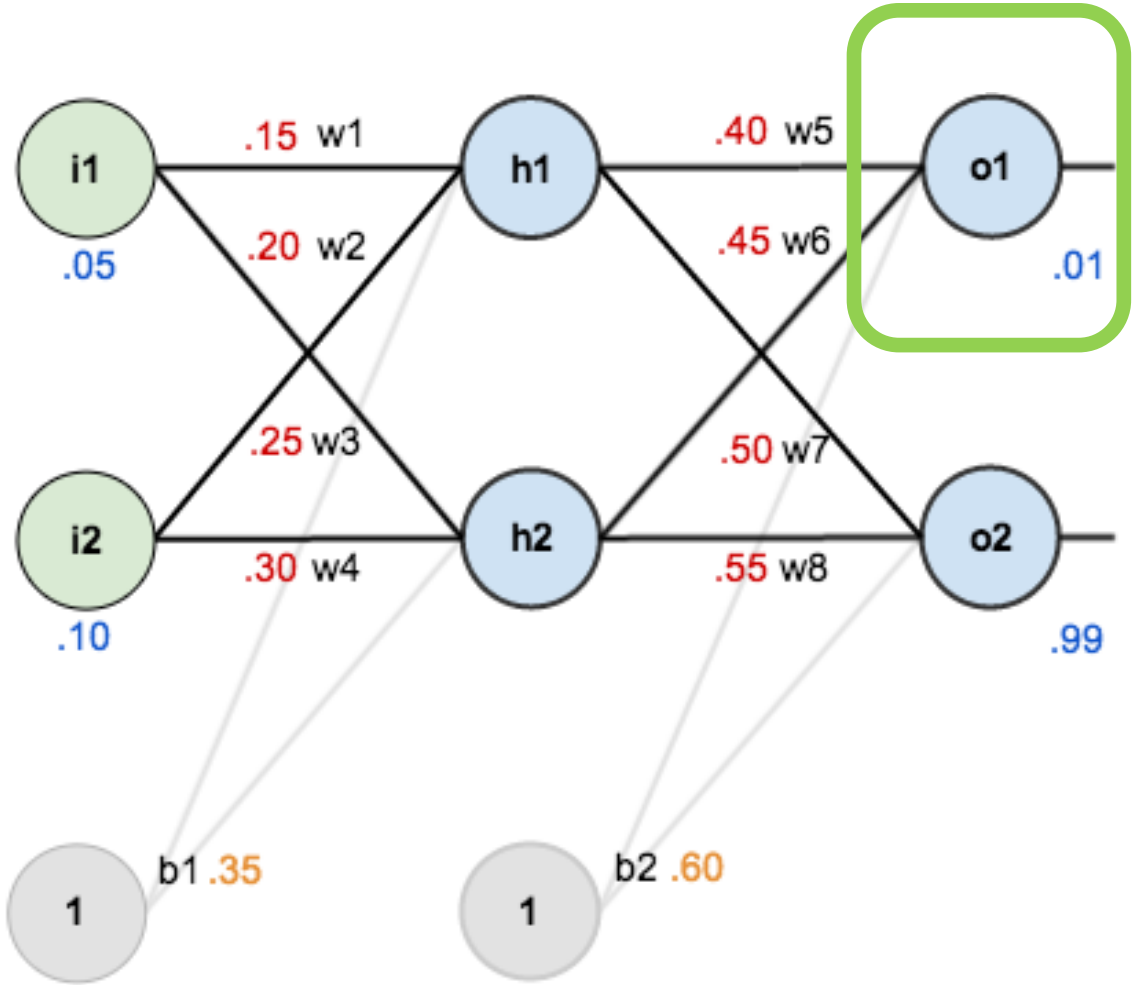
IVOM

Regular Neural Networks

Backward Propagation - Chain Rule



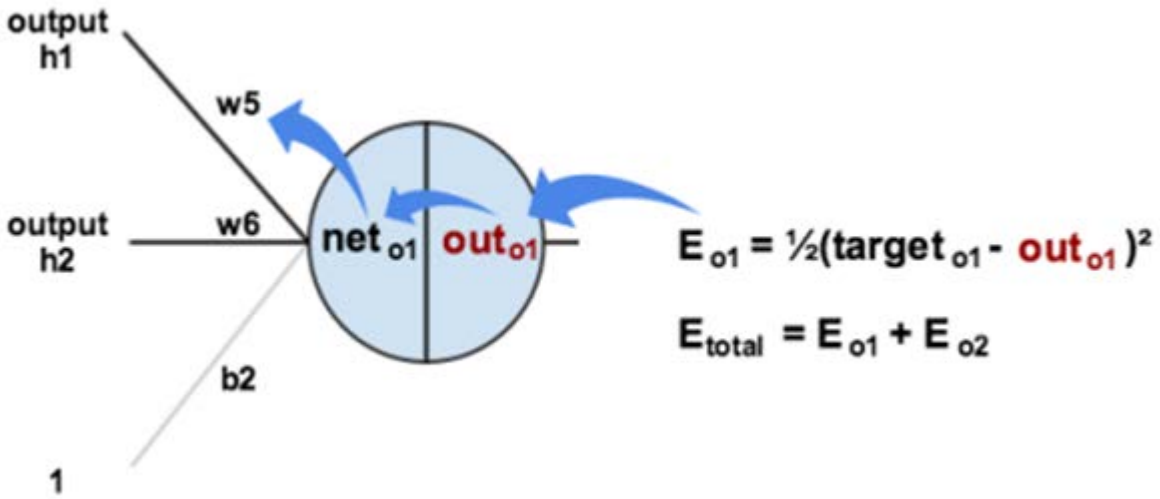
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$



IVOM

Regular Neural Networks

Backward Propagation - Chain Rule



$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

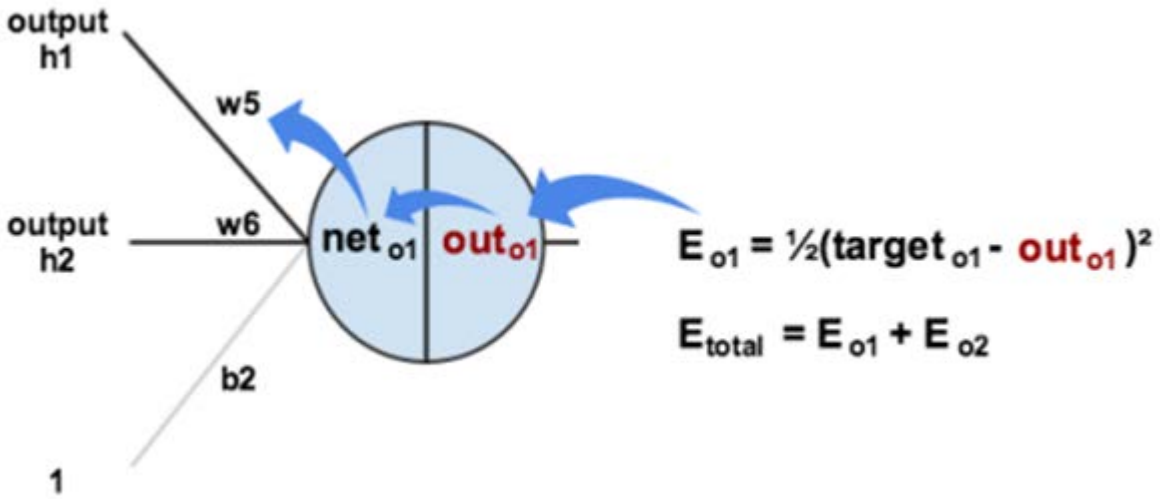
$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Regular Neural Networks

Backward Propagation - Chain Rule



$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

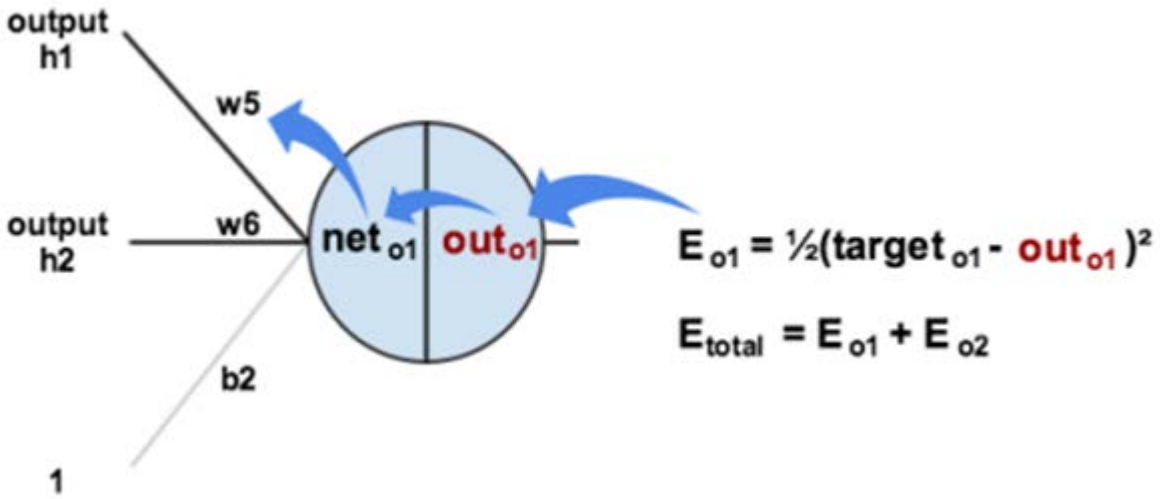
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Regular Neural Networks

Backward Propagation - Chain Rule



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

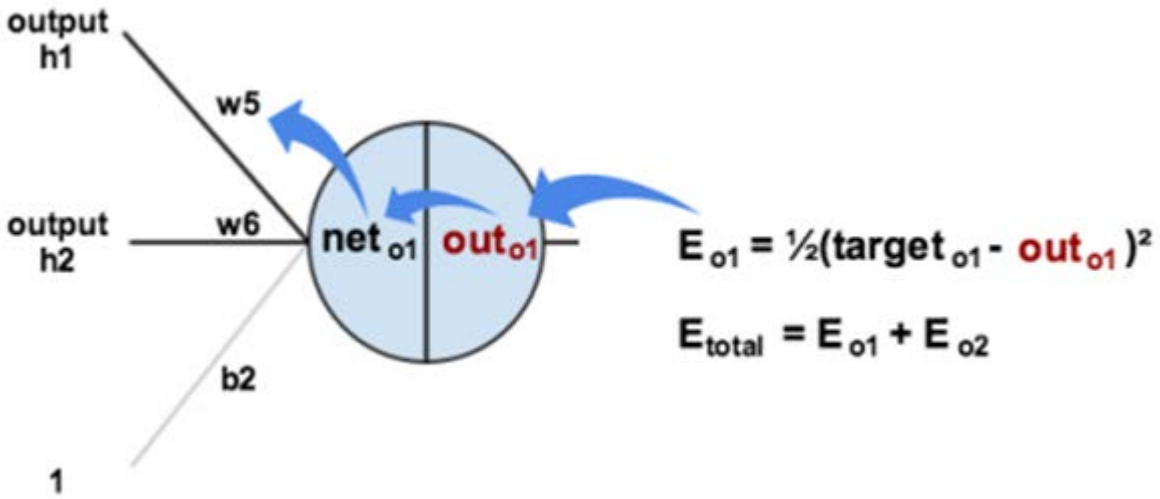
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

IVOM

Regular Neural Networks

Backward Propagation - Chain Rule



$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

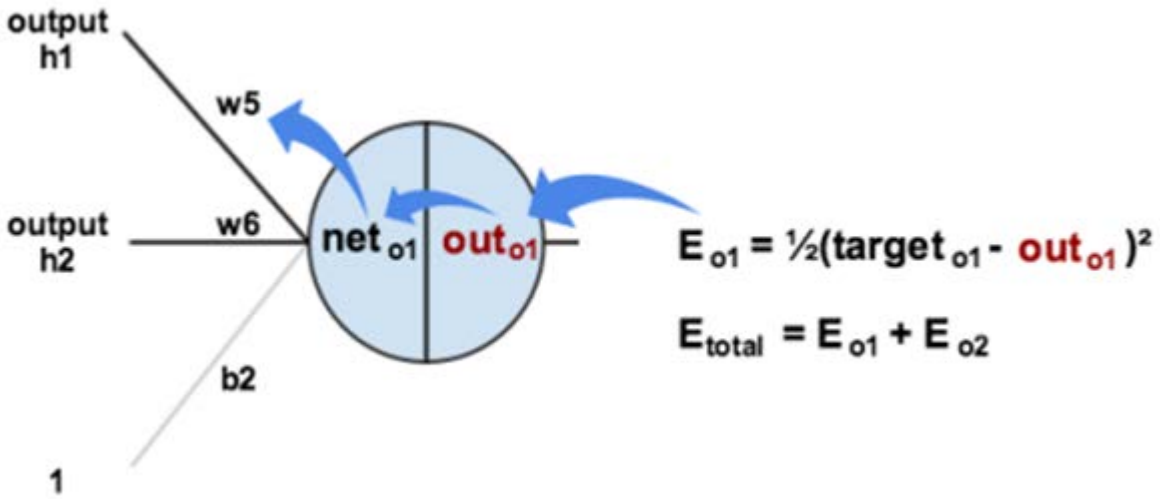
$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

Regular Neural Networks

Backward Propagation - Gradient Descent



To decrease the error, we update the weights by subtracting this value from the current weight (optionally multiplied by some learning rate, eta)

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

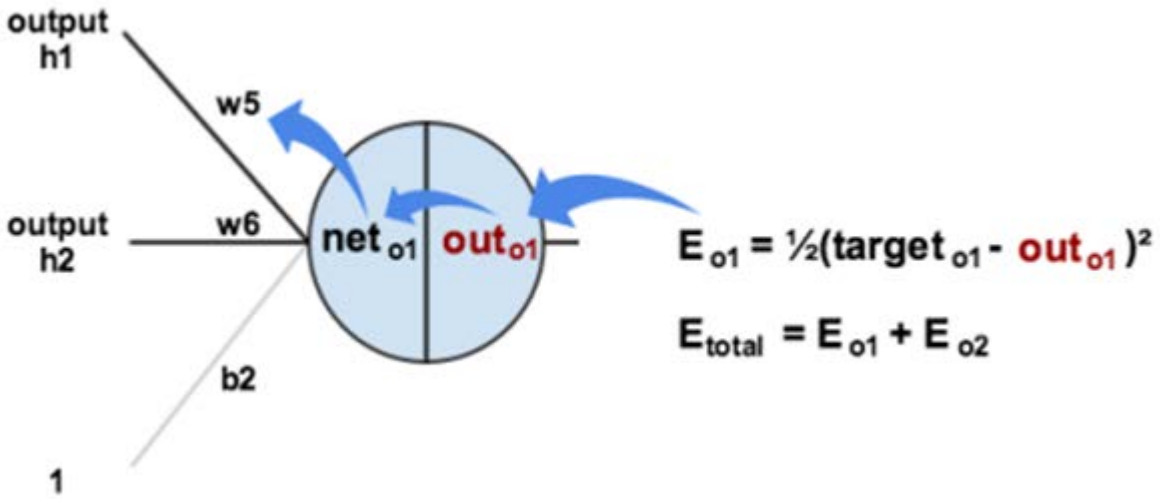
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

IVOM

Regular Neural Networks

Backward Propagation - Gradient Descent



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

To decrease the error, we update the weights by subtracting this value from the current weight (optionally multiplied by some learning rate, eta)

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

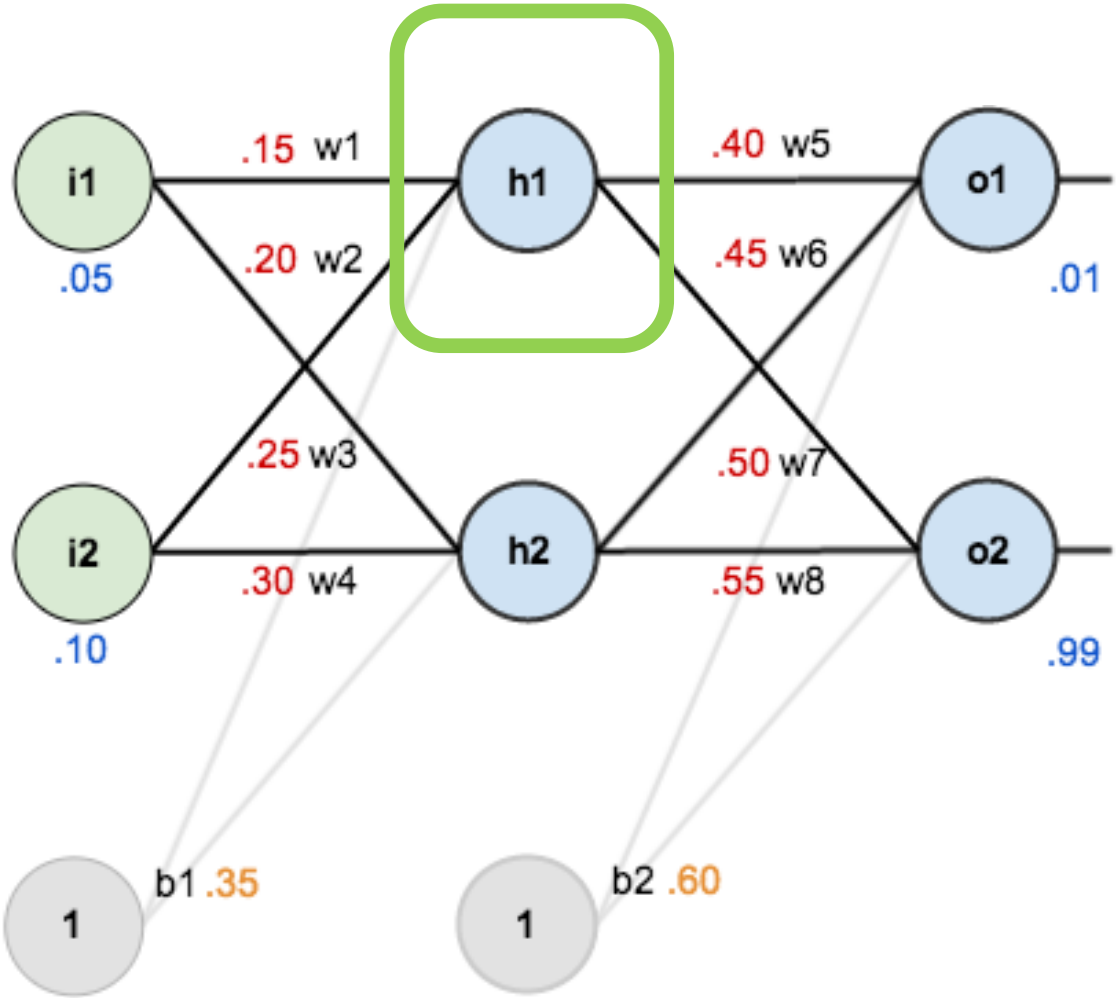
$$w_8^+ = 0.561370121$$

The updates are performed *after* we have the new weights leading into the hidden layer neurons (we use the original weights, not the updated ones, when we compute the derivatives for the other neurons).

IVOM

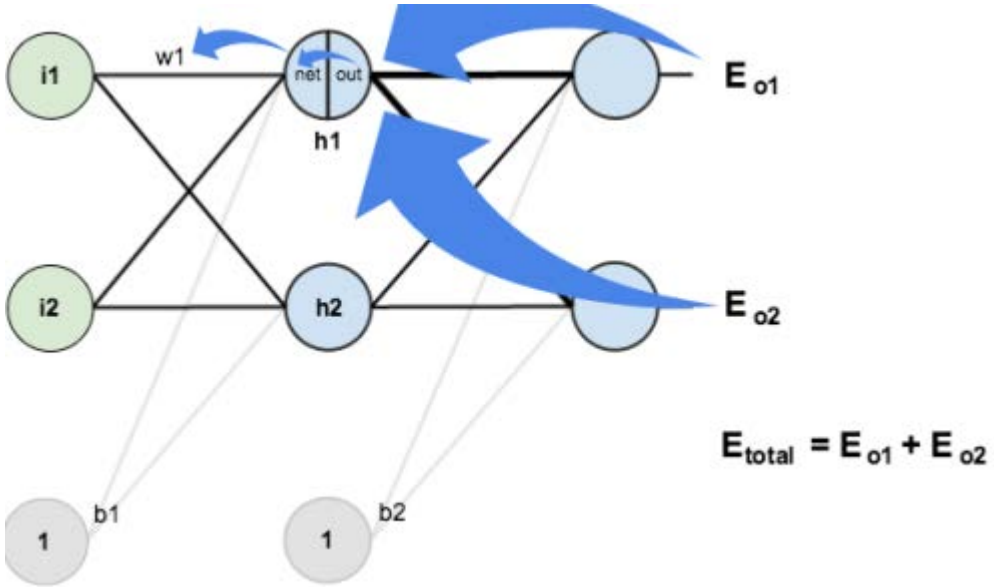
Regular Neural Networks

Backward Propagation - Gradient Descent

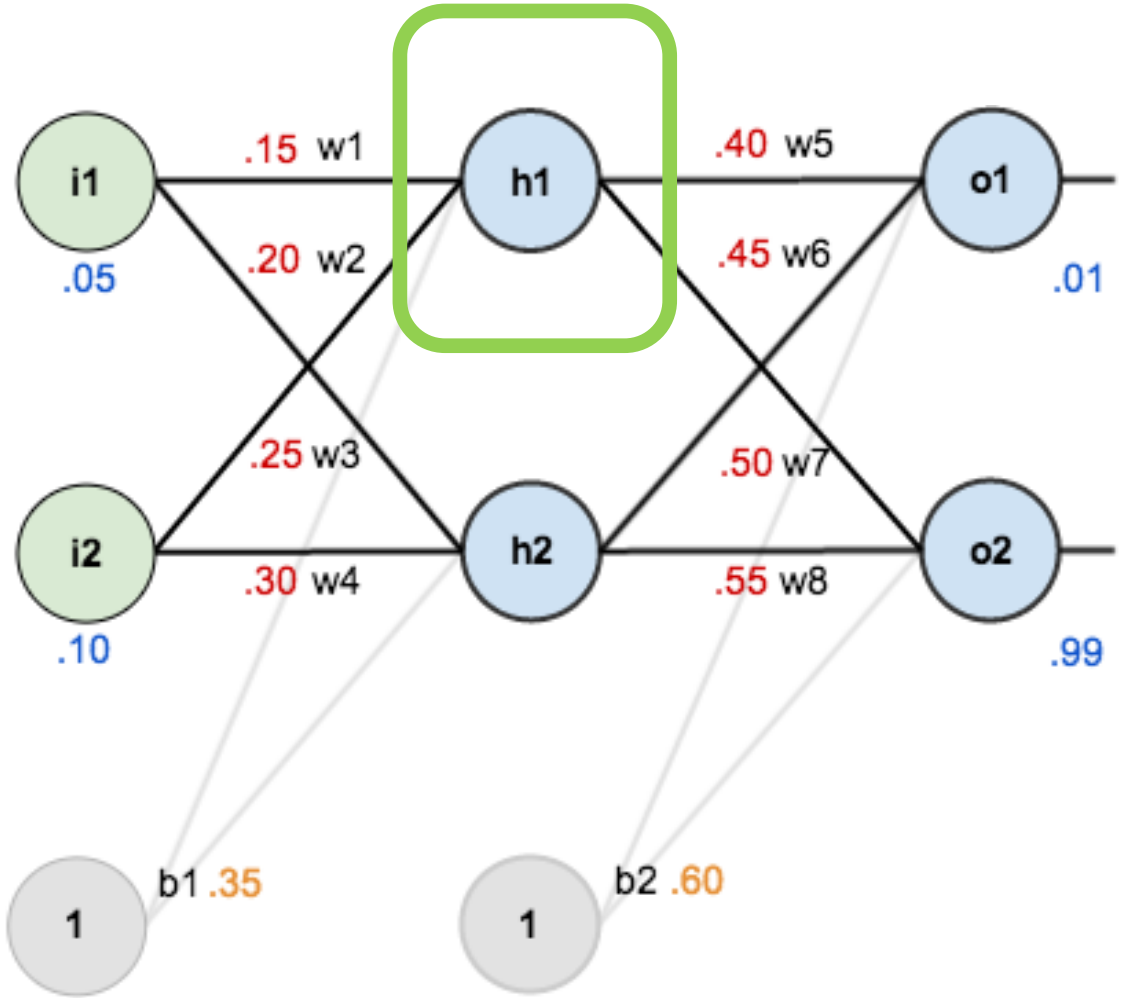


Regular Neural Networks

Backward Propagation - Gradient Descent



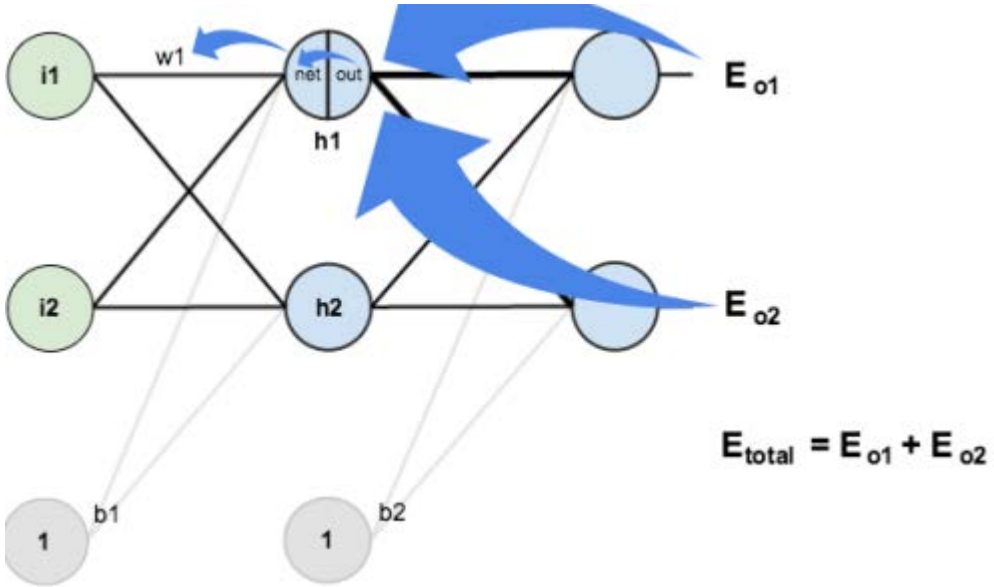
$$E_{total} = E_{o1} + E_{o2}$$



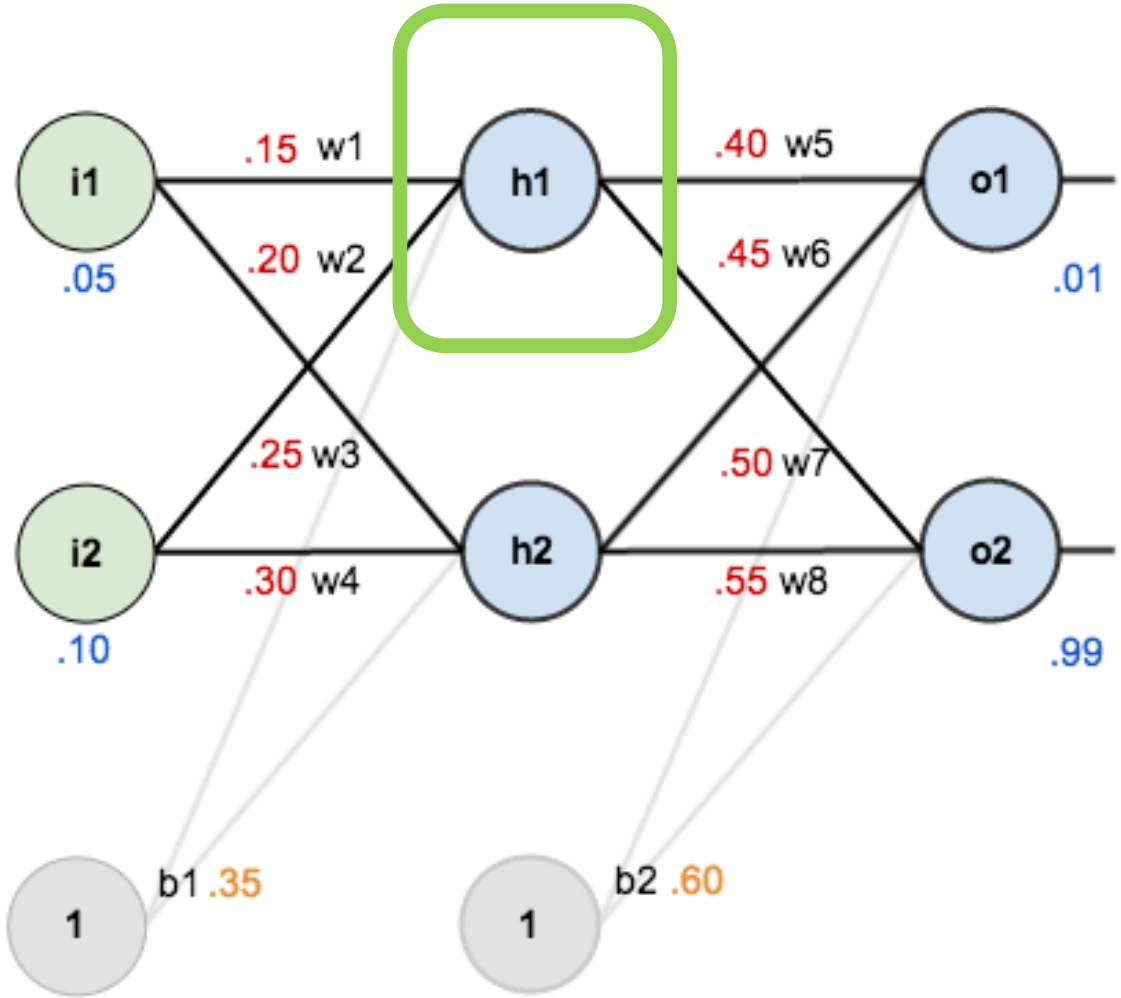
IVOM

Regular Neural Networks

Backward Propagation - Gradient Descent



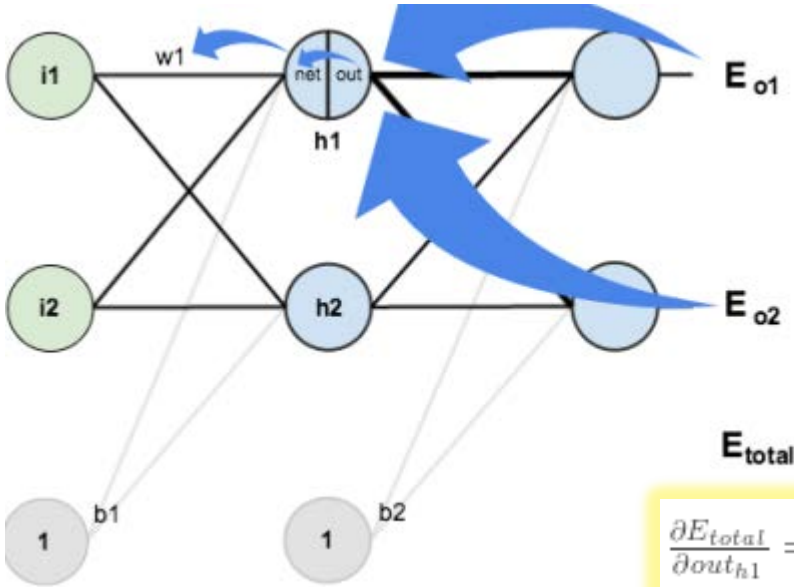
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$



IVOM

Regular Neural Networks

Backward Propagation - Gradient Descent



$$E_{total} = E_{o1} + E_{o2}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

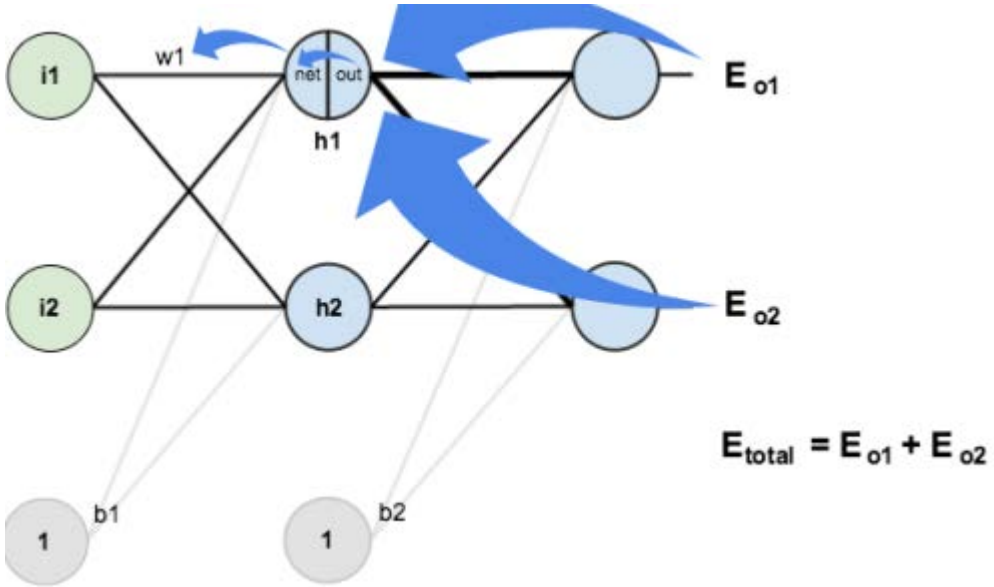
$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

IVOM

Regular Neural Networks

Backward Propagation - Gradient Descent



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

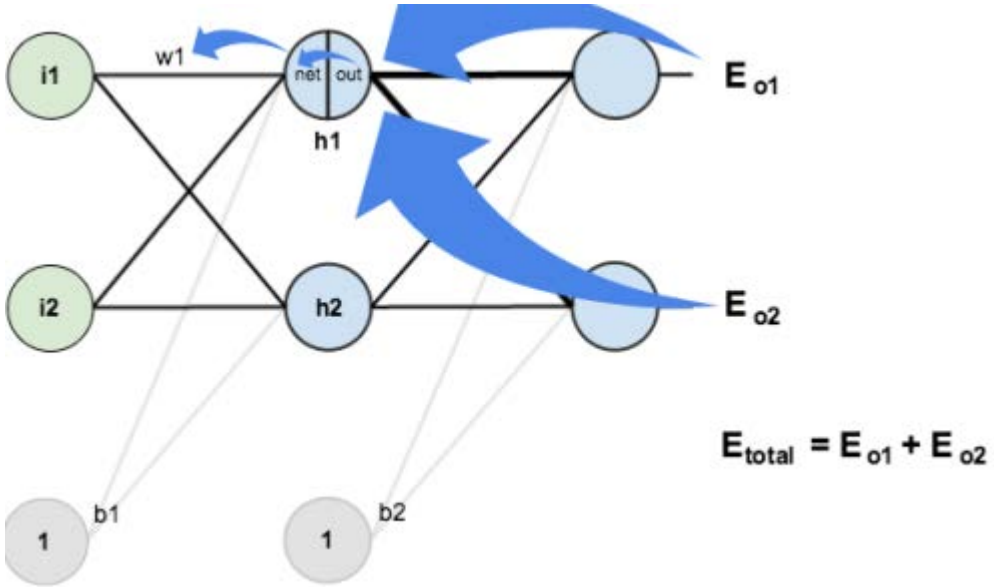
Now we can update the weights

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

IVOM

Regular Neural Networks

Backward Propagation - Gradient Descent



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Now we can update the weights

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

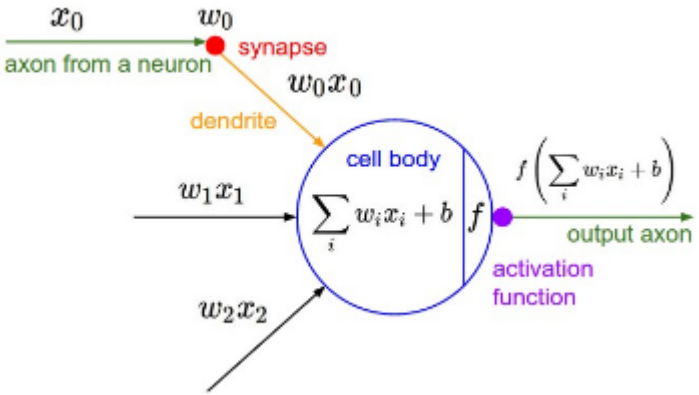
Finally, we've updated all of our weights!
When we fed forward inputs originally, the error on the network was 0.298371109.
After this first round of backpropagation, the total error is now down to 0.291027924.
Repeating this process 10,000 times, for example, the error plummets to 0.000035085.

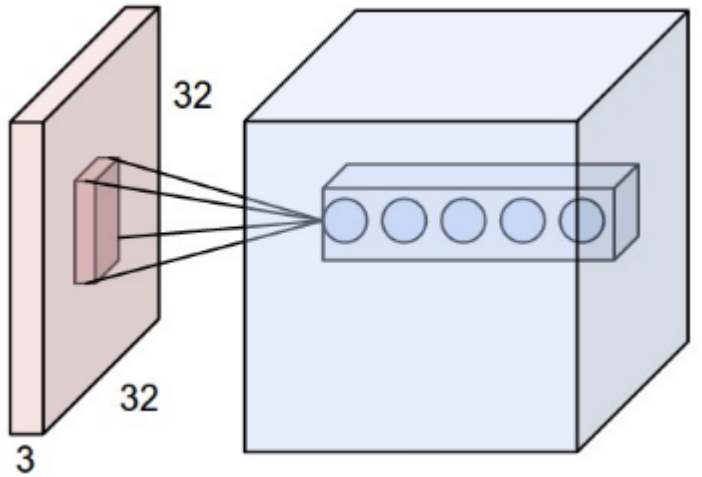
IVOM

Convolutional Neural Networks

CNNs vs. MLP

- 3D Volumes of neurons
  - Neurons in each layer are disposed as a stack of ***D*** feature maps of size ***HxW***
  - 3D volume of **DxHxW** neurons
- Local connectivity
  - Each neuron in layer "*n*" is connected to a small patch of the previous layer: ***receptive field.***
- Weight sharing
  - In each feature map (*depth slice*), all neurons are connected to their respective receptive fields by the ***same set of weights.***



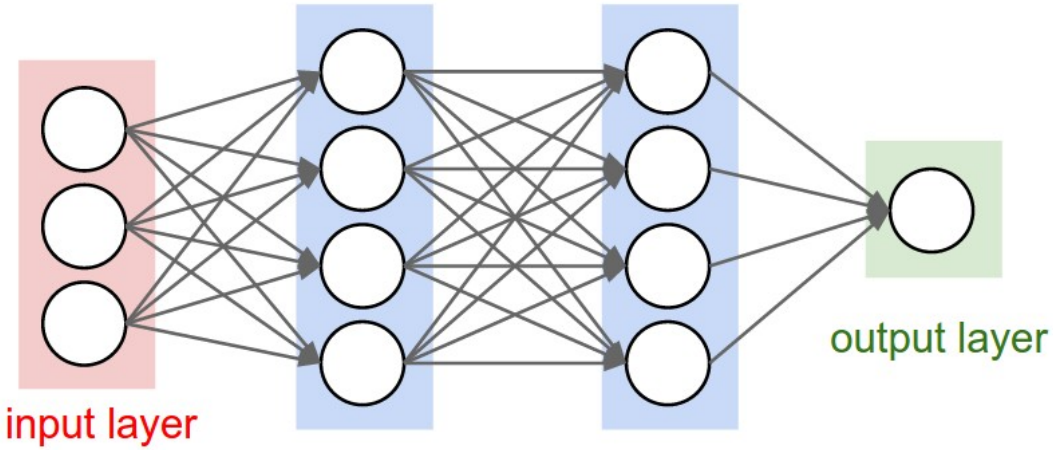


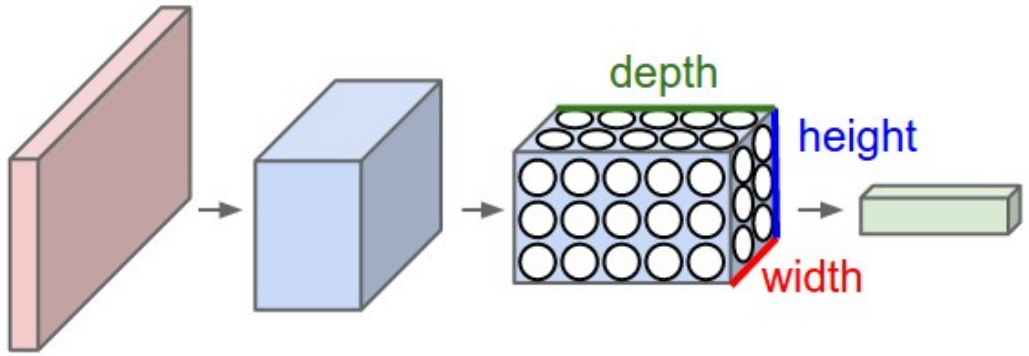
IVOM

Convolutional Neural Networks

CNNs vs. MLP

- 3D Volumes of neurons
  - Neurons in each layer are disposed as a stack of ***D*** feature maps of size ***HxW***
  - 3D volume of **DxHxW** neurons
- Local connectivity
  - Each neuron in layer "*n*" is connected to a small patch of the previous layer: ***receptive field.***
- Weight sharing
  - In each feature map (*depth slice*), all neurons are connected to their respective receptive fields by the ***same set of weights.***

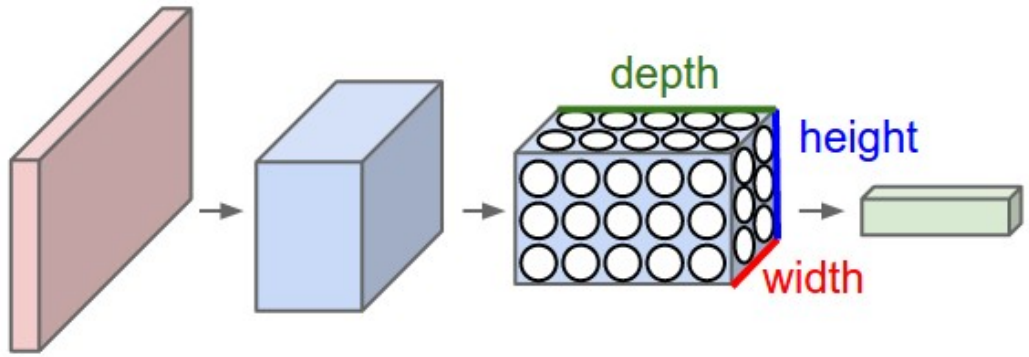




Convolutional Neural Networks

CNN Common Layer Types:

- Input Layers
  - Gather the and data, and eventually ***preprocess*** it.
- Convolutional Layers
  - Performs convolutions between input feature maps and ***learnable*** kernels. The transfer function from ***MLP.***
- Non-linearity Layers
  - Similar to the activation function of the MLP. For medium-large CNNs ***use ReLU!***
- Fully-connected Layers
  - Each neuron in this layer is connected to all the neurons in the previous layer. ***MLP*** legacy layer.
- Pooling Layers
  - Down-sampling of the feature maps of the previous layers along to provide ***translation invariance.***
- Output Layers
  - Computes the ***PDF*** over the space of possible classes.
  - Compute the ***error*** of the network.



IVOM

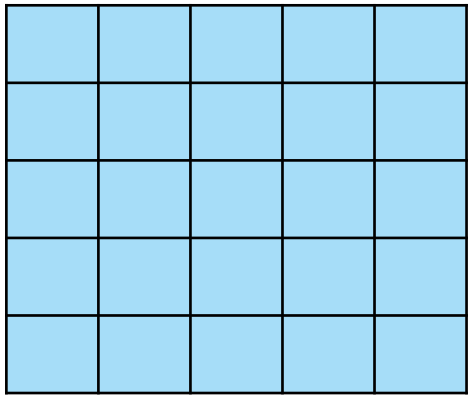
Convolutional Neural Networks

Convolutional Layer (Similar to the transfer function in MLP)

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5

0	0	0	0	0	0	0
0	4	7	5	1	-1	0
0	5	2	5	2	4	0
0	0	1	-6	3	5	0
0	-1	5	0	2	2	0
0	2	2	1	1	3	0
0	0	0	0	0	0	0

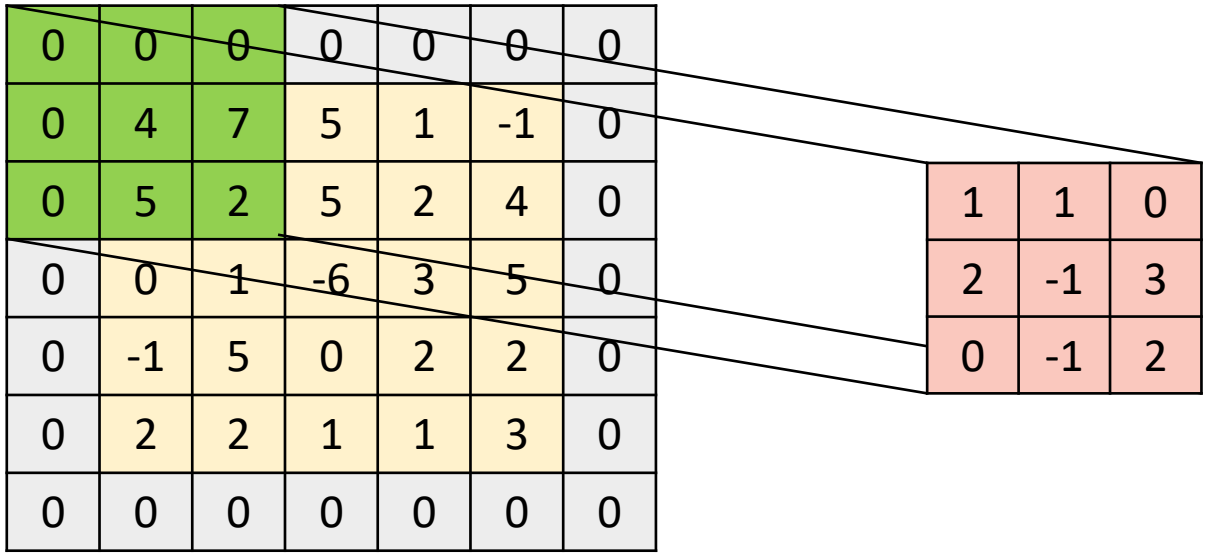
1	1	0
2	-1	3
0	-1	2

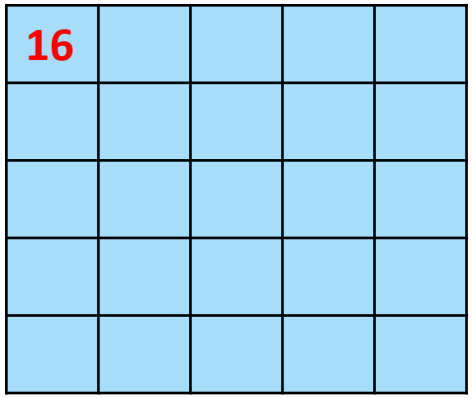


Convolutional Neural Networks

Convolutional Layer

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5

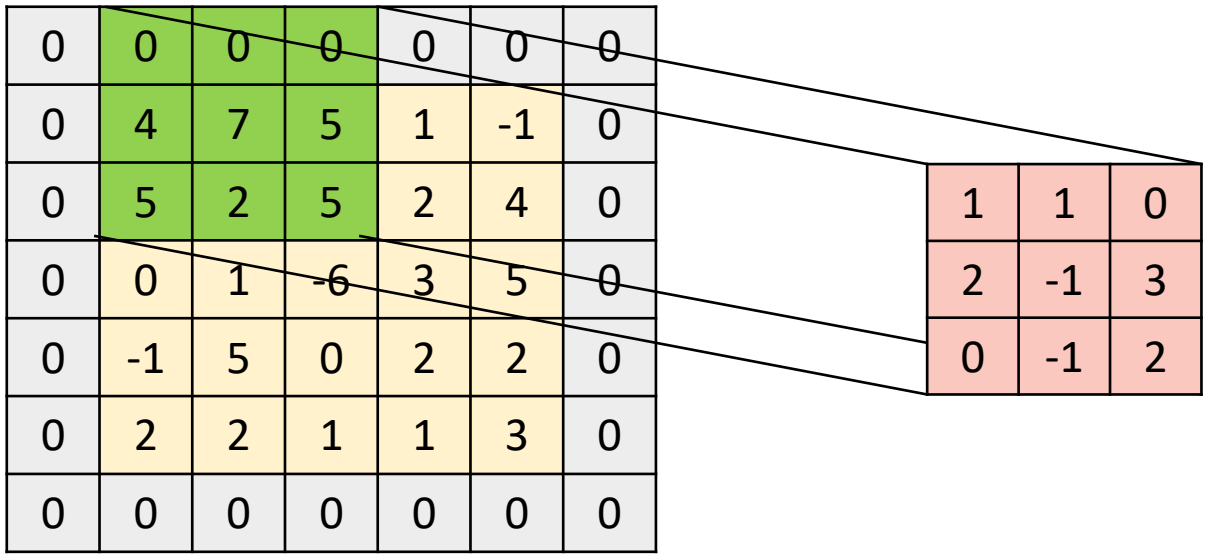


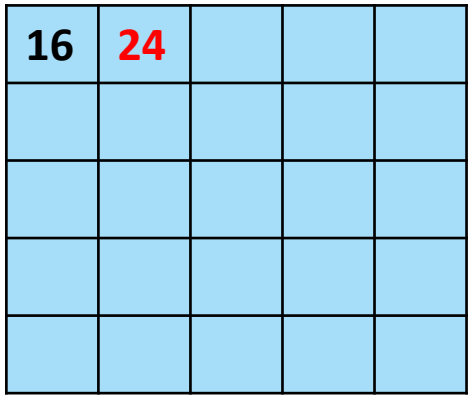


Convolutional Neural Networks

Convolutional Layer

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5

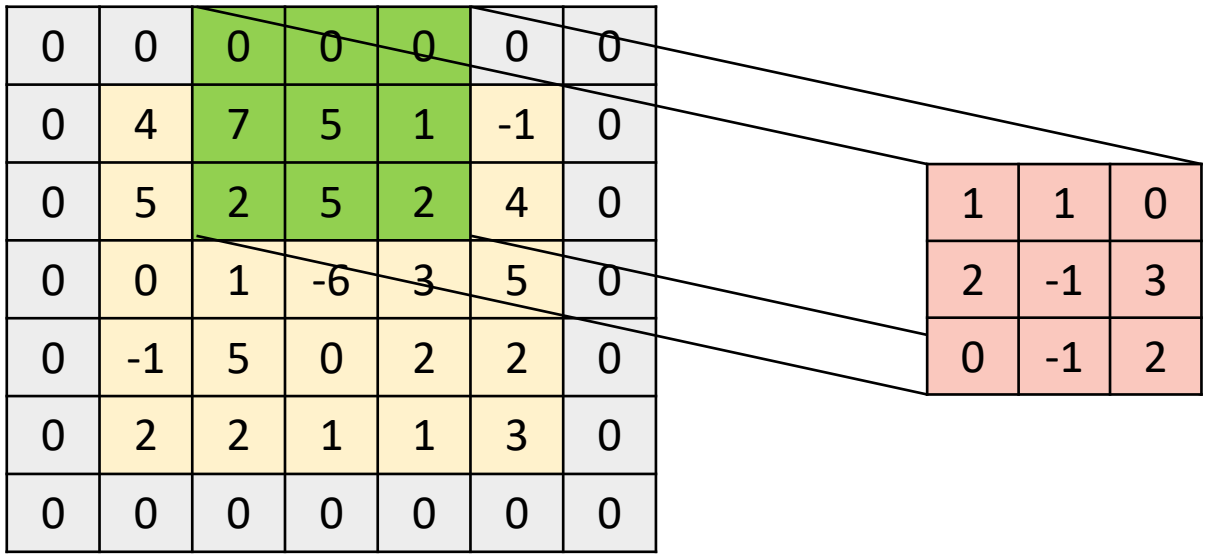




Convolutional Neural Networks

Convolutional Layer

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5



16	**24**	**11**		

Convolutional Neural Networks

Convolutional Layer

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5

0	0	0	0	0	0	0
0	4	7	5	1	-1	0
0	5	2	5	2	4	0
0	0	1	-6	3	5	0
0	-1	5	0	2	2	0
0	2	2	1	1	3	0
0	0	0	0	0	0	0

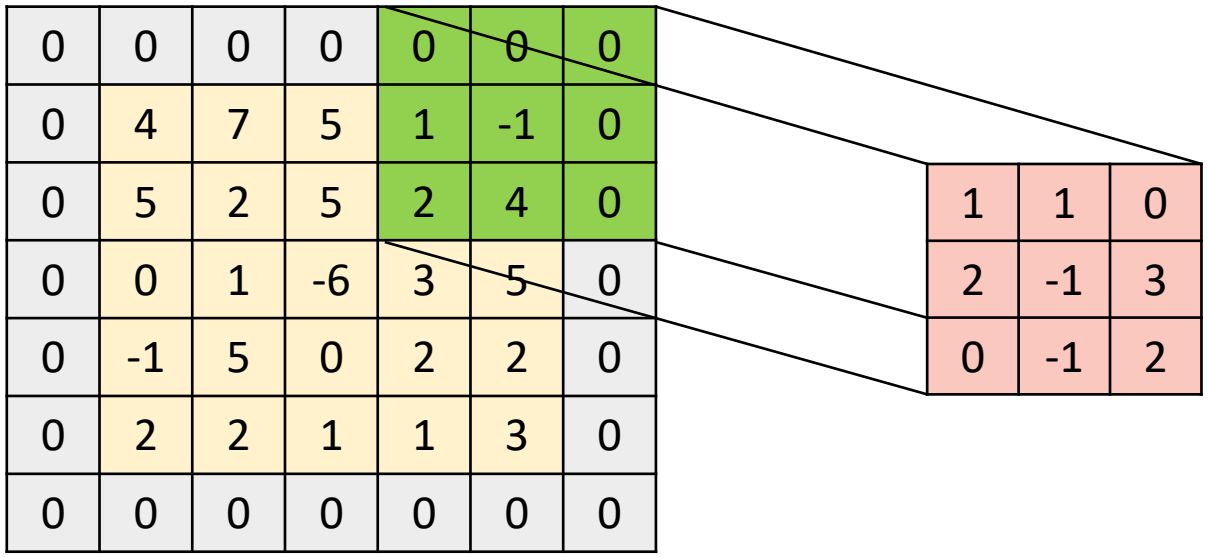
1	1	0
2	-1	3
0	-1	2

16	**24**	**11**	**12**	

Convolutional Neural Networks

Convolutional Layer

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5

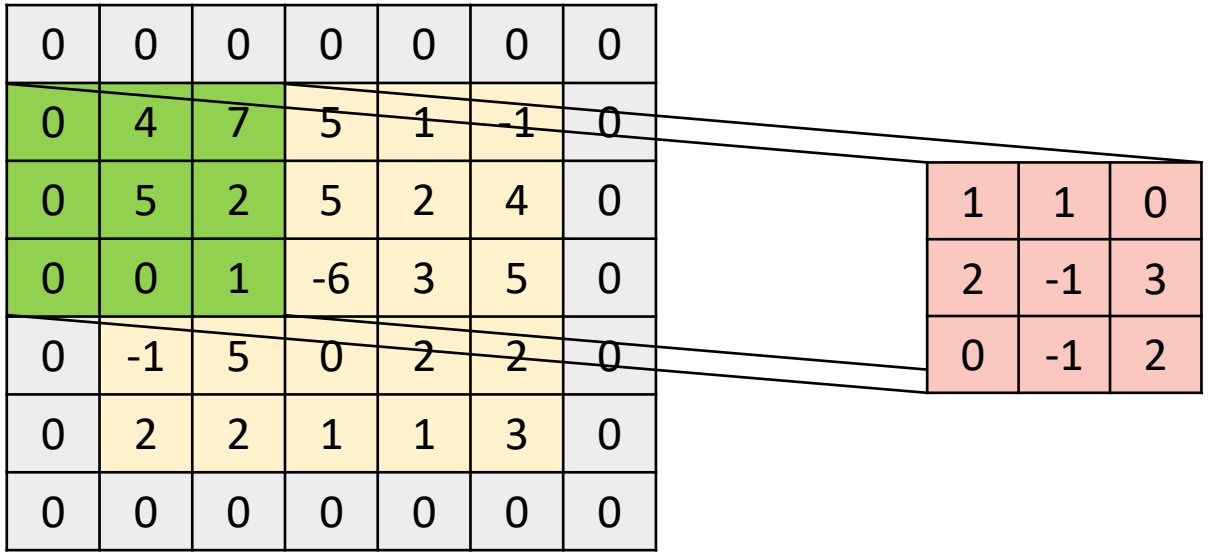


16	**24**	**11**	**12**	**-1**

Convolutional Neural Networks

Convolutional Layer

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5



16	**24**	**11**	**12**	**-1**
7				

Convolutional Neural Networks

Convolutional Layer

Padded Input Map: 7x7
Kernel Size: 3x3
Output: 5x5

0	0	0	0	0	0	0
0	4	7	5	1	-1	0
0	5	2	5	2	4	0
0	0	1	-6	3	5	0
0	-1	5	0	2	2	0
0	2	2	1	1	3	0
0	0	0	0	0	0	0

1	1	0
2	-1	3
0	-1	2

16	**24**	**11**	**12**	**-1**
7	**21**	**29**	**33**	**-5**
19	**-17**	**28**	**9**	**5**
18	**-6**	**12**	**6**	**7**
3	**9**	**11**	**12**	**3**

Convolutional Neural Networks

Non-linearity Layer (The same as the activation function in MLP)

Apply a non-linear function (activation function) to each element of the input feature maps.

- **ReLU** (Rectified Linear Unit) as activation function.
- **Sigmoid**
- **TanH**

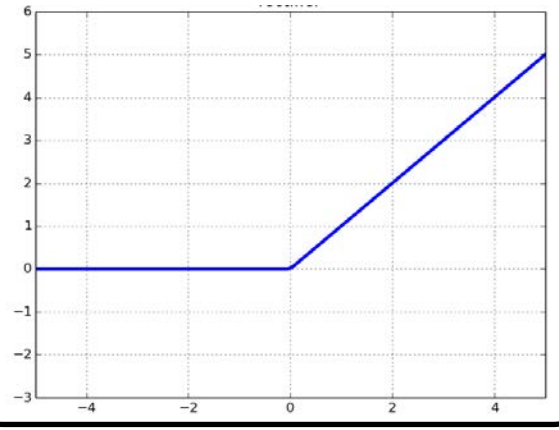
Input Map

15	5	2.5
8	-5	-4
-7	-1	0
-2	2	-2
1	3	4

ReLU
Non-linearity

Output Map

ReLU $f(x) = \max(0, x)$



Convolutional Neural Networks

Non-linearity Layer

Apply a non-linear function (activation function) to each element of the input feature maps.

- **ReLU** (Rectified Linear Unit) as activation function.
- **Sigmoid**
- **TanH**

Input Map

15	5	2.5
8	-5	-4
-7	-1	0
-2	2	-2
1	3	4

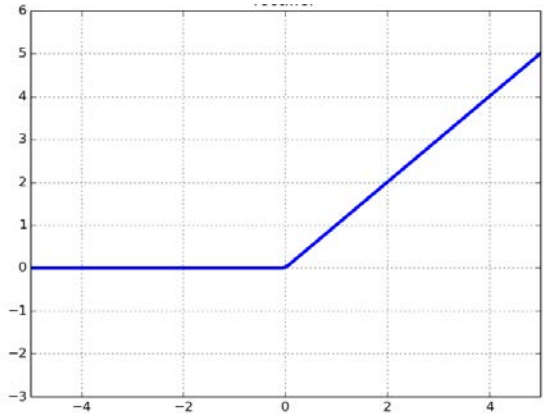
ReLU
Non-linearity



Output Map

15	5	2.5
8	0	0
0	0	0
0	2	0
1	3	4

ReLU $f(x) = \max(0, x)$

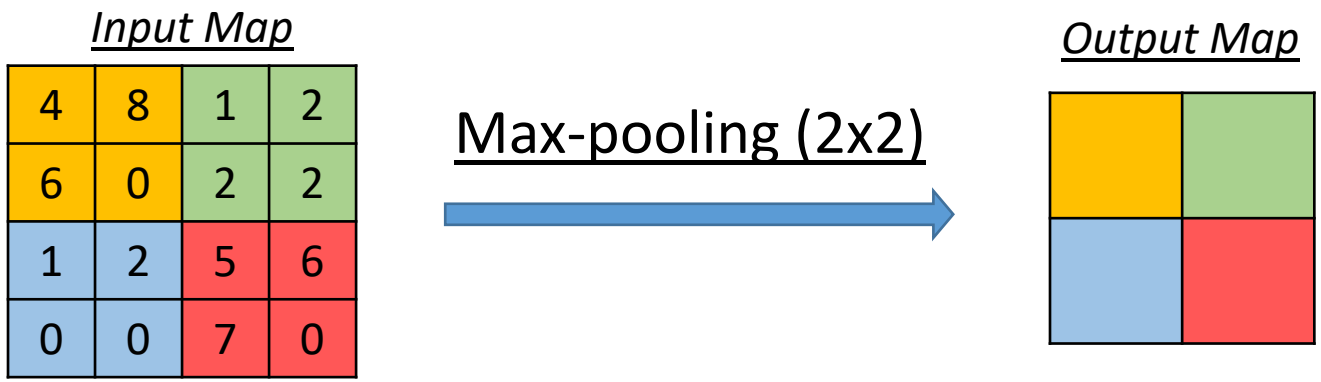


Convolutional Neural Networks

Pooling Layer

Block-wise **down-sampling** along the width/height dimensions of each input feature maps.

- **Max-Pooling**
- **Average-Pooling**

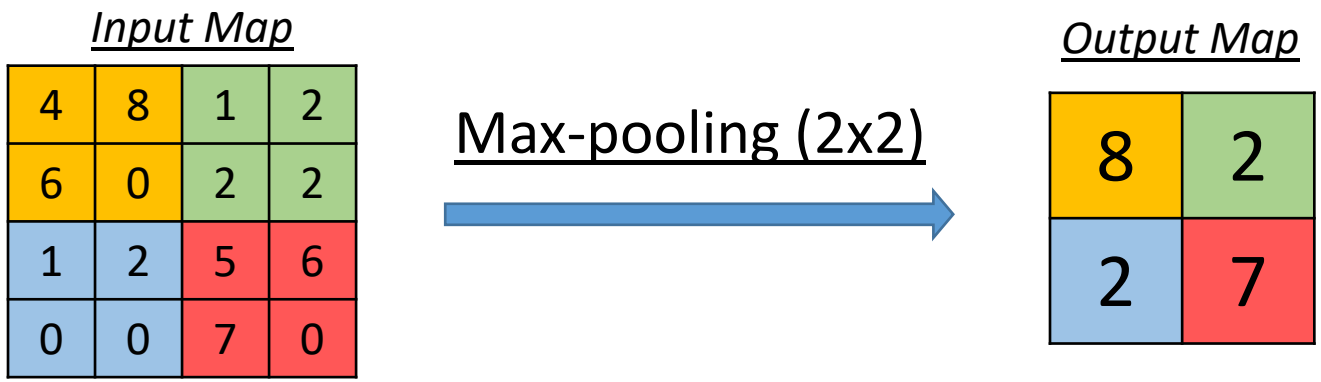


Convolutional Neural Networks

Pooling Layer

Block-wise **down-sampling** along the width/height dimensions of each input feature maps.

- **Max-Pooling**
- **Average-Pooling**

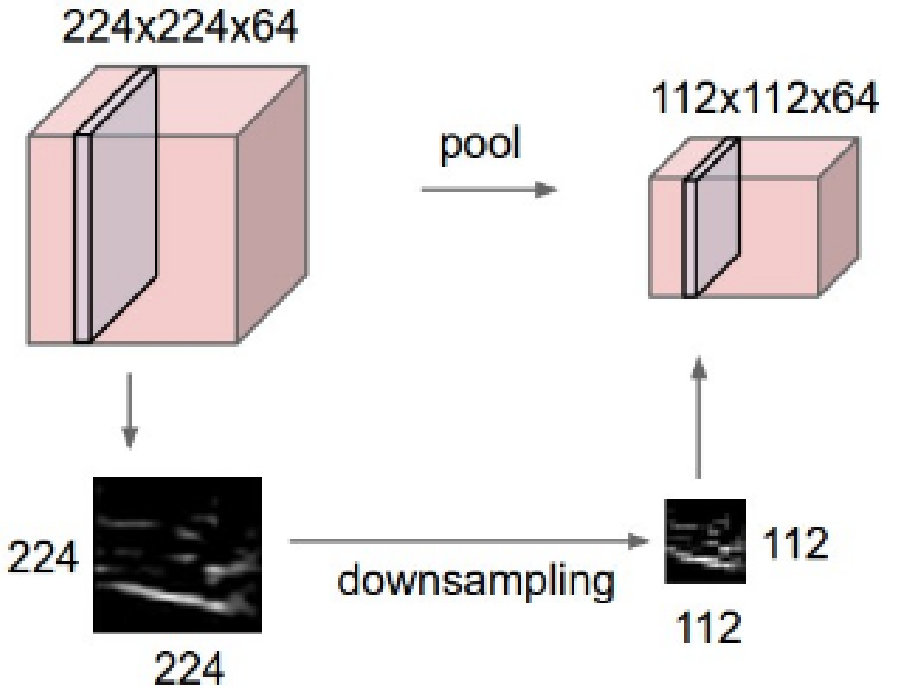


Convolutional Neural Networks

Pooling Layer

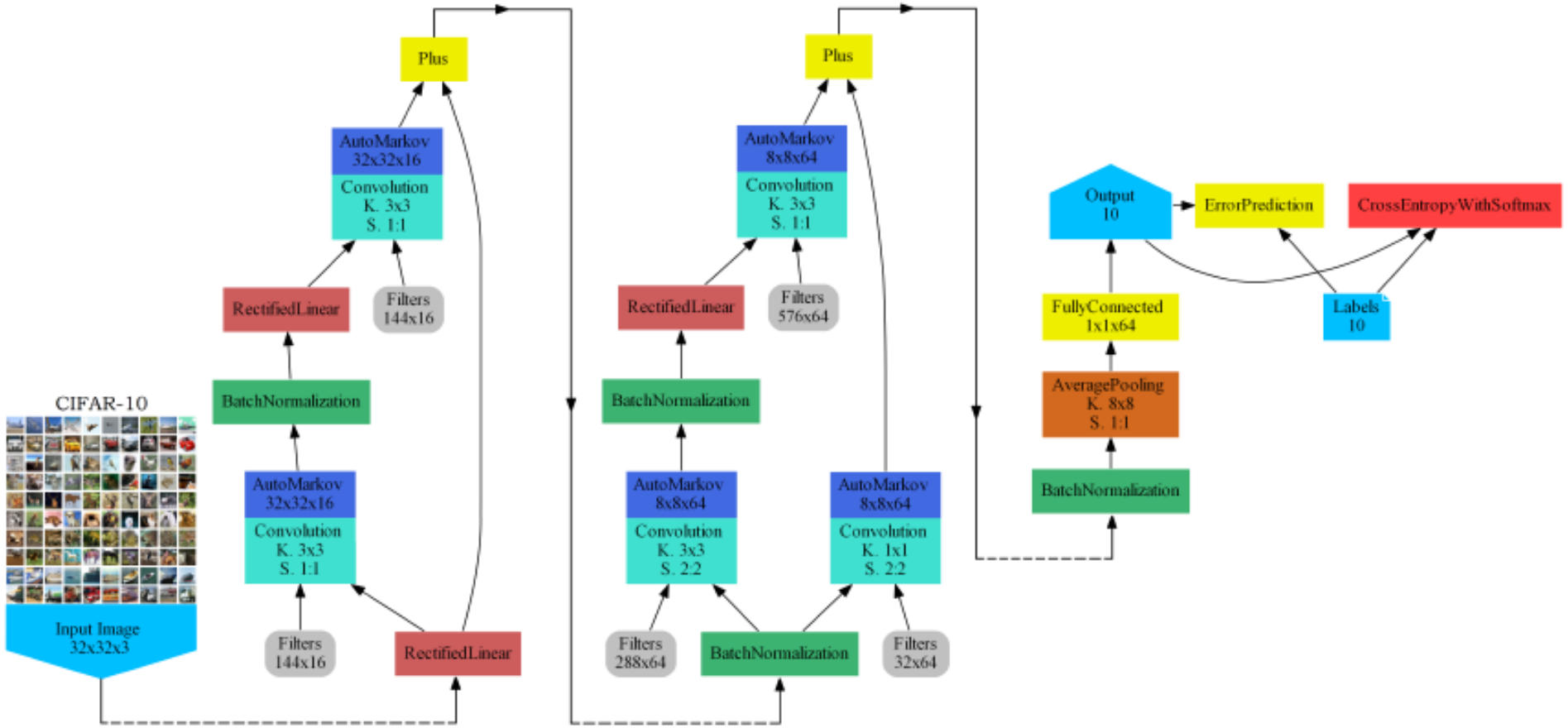
Block-wise **down-sampling** along the width/height dimensions of each input feature maps.

- **Max-Pooling**
- **Average-Pooling**



Convolutional Neural Networks

Networks Visualization

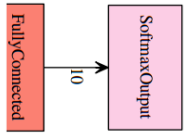


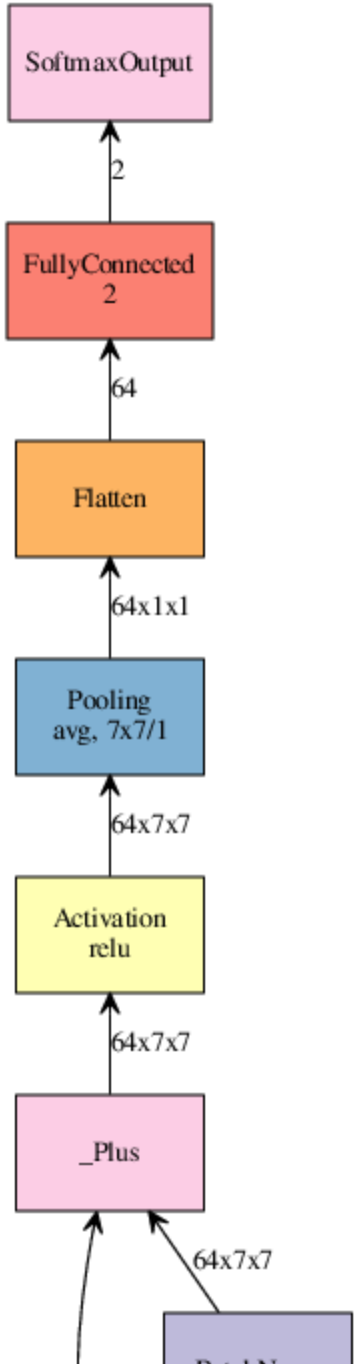
IVOM

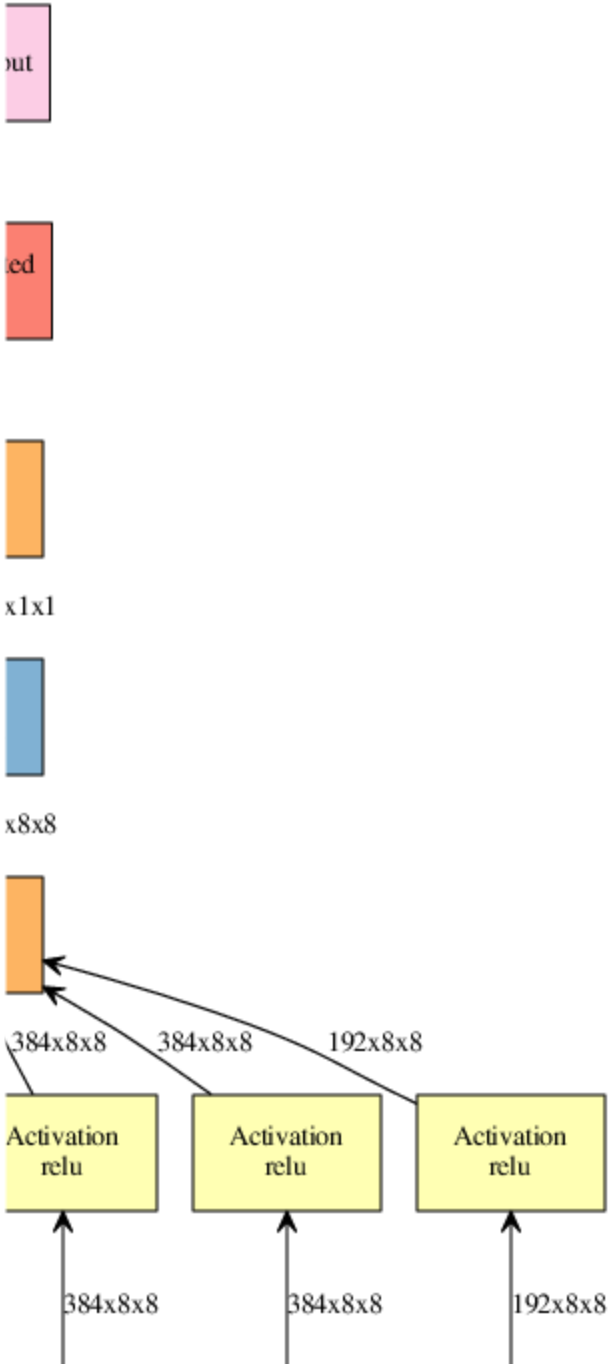
CN

Le
(19

works



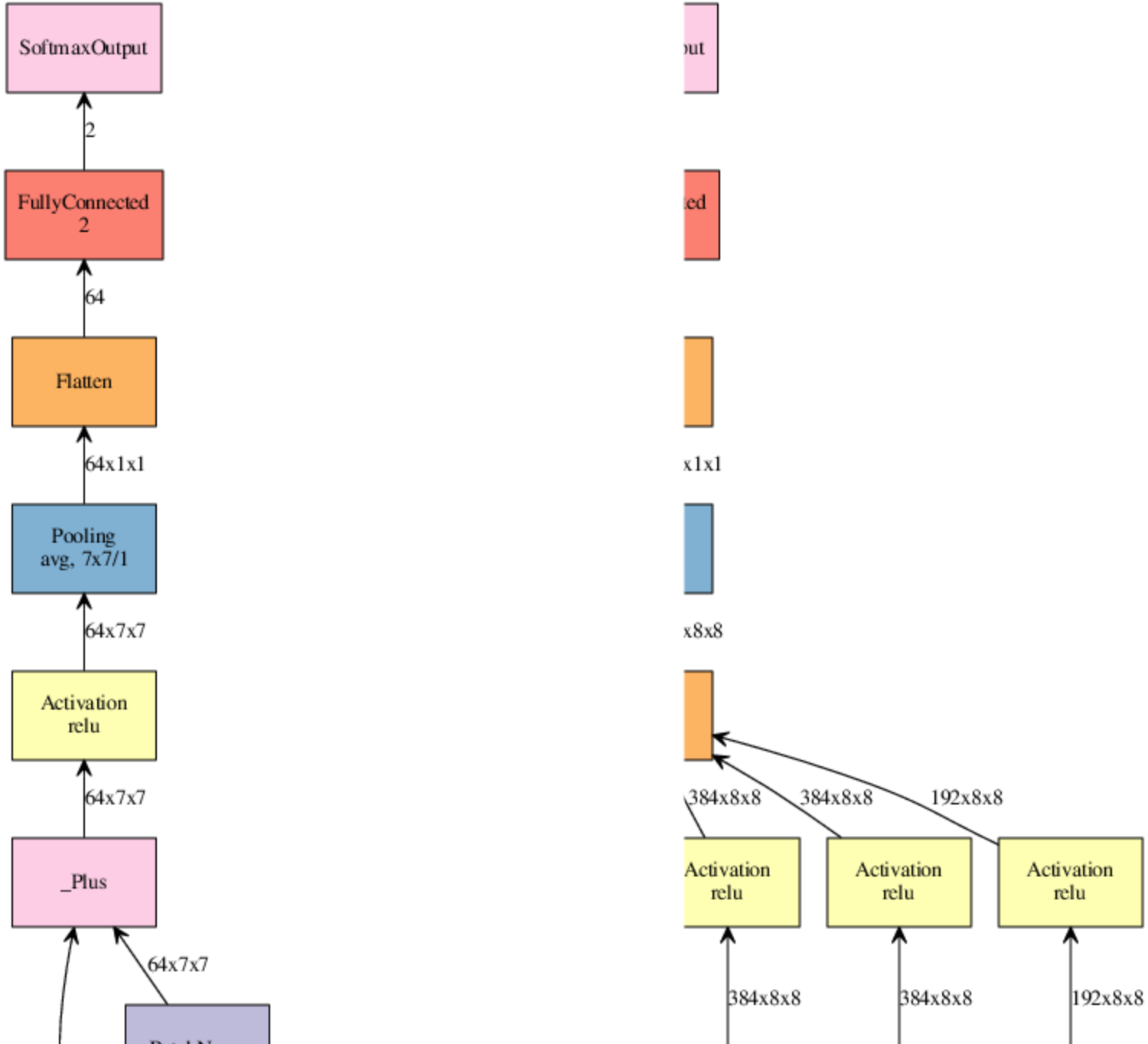




works

CN

Le
(19

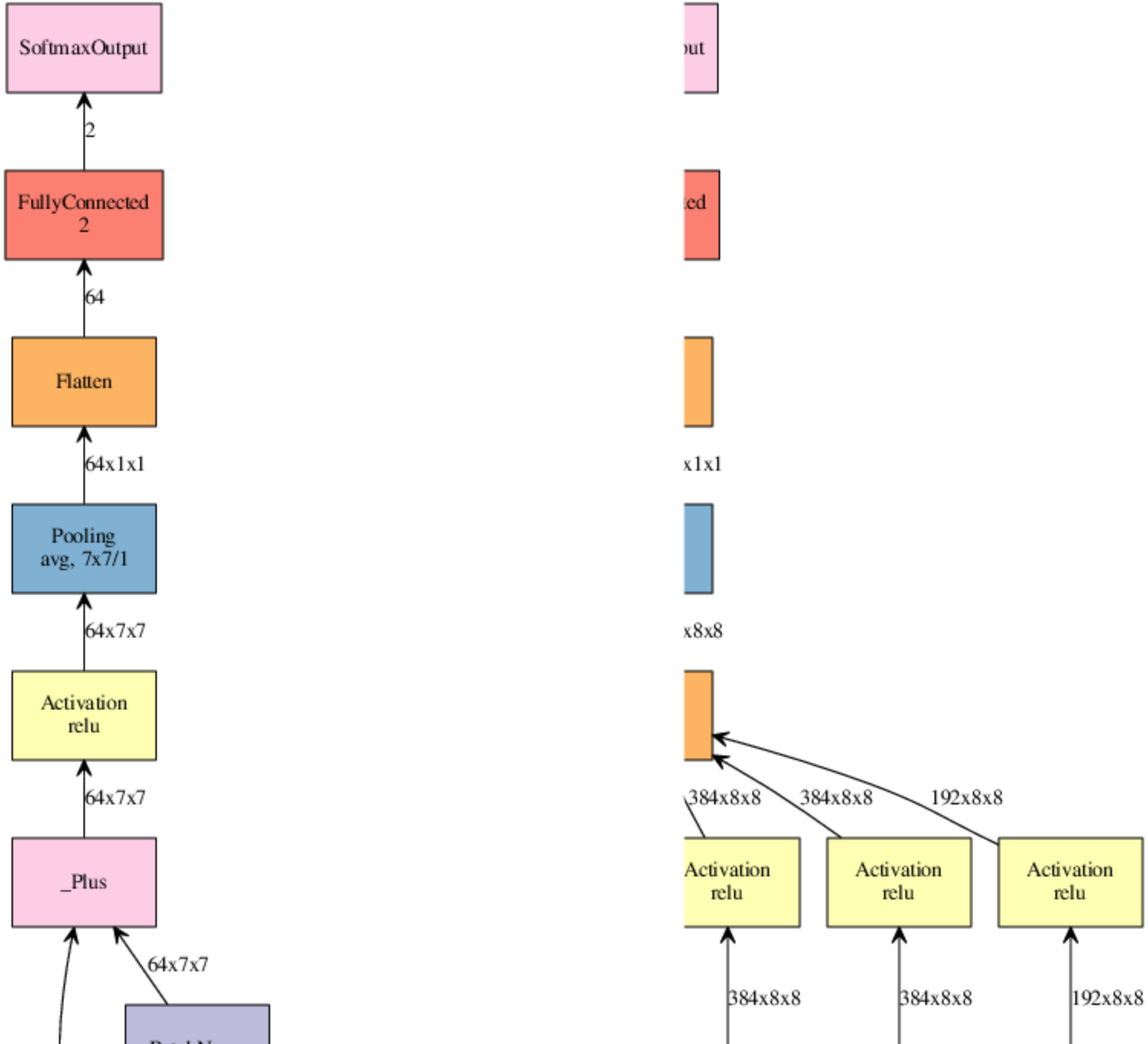


works

CN

Le
(19

Ale
(20



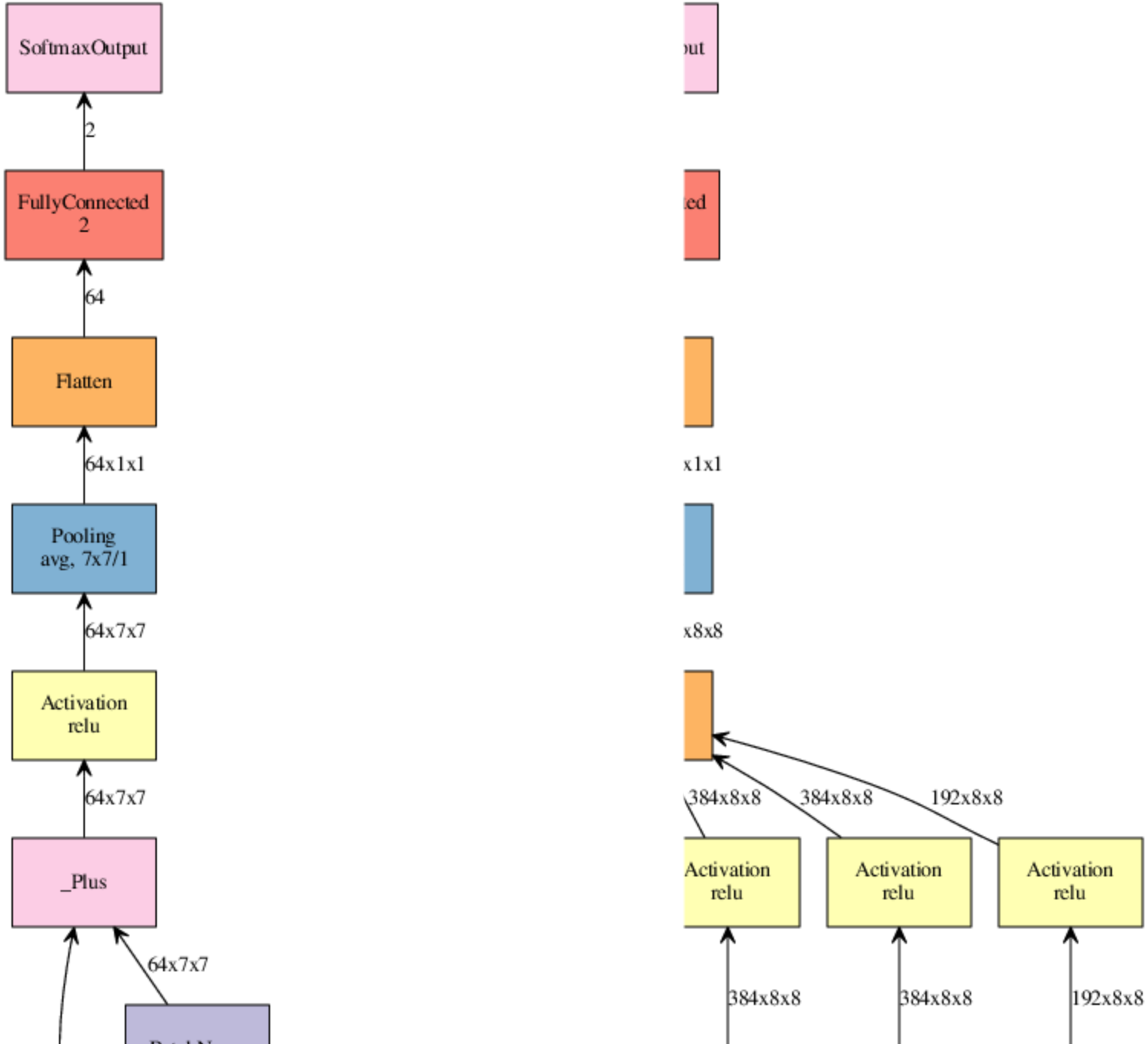
works

CN

Le
(19

Ale
(20

VG
(9/



works

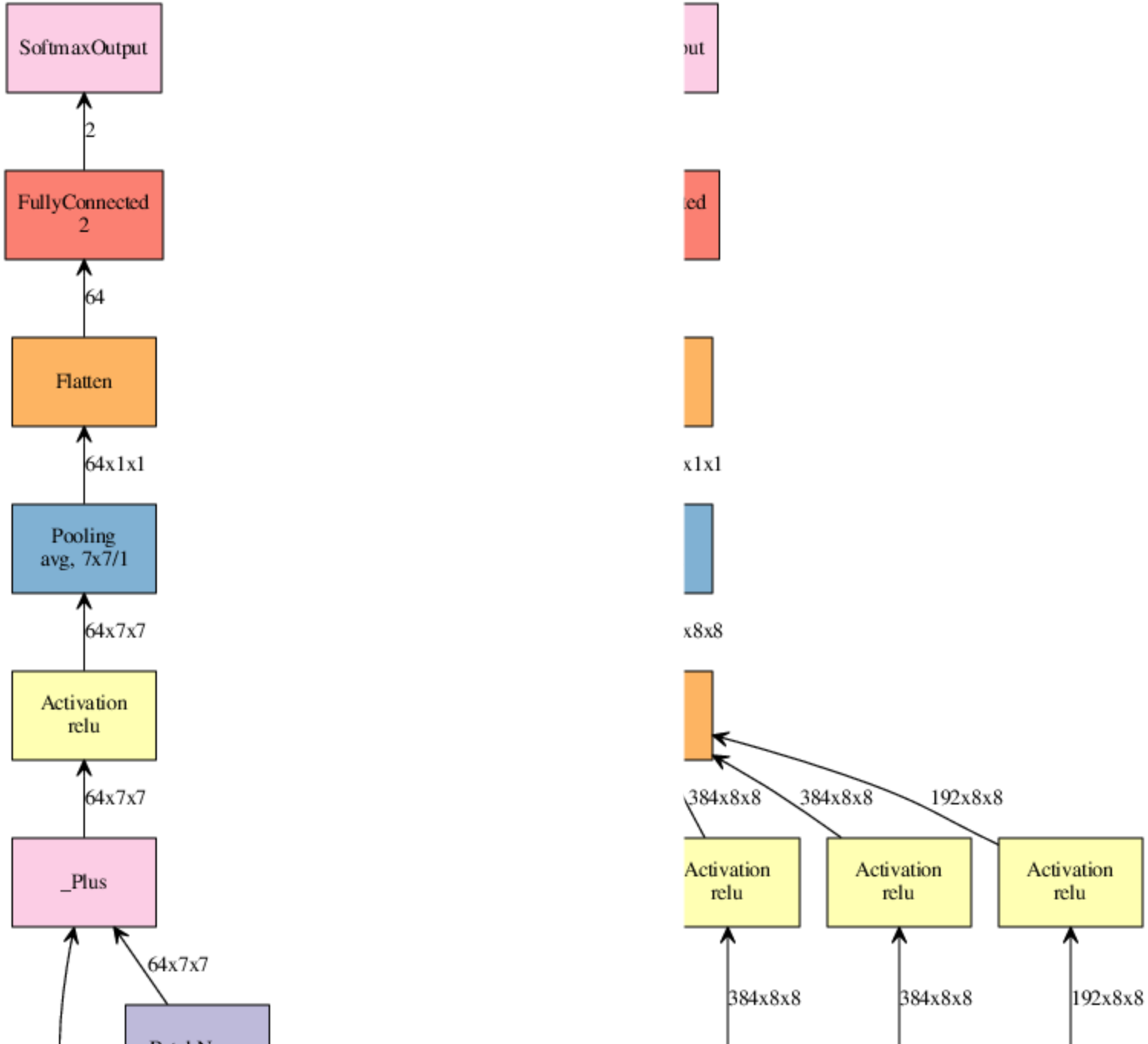
CN

Le
(19

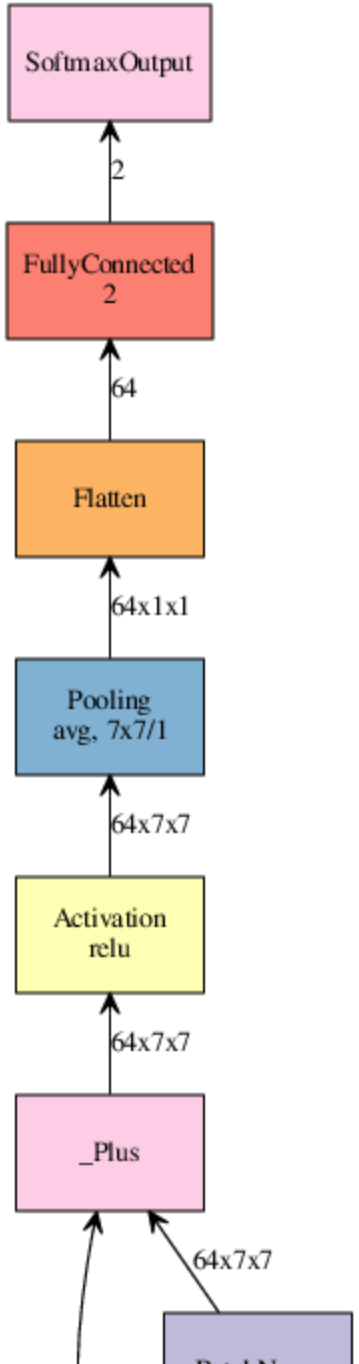
Ale
(20

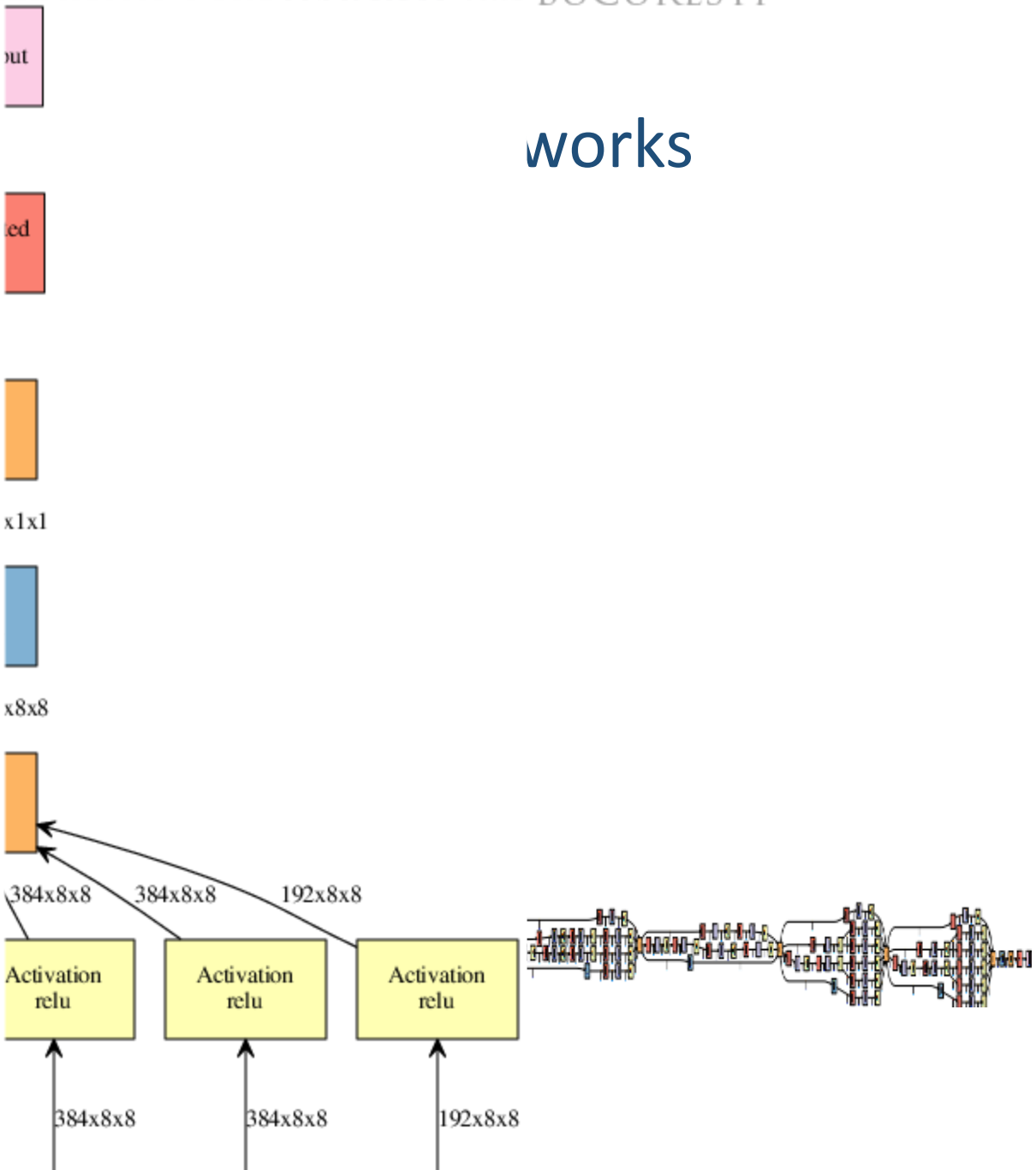
VG
(9/

Go
(9/

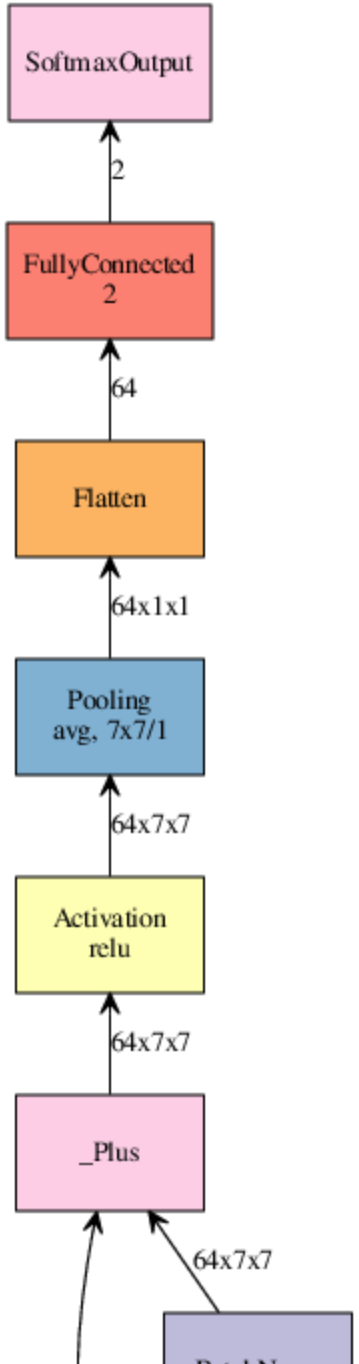


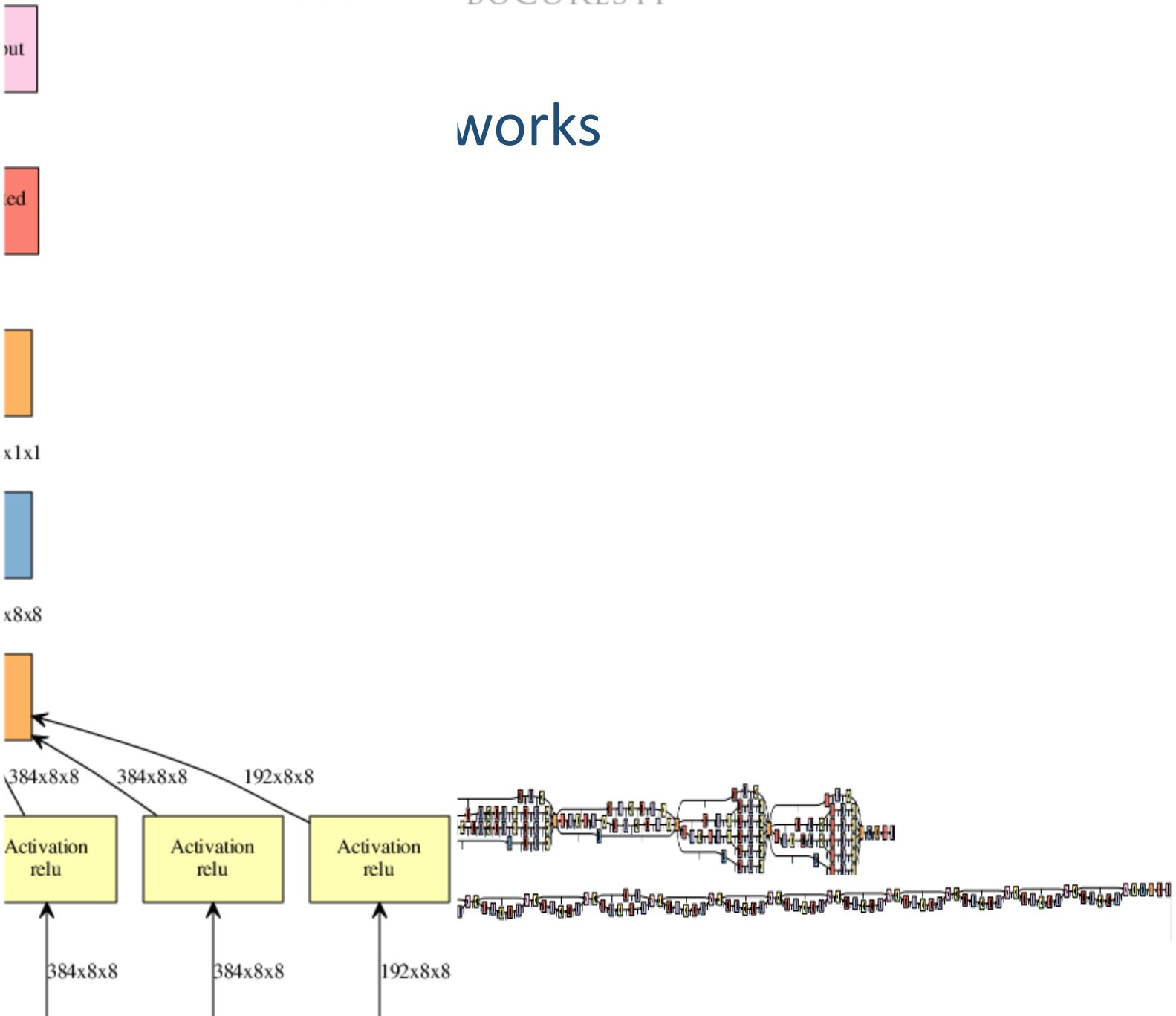
works

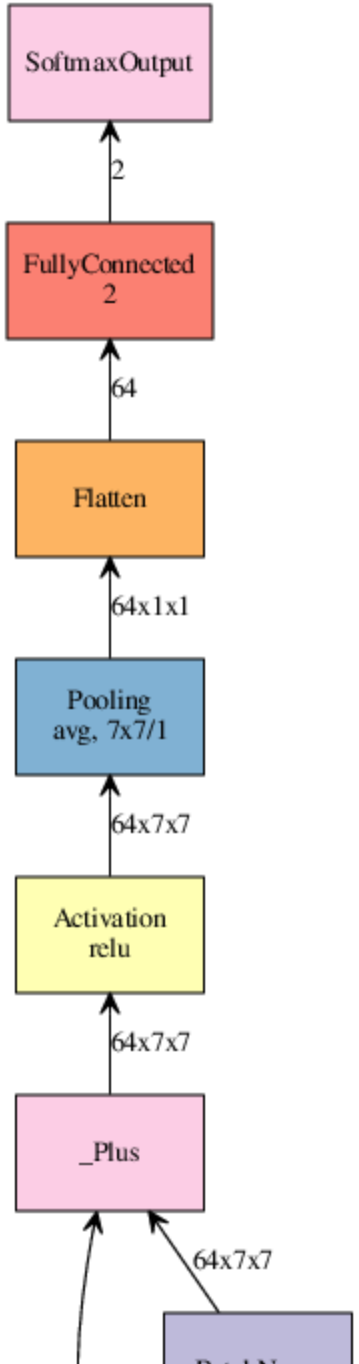

SoftmaxOutput
2
FullyConnected
2
64
Flatten
64x1x1
Pooling
avg, 7x7/1
64x7x7
Activation
relu
64x7x7
_Plus
64x7x7


384x8x8
384x8x8
192x8x8
Activation
relu
Activation
relu
Activation
relu
384x8x8
384x8x8
192x8x8

works


SoftmaxOutput
2
FullyConnected
2
64
Flatten
64x1x1
Pooling
avg, 7x7/1
64x7x7
Activation
relu
64x7x7
_Plus
64x7x7

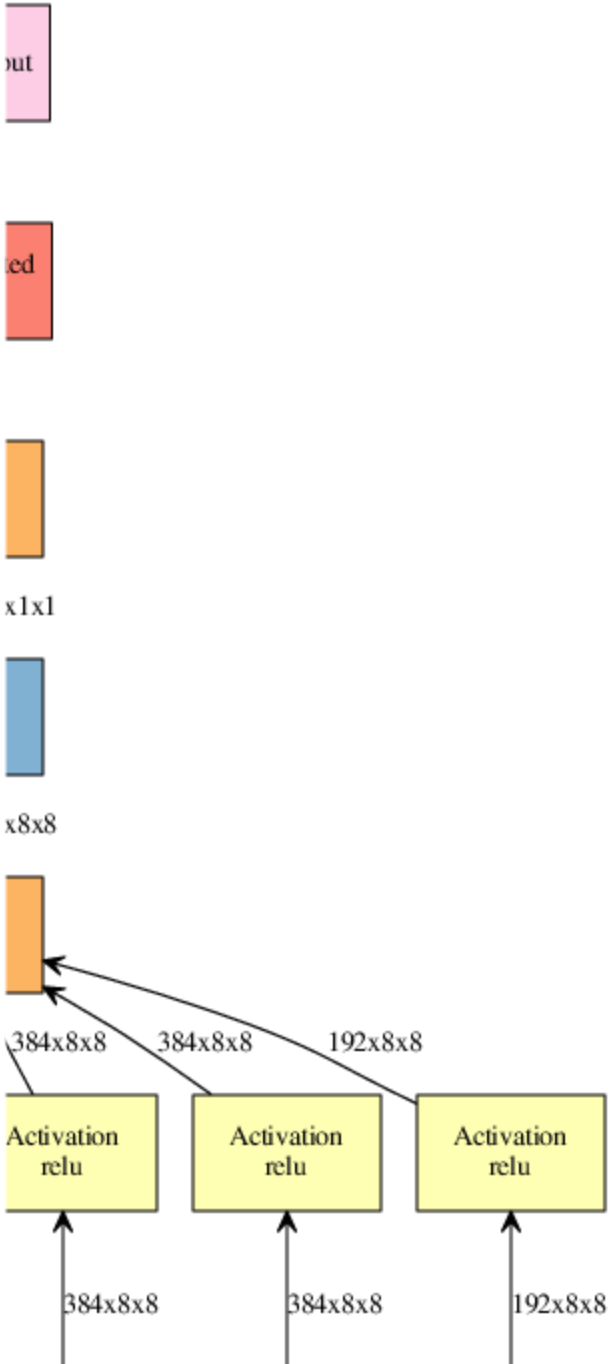

384x8x8
384x8x8
192x8x8
Activation
relu
Activation
relu
Activation
relu
384x8x8
384x8x8
192x8x8



ages

torch

ano



K

rning4j, Keras, Lasagne(Theano), Leaf

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.*Proceedings of the IEEE*, november 1998
[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in CVPR09, 2009.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in Neural Information Processing Systems, 2012.
[4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” CoRR, vol. abs/1409.1556, 2014.
[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in Computer Vision and Pattern Recognition (CVPR), 2015.
[6] A. Agarwal,et al., “An introduction to computational networks and the computational network toolkit,” Microsoft Technical Report MSR-TR-2014-112, Tech. Rep., 2014.

Q&A