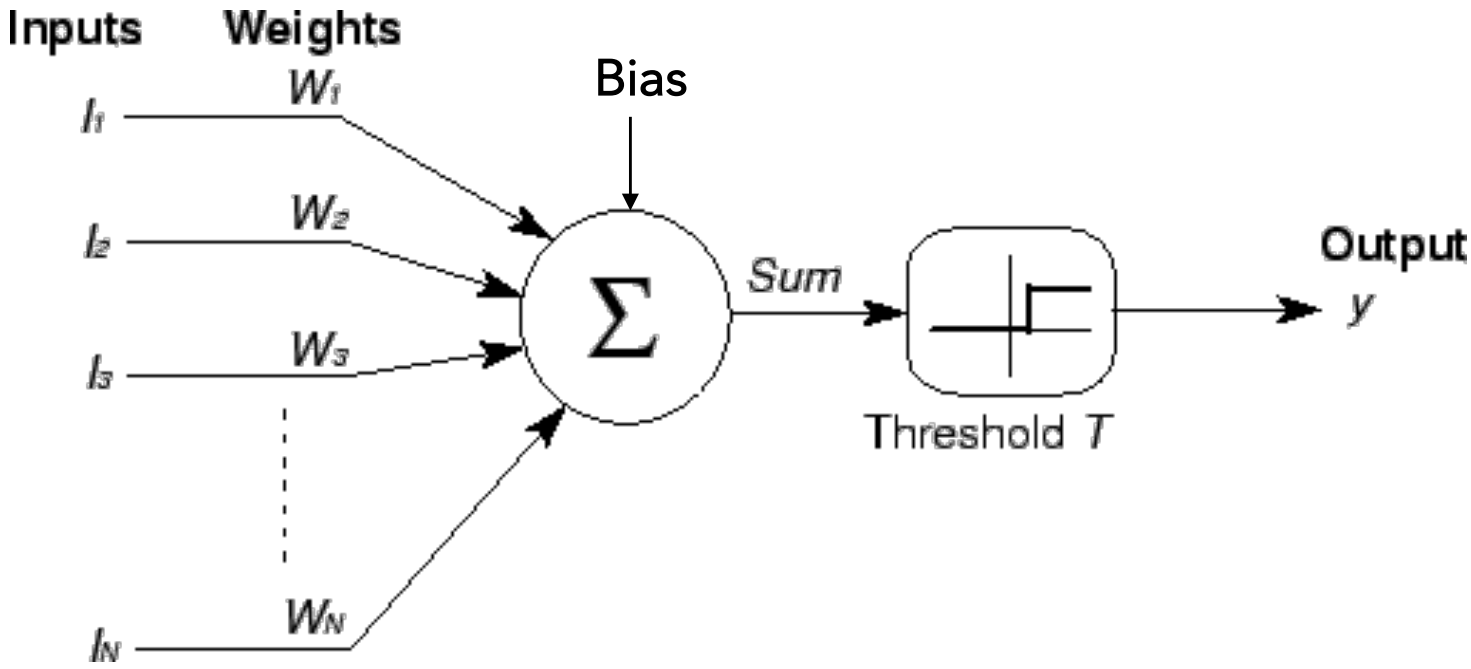
# CNN

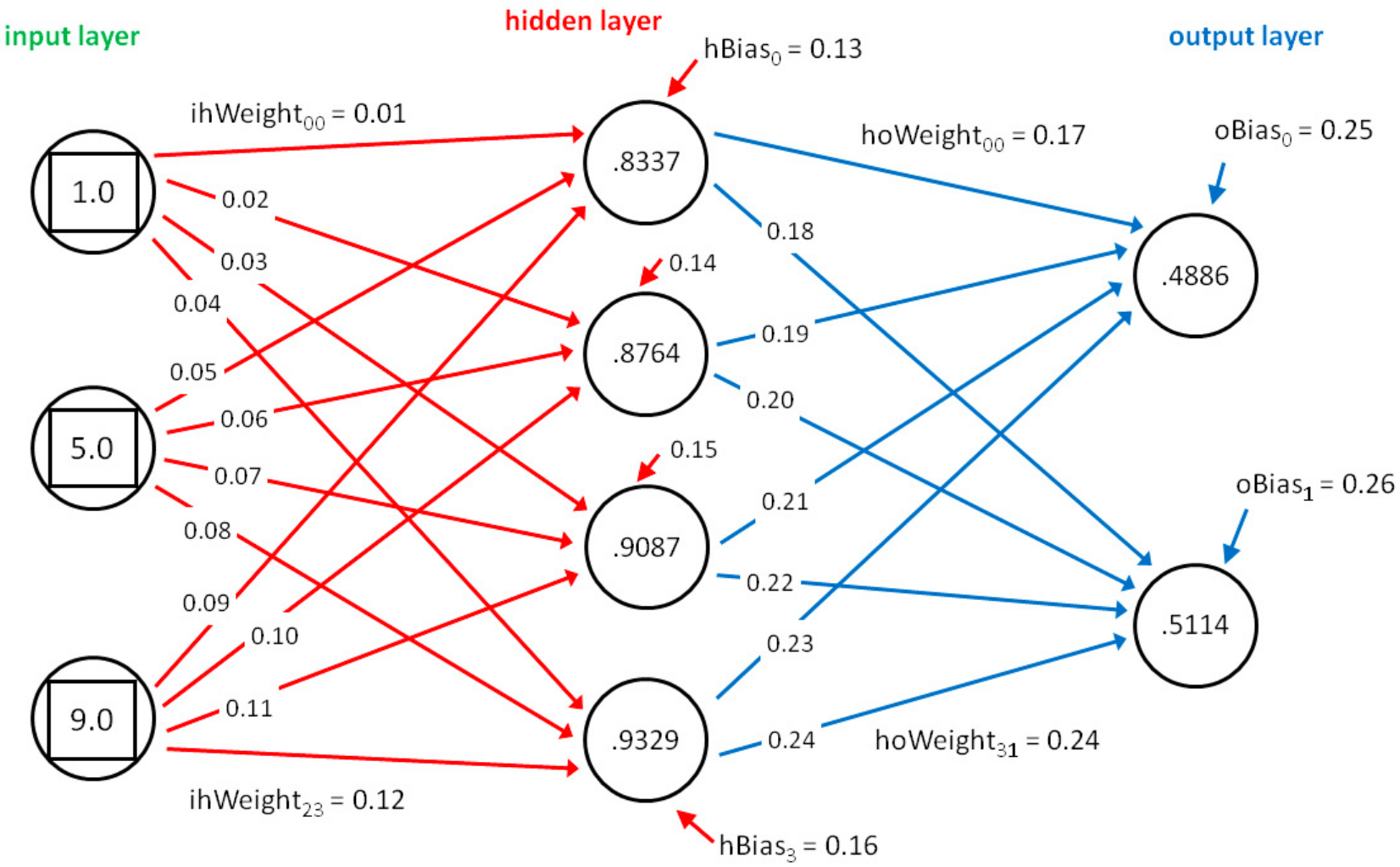
DATA 1030

TIM KRASKA

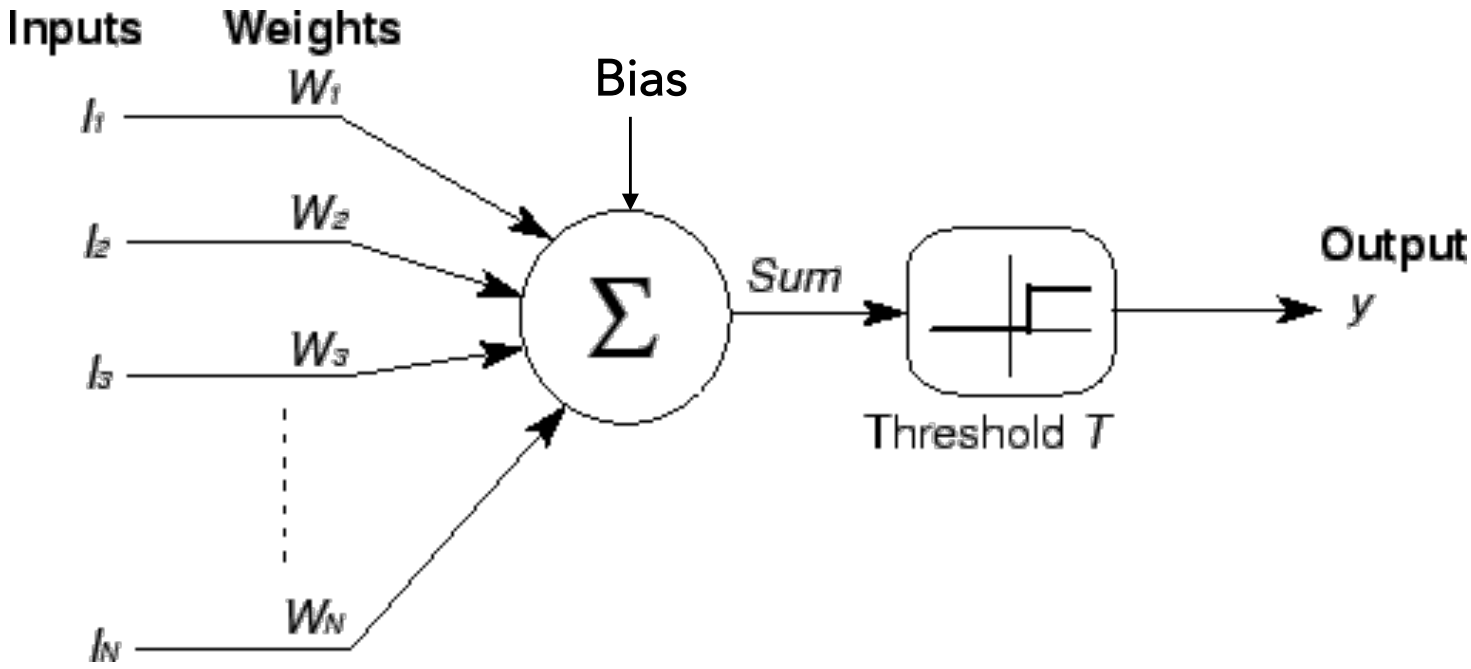
# RECAP: NN


Inputs
Weights
Bias
$I_1$
$I_2$
$I_3$
$I_N$
$W_1$
$W_2$
$W_3$
$W_N$
Σ
Sum
Threshold $T$
Output
$y$

# RECAP: NN


input layer
hidden layer
output layer
hBias0 = 0.13
ihWeight00 = 0.01
hoWeight00 = 0.17
oBias0 = 0.25
1.0
0.02
0.03
0.04
.8337
0.18
.4886
0.14
.8764
0.19
0.20
0.05
0.06
5.0
0.07
0.15
.9087
0.21
0.22
oBias1 = 0.26
0.08
0.09
0.10
0.11
.5114
0.23
9.0
.9329
0.24
hoWeight31 = 0.24
ihWeight23 = 0.12
hBias3 = 0.16

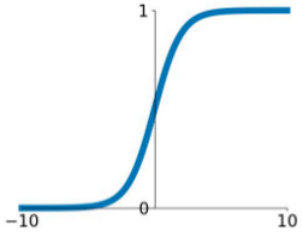
# RECAP: NN


Inputs
Weights
Bias
$I_1$
$I_2$
$I_3$
$I_N$
$W_1$
$W_2$
$W_3$
$W_N$
Σ
Sum
Threshold $T$
Output
$y$

# ACTIVATION FUNCTIONS

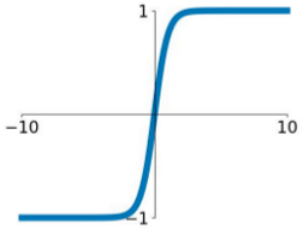
**Sigmoid**

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



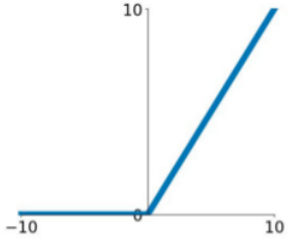

**tanh**

$$\tanh(x)$$



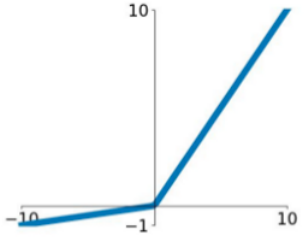

**ReLU**

$$\max(0, x)$$




**Leaky ReLU**

$$\max(0.1x, x)$$

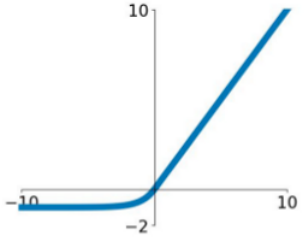



**Maxout**

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

**ELU**

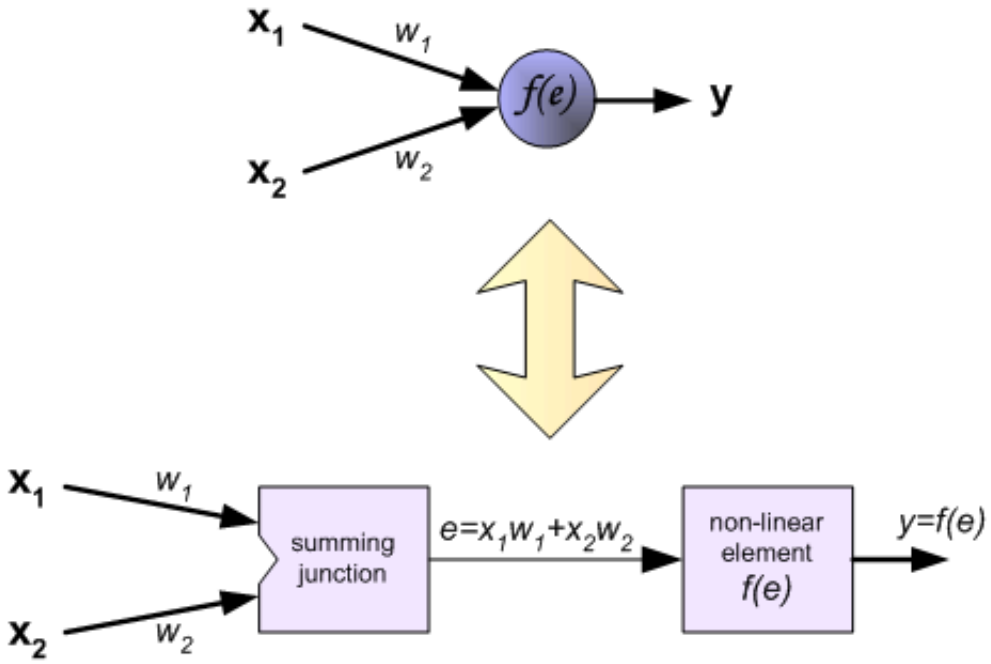
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$




Question: Could we use a linear activation functin f(x) = c * x?
What does that mean?
What does it mean for layering?

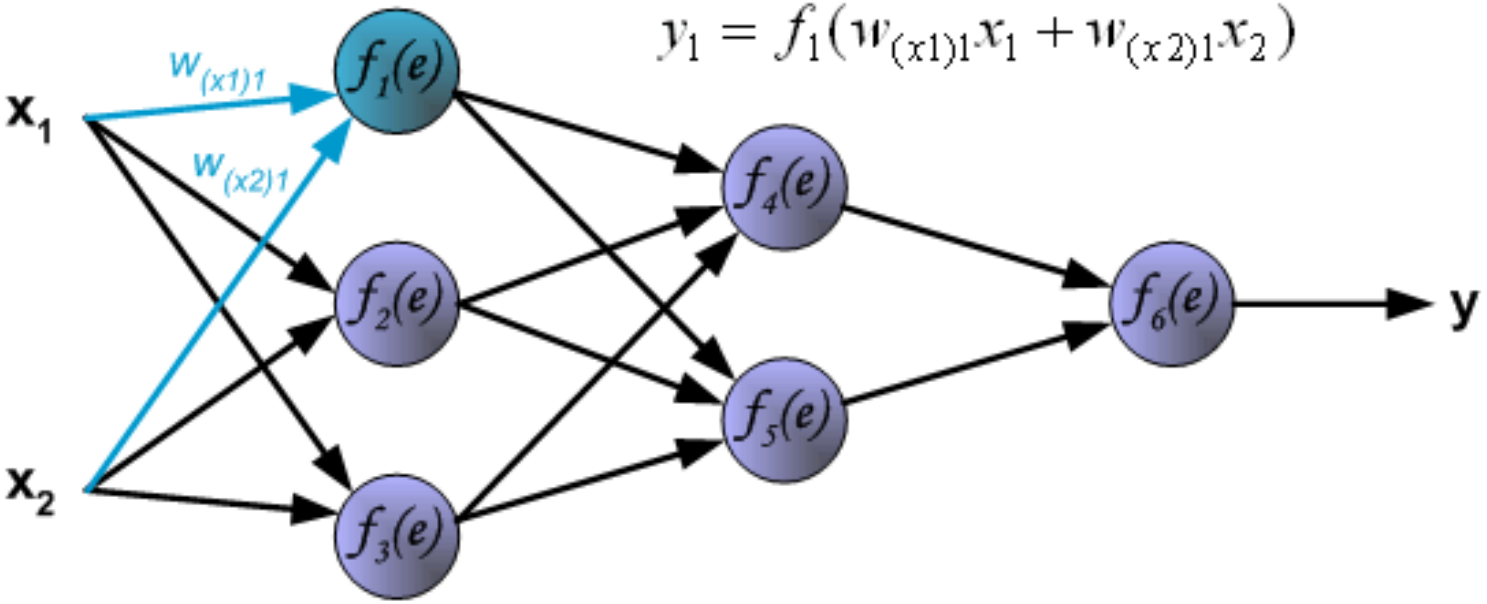
# BACKPROPAGATION

Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realise nonlinear function, called neuron transfer (activation) function. Signal *e* is adder output signal, and *y = f(e)* is output signal of nonlinear element. Signal *y* is also output signal of neuron.

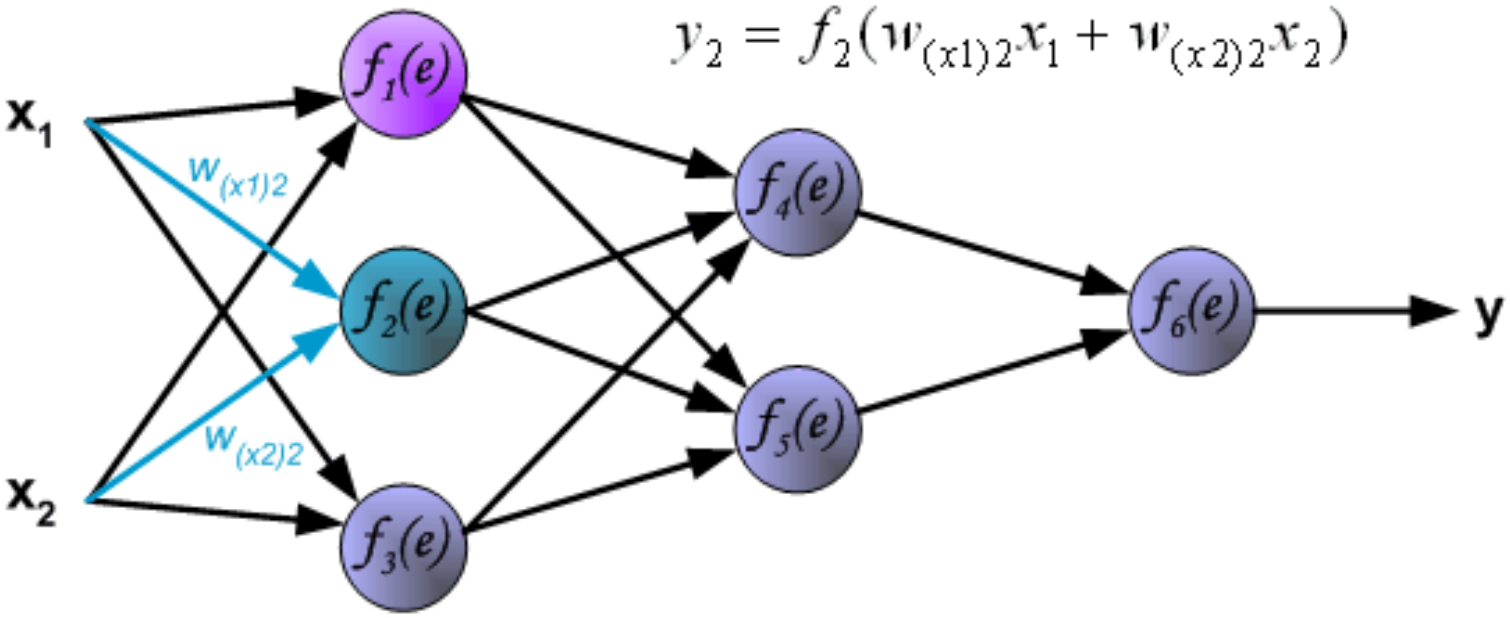


# FORWARD PASS

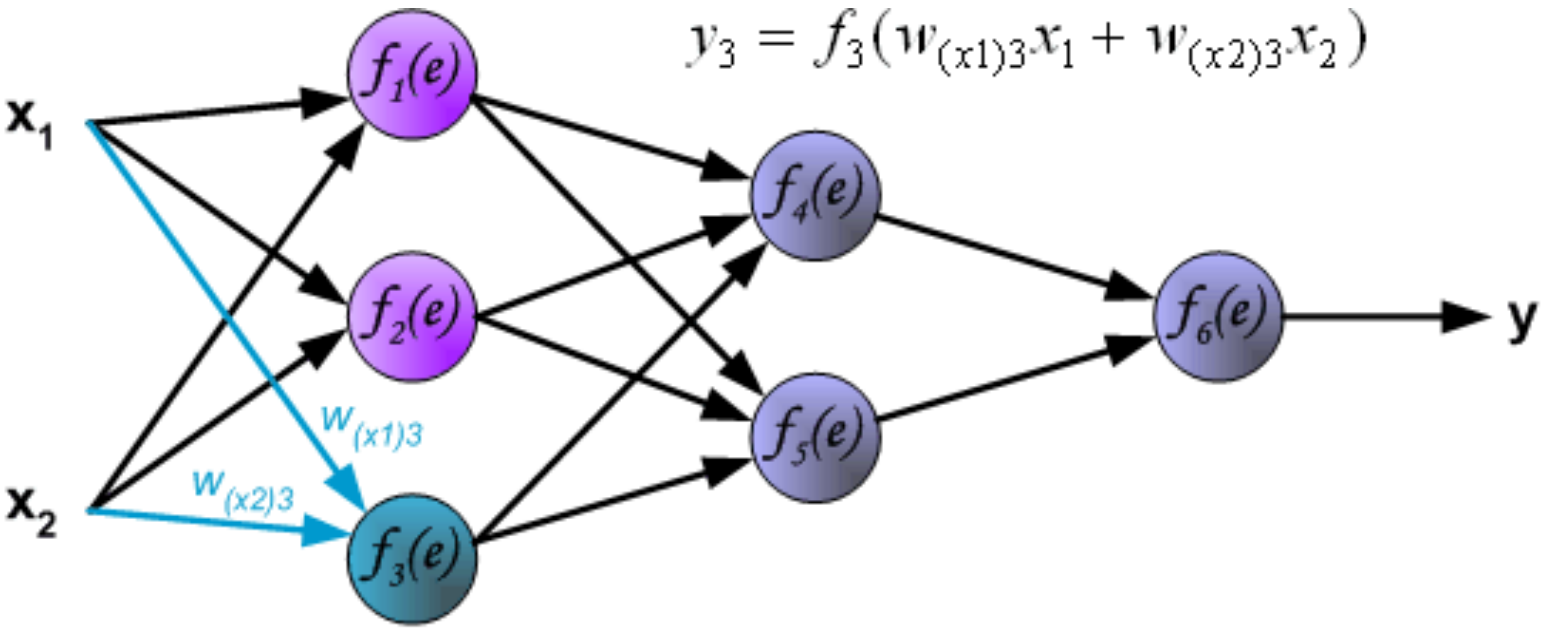
Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input $x_m$ and neuron $n$ in input layer. Symbols $y_n$ represents output signal of neuron $n$.



# FORWARD PASS

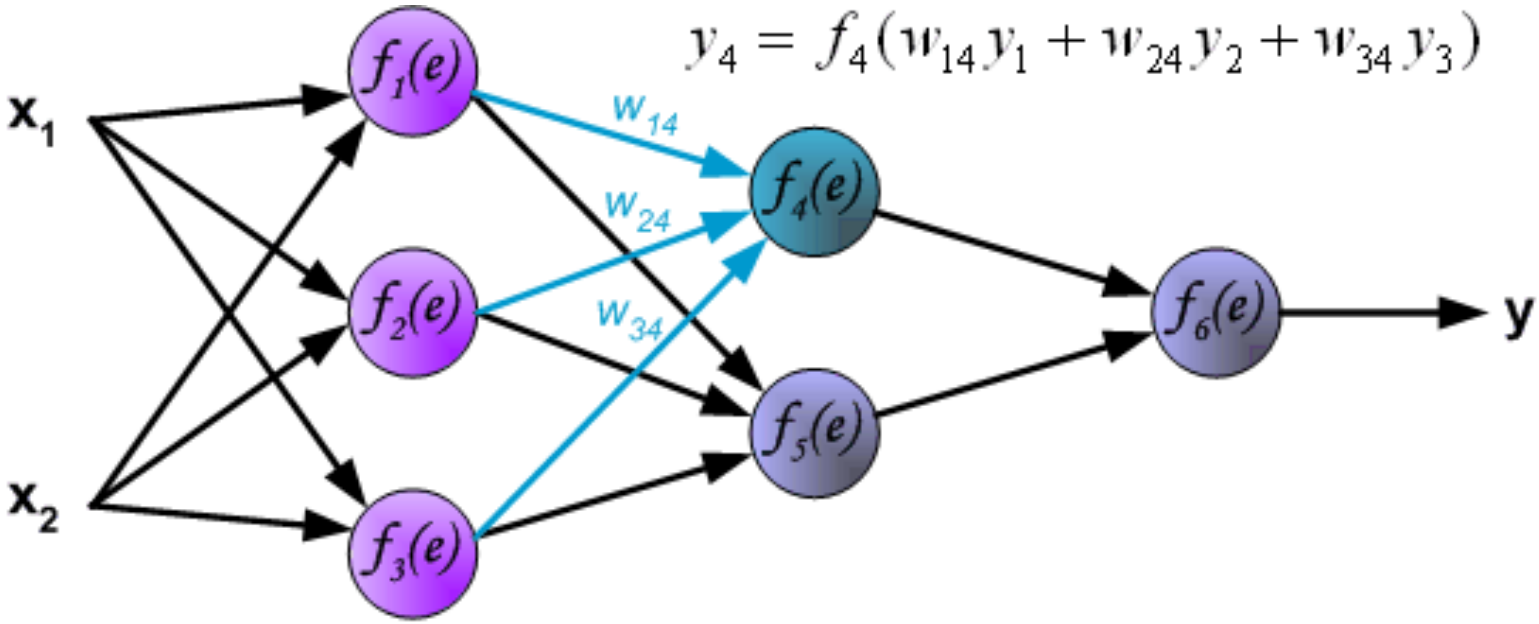


# FORWARD PASS

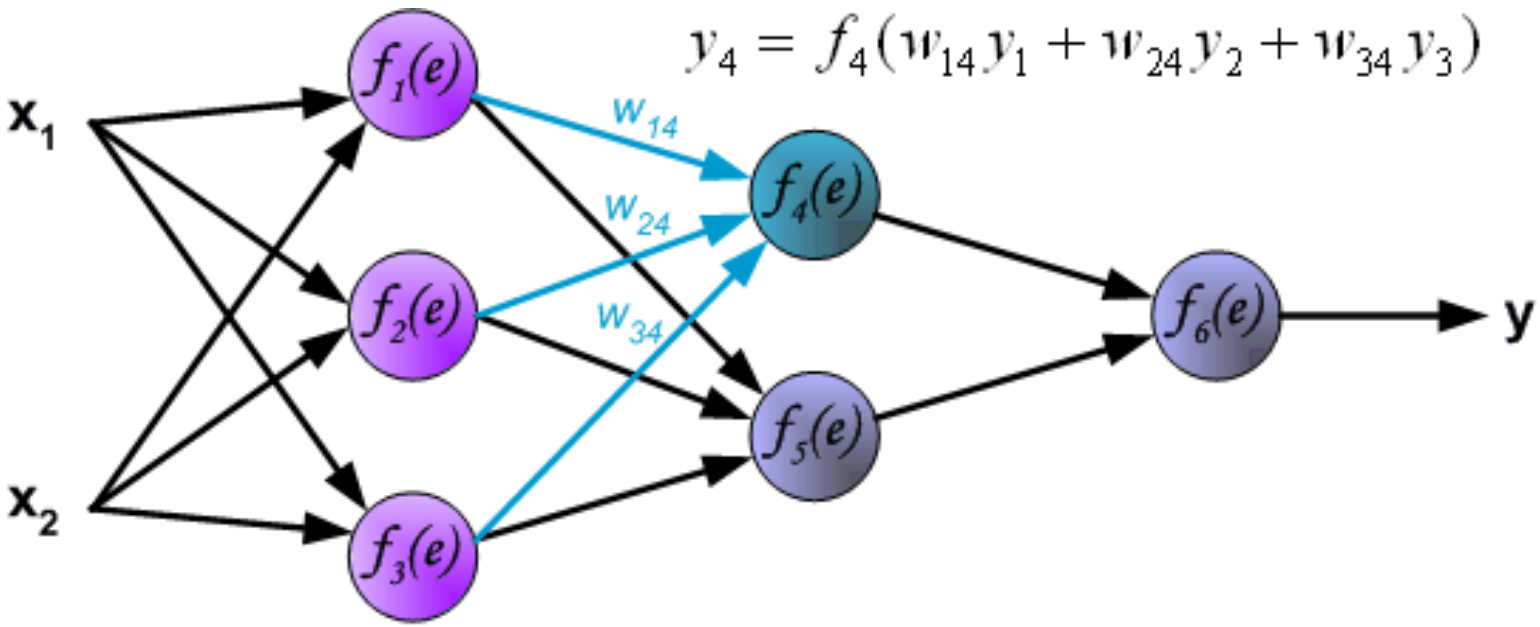


# FORWARD PASS

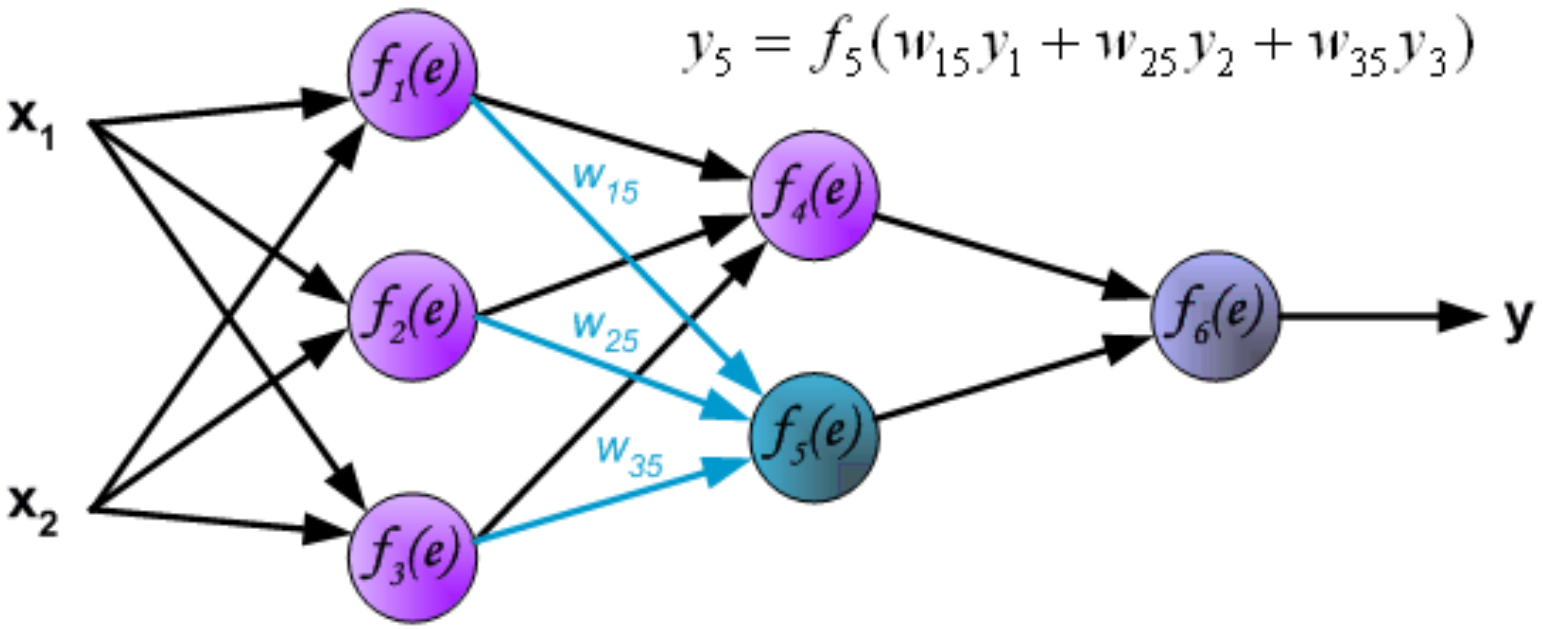
Propagation of signals through the hidden layer. Symbols $w_{mn}$ represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.



# FORWARD PASS

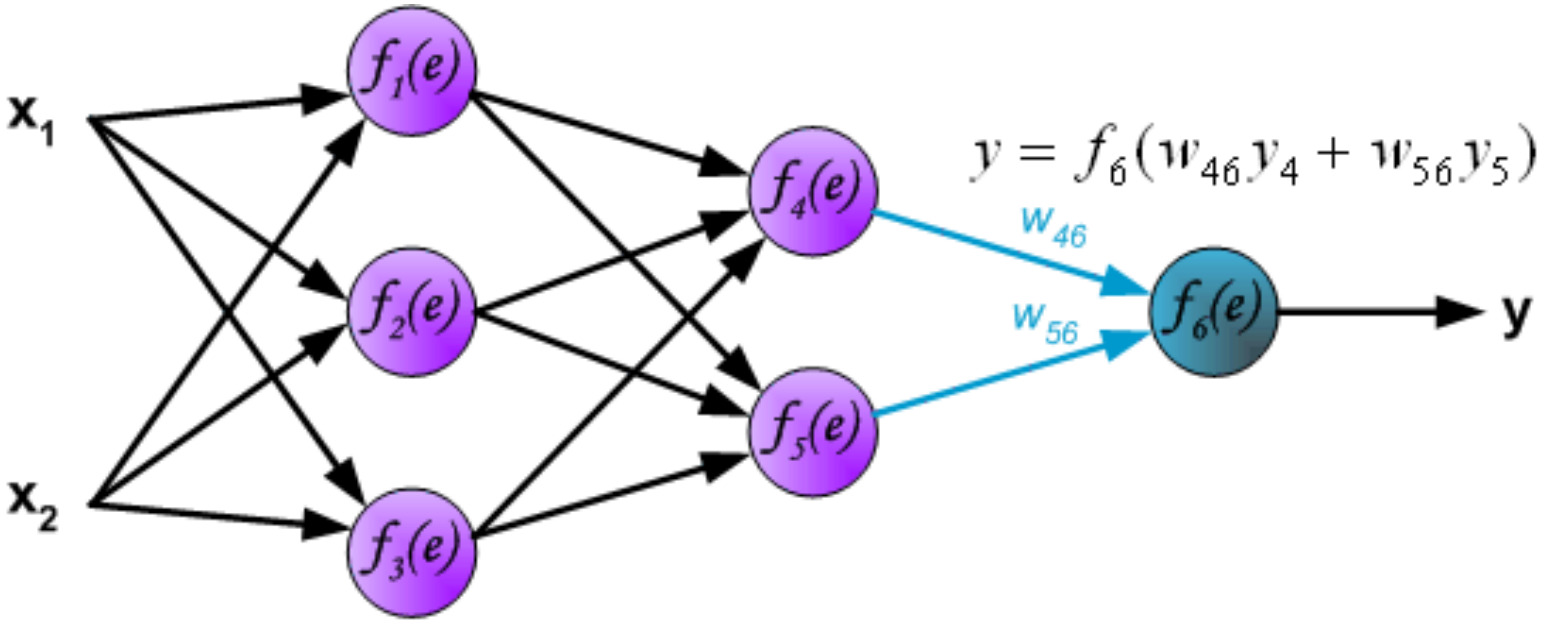


# FORWARD PASS



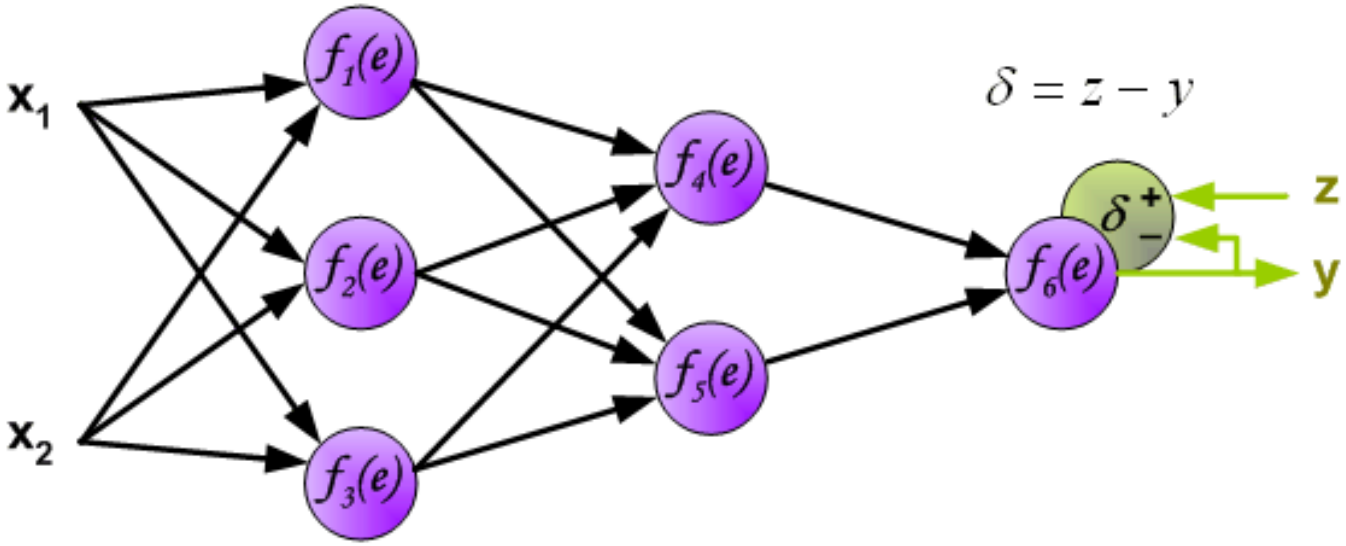
# FORWARD PASS

Propagation of signals through the output layer.



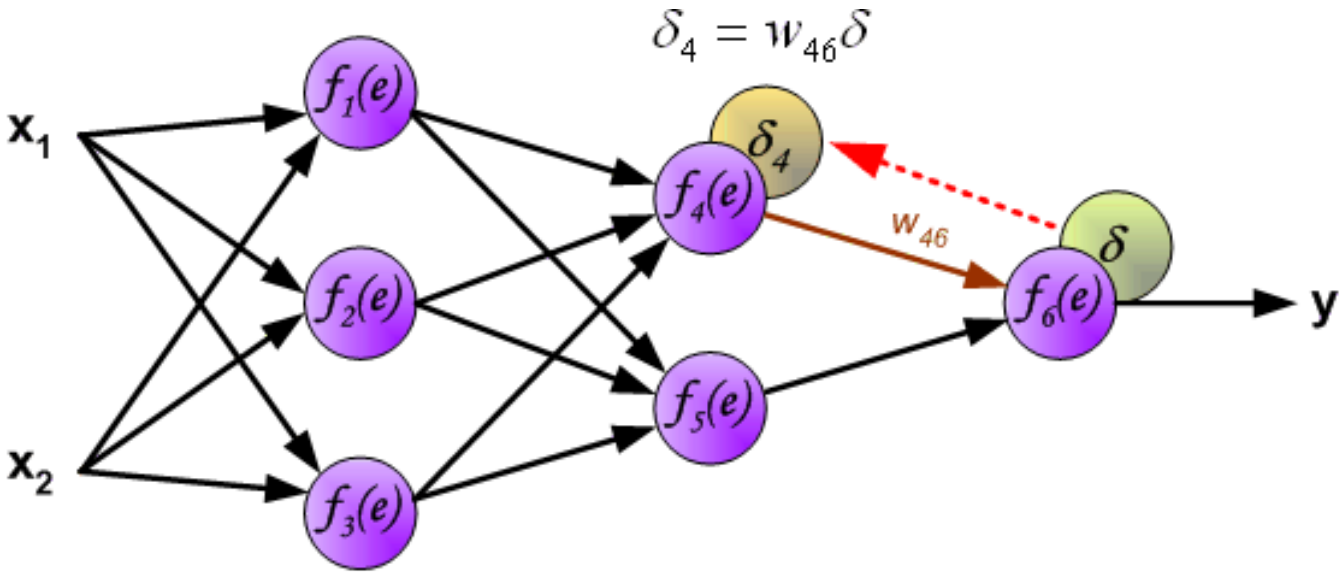
# BACKPROPOGATİON

In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal *d* of output layer neuron



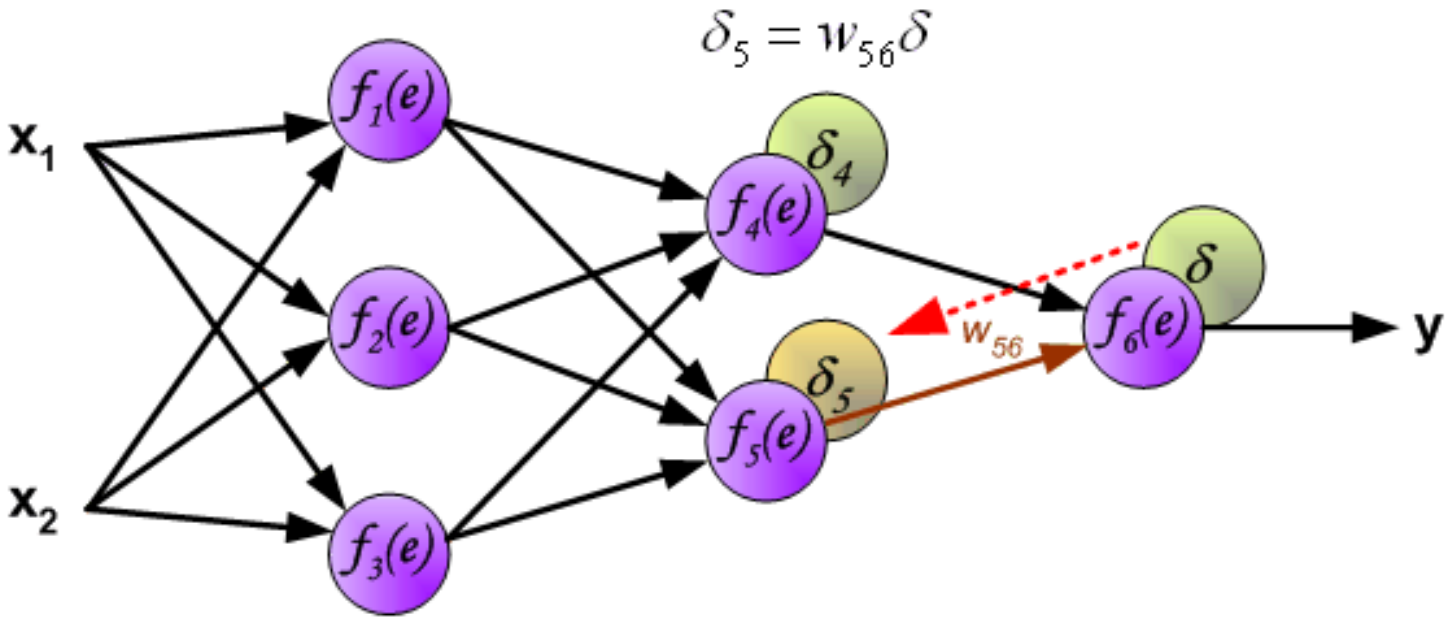
# BACKPROPAGATİON

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



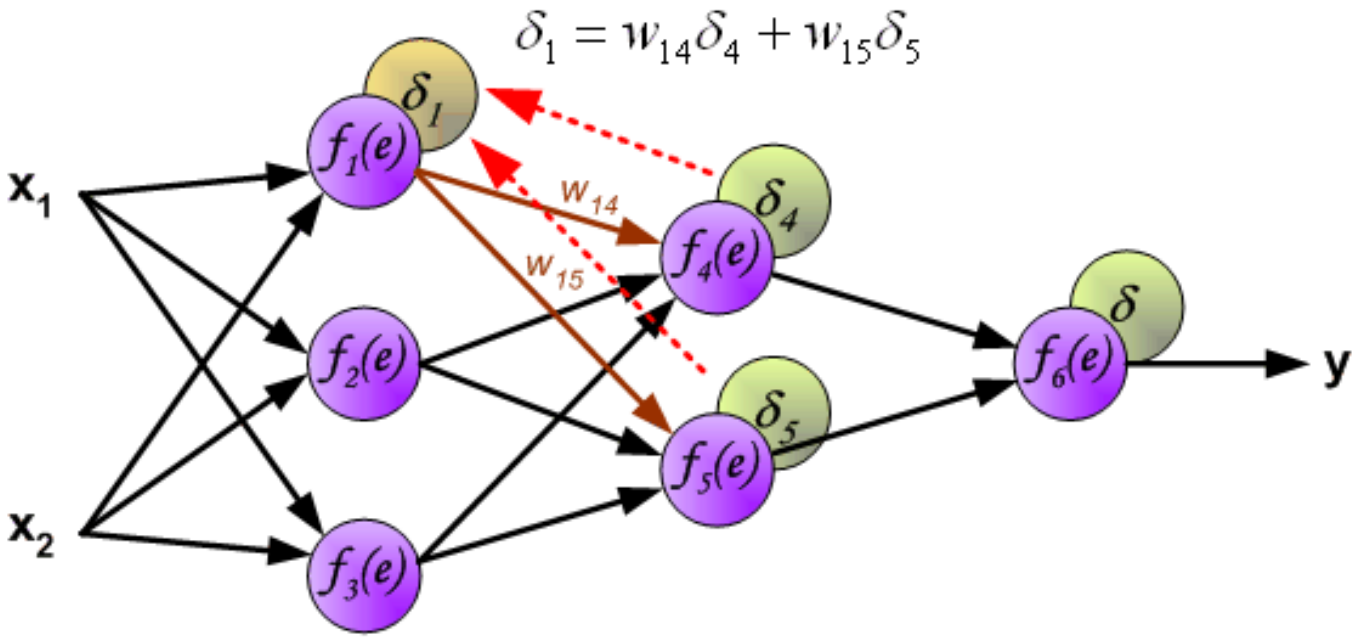
# BACKPROPAGATİON

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



# BACKPROPAGATİON

The weights' coefficients $w_{mn}$ used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:

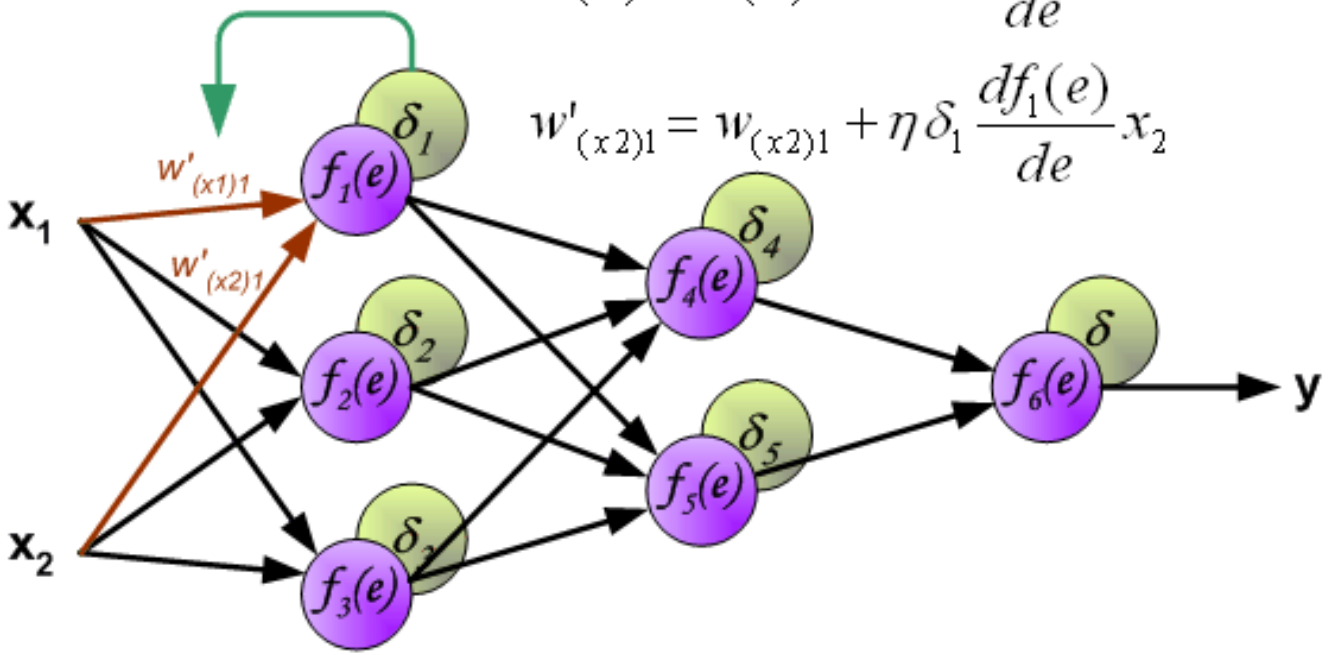


# BACKPROPAGATİON

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

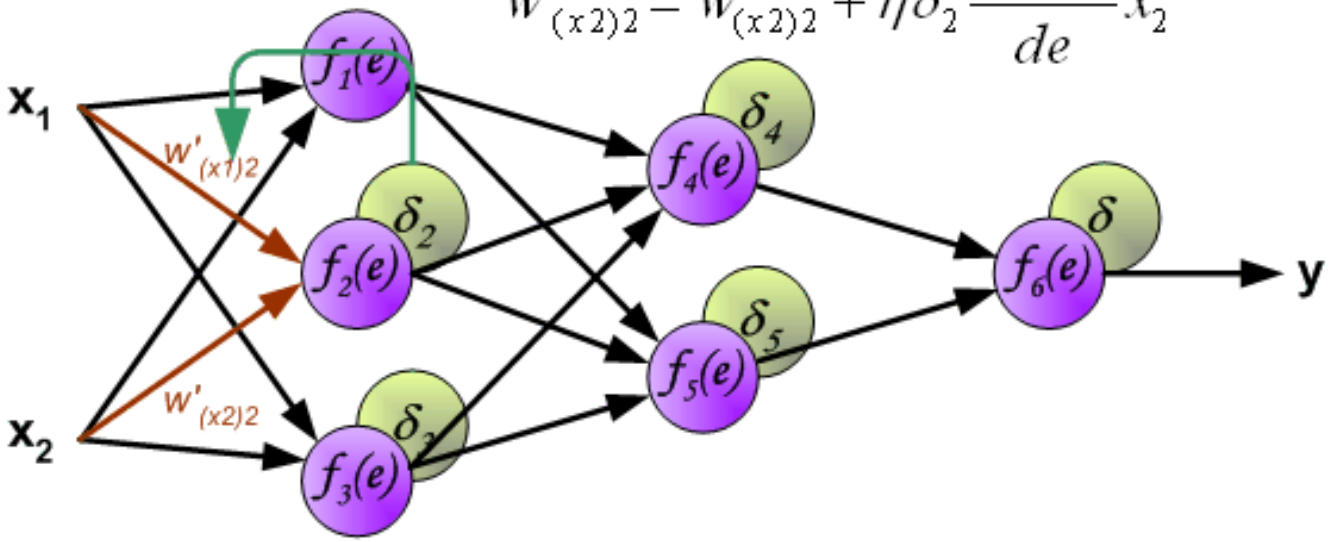


# BACKPROPAGATİON

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$

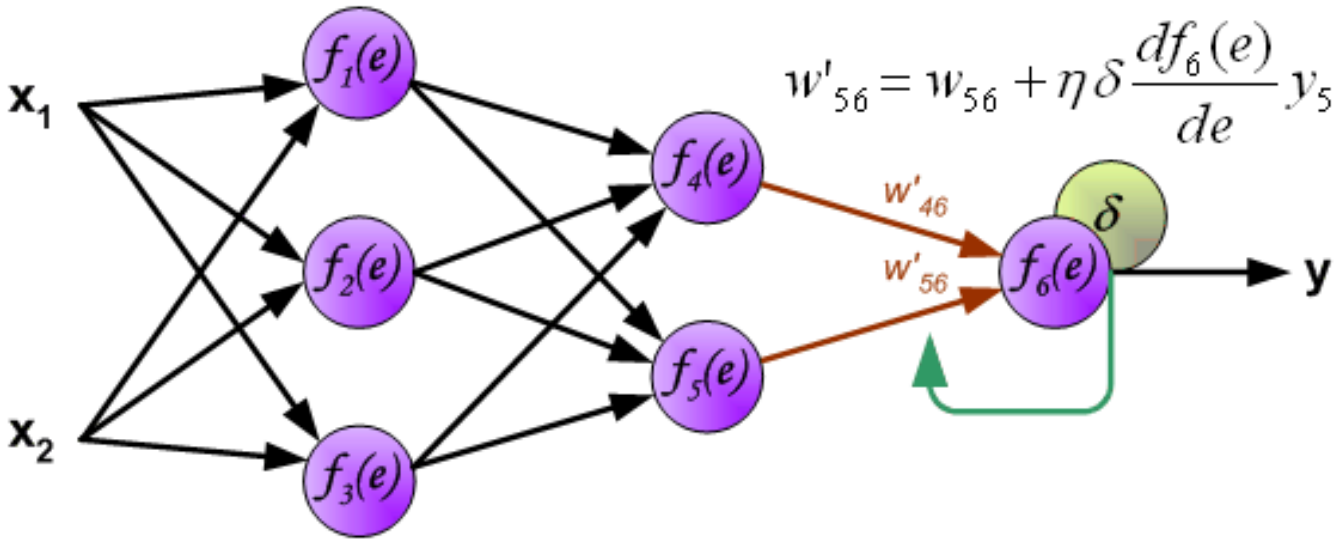


# BACKPROPAGATİON

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



# TUNING NN

## playground.tensorflow.org

Task 1: Select first data set

- No regularization 
- ReLU Activation
- Use 1 hidden layer with 2 neurons
- Only X1 and X2 as features

Question: Why can the network not separate the two?

# TASK 2

Data set 

- Only X1 and X2 as features
- No regularization
- ReLU Activation
- Use 1 hidden layer with 3 neurons

Question:

Can the network  now separate the classes? Why?

# TASK 3

Data set 

- Only X1 and X2 as features
- No regularization
- Try Sigmoid and Tanh Activation
- Use 1 hidden layer with 3 neurons

Questions:

1. Can you make the network to separate the classes?
2. Why does it take longer/faster to converge?
3. Do you notice a difference in the decision boundary?

# TASK 4

Data set 

- No regularization
- Linear Activation
- Use 1 hidden layer with 3 neurons

Questions:

1. Can you make the network to separate the classes?
2. What happens if you add more hidden layers? Does it help? Why?

# TASK 5

Data set 

- No regularization
- X1 and X2 as the only features

Question/Task:

1. What is the smallest network architecture you can find with a test lost of less than 0.02?
2. With the smallest architecture, restart the problem a couple of times. Do you notice something?

# TASK 6

Data set 

- No regularization
- Experiment with other features

Question/Task:

1. What is the smallest network architecture you can find with a test lost of less than 0.02?
2. Explain why a smaller architecture is possible.

# TASK 7

Data set 

- Only X1 and X2 as features
- No regularization
- ReLU
- No regularization
- 3 hidden layers (width: 6, 6, 2)

Question/Task:
Look at the weights (thickness of the lines). Do you notice anything? What does it tell you about the capacity of the model?

# TASK 8

Data set 

- Only use X1 and X2 as features

Question/Task:
Find the best network architecture for the data set (aim for a test loss of less than 0.04)

Vary #layers, layer width, learning rate, activation function, and regularization

1. What is the lowest error you are able to achieve?
2. Do you notice any problems with your gradient?
3. How stable is the training?

# TASK 9

Data set 

- Experiment with other features

Question/Task:

1. Are you able to reduce the network size with the same quality?
2. How does the loss function behave?

# TASK 10

Data set 

- Use all features
- Experiment with other hyper parameters

Question/Task:

1. Can you train a 6 hidden layer (width 8) with a test loss of 0.01 or better?
2. What problems did you encounter
3. How did the gradient behave?

# CNN

DATA 1030

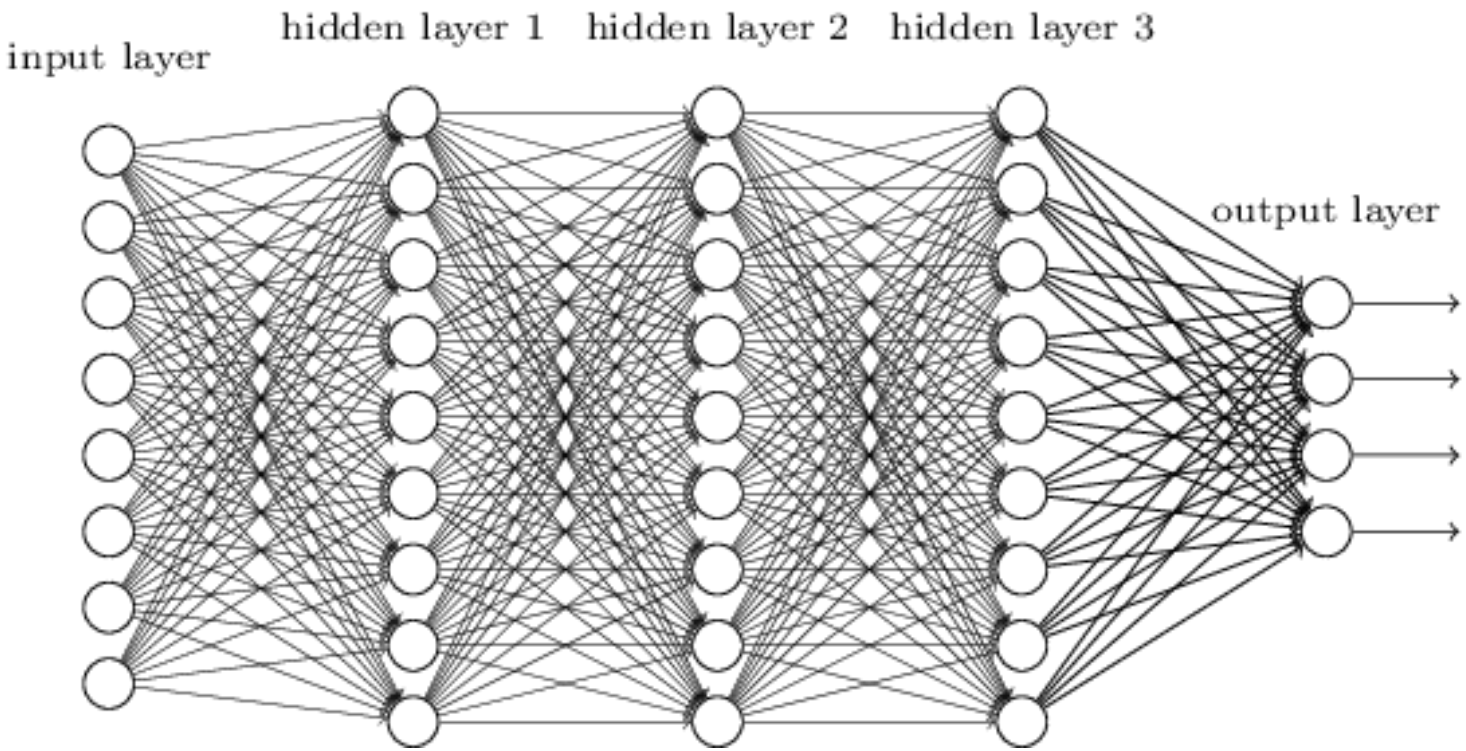
TIM KRASKA

# SMALLER NETWORK: CNN

Why is learning a big model problematic?

From this fully connected model, do we really need all the edges?

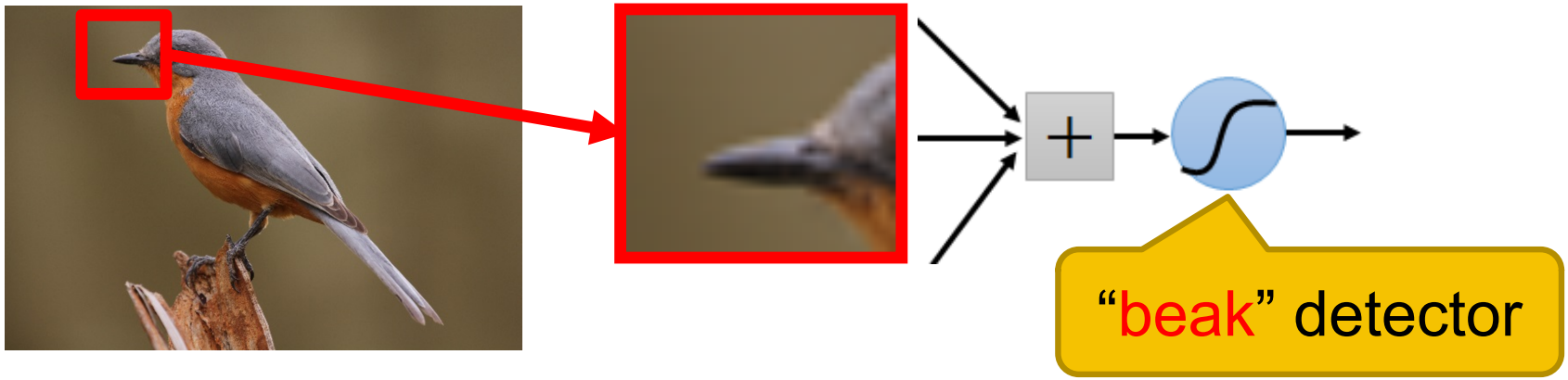
Can some of these be shared?



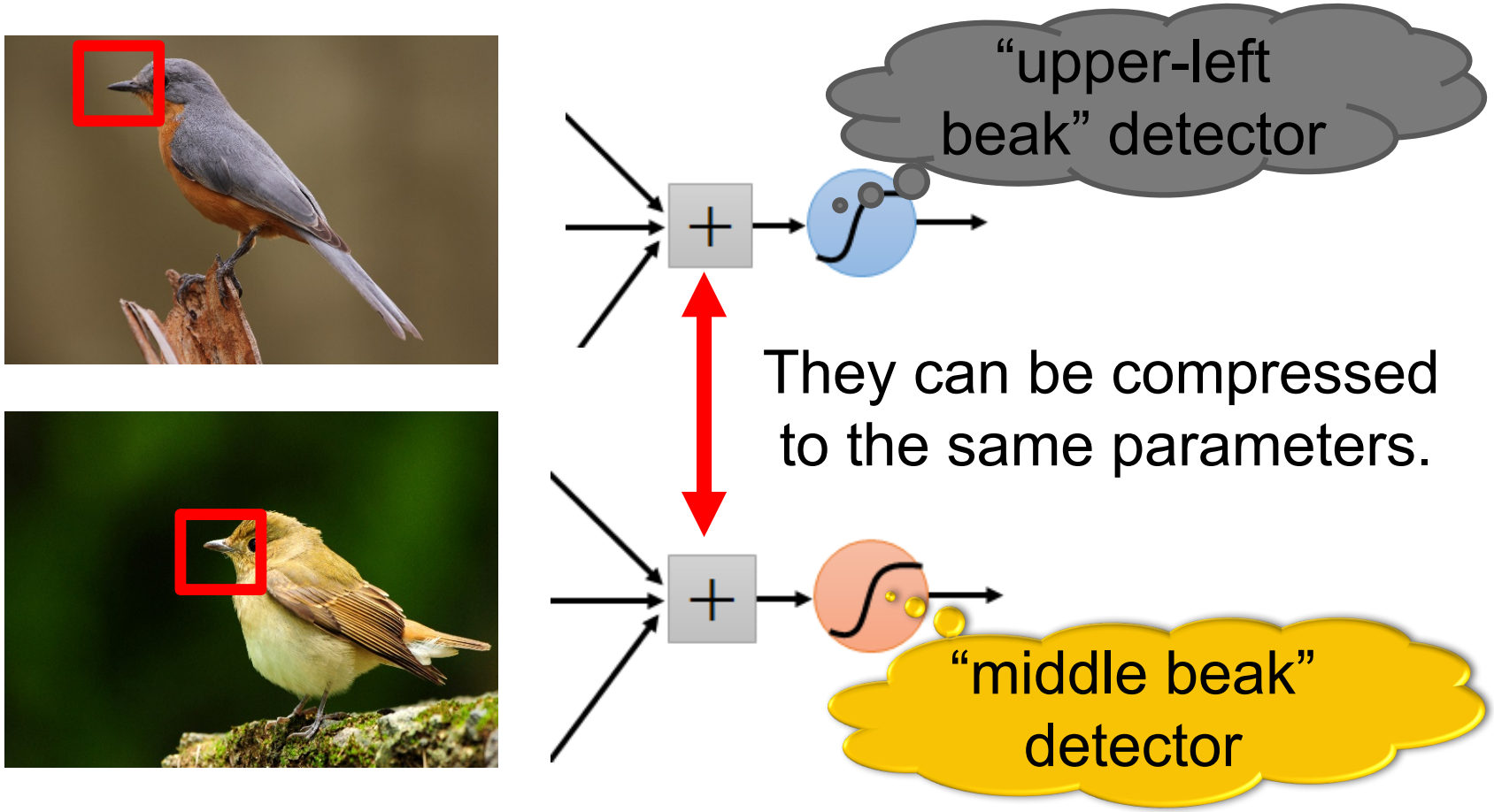
# CONSIDER LEARNING AN IMAGE:

Some patterns are much smaller than the whole image

Can we represent this smaller region with fewer parameters?

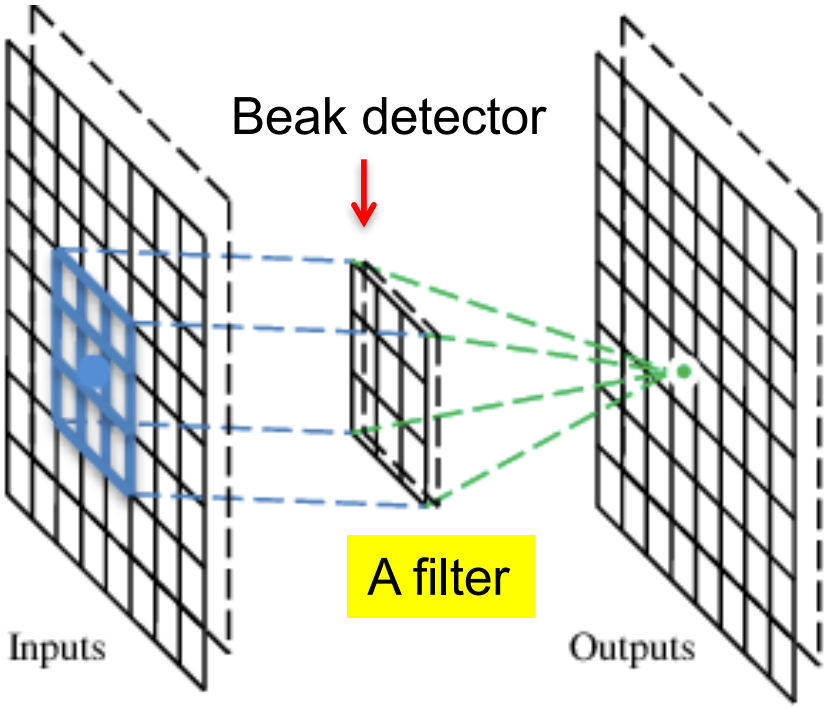


SAME PATTERN APPEARS IN DIFFERENT PLACES:
THEY CAN BE COMPRESSED!
WHAT ABOUT TRAINING A LOT OF SUCH "SMALL" DETECTORS
AND EACH DETECTOR MUST "MOVE AROUND".



# A CONVOLUTIONAL LAYER

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



# CONVOLUTION

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

**These are the network parameters to be learned.**

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

# CONVOLUTION

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

stride=1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

Dot product


6 x 6 image

# CONVOLUTION

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

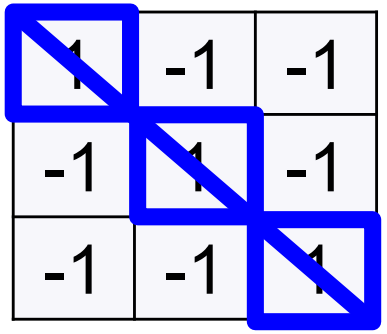
If stride=2

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

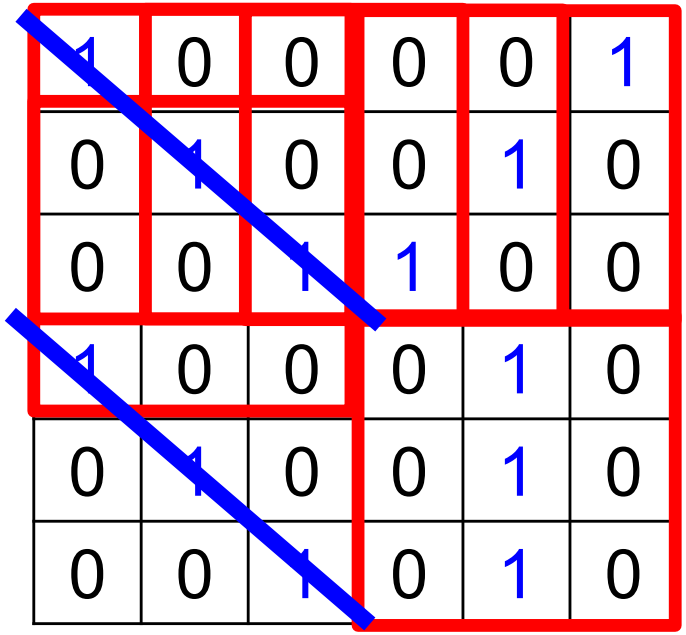
3 -3

# CONVOLUTION

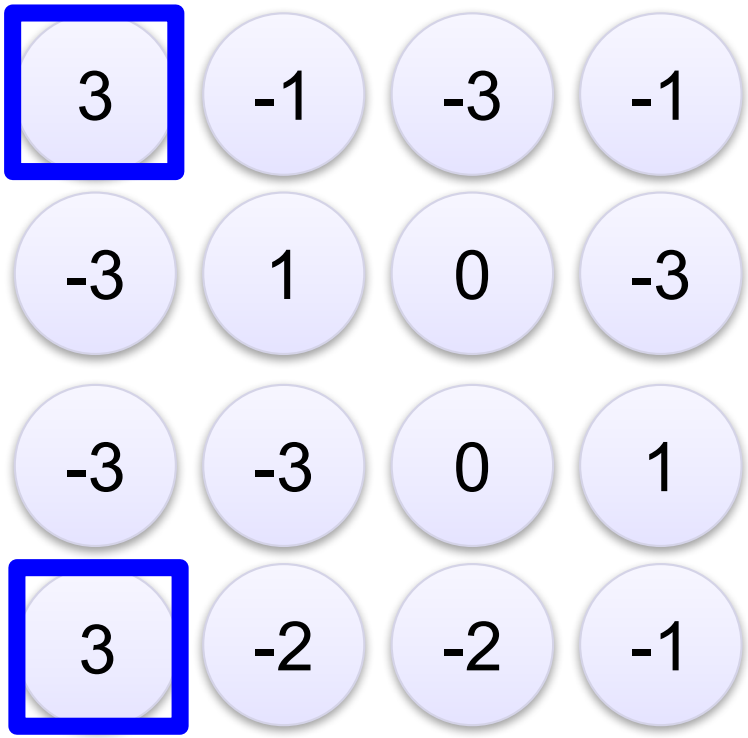



Filter 1

stride=1




6 x 6 image



# CONVOLUTION

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

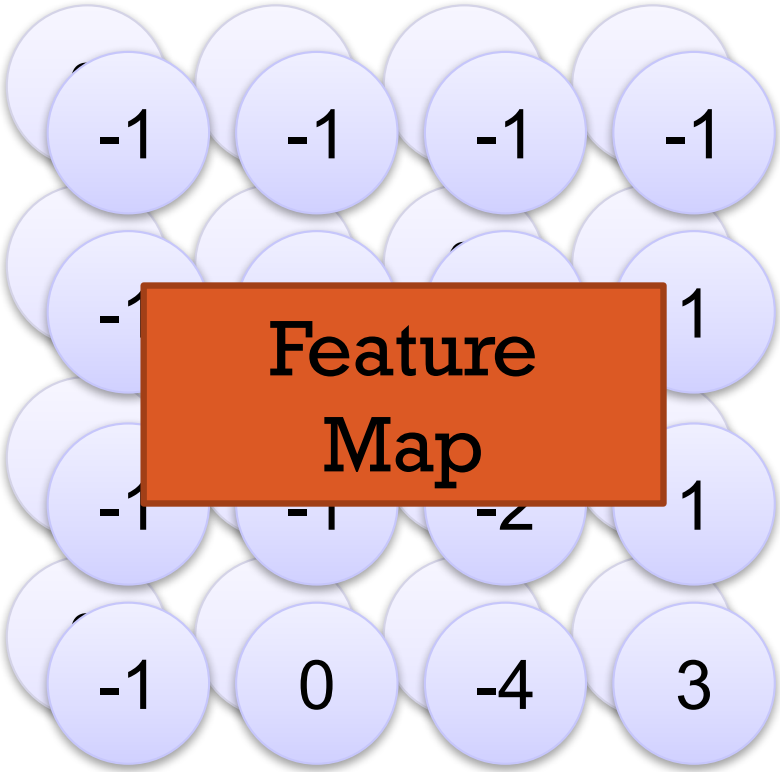
Filter 2

stride=1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

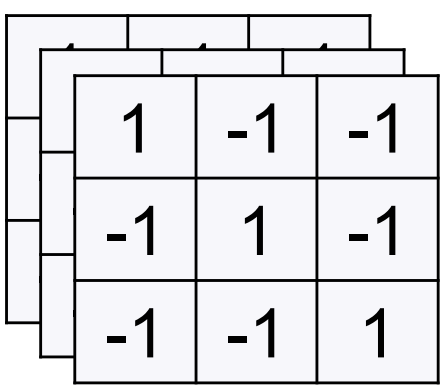
6 x 6 image

Repeat this for each filter

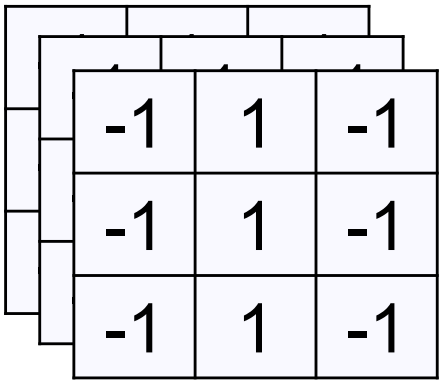



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

# COLOR IMAGE: RGB 3 CHANNELS

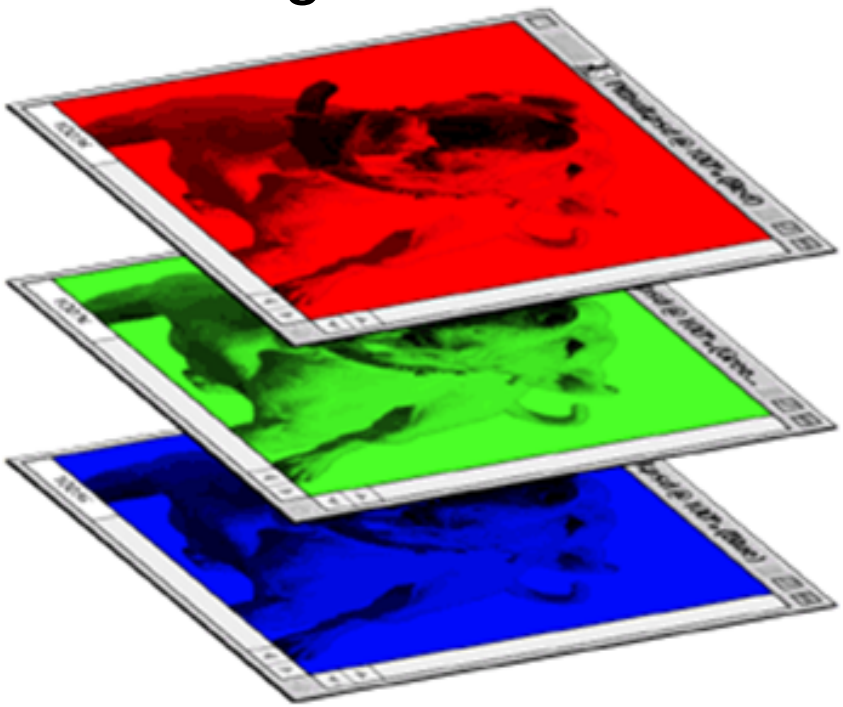


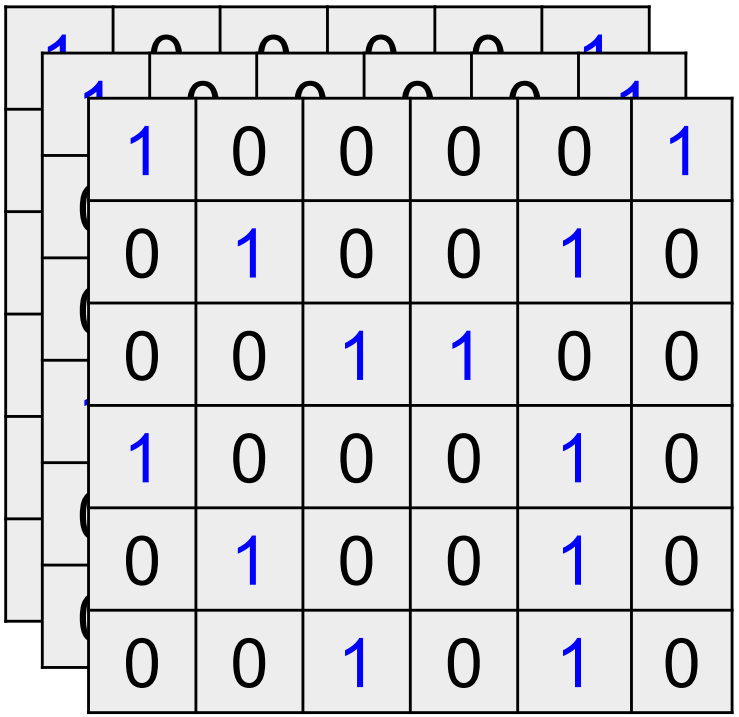

Filter 1



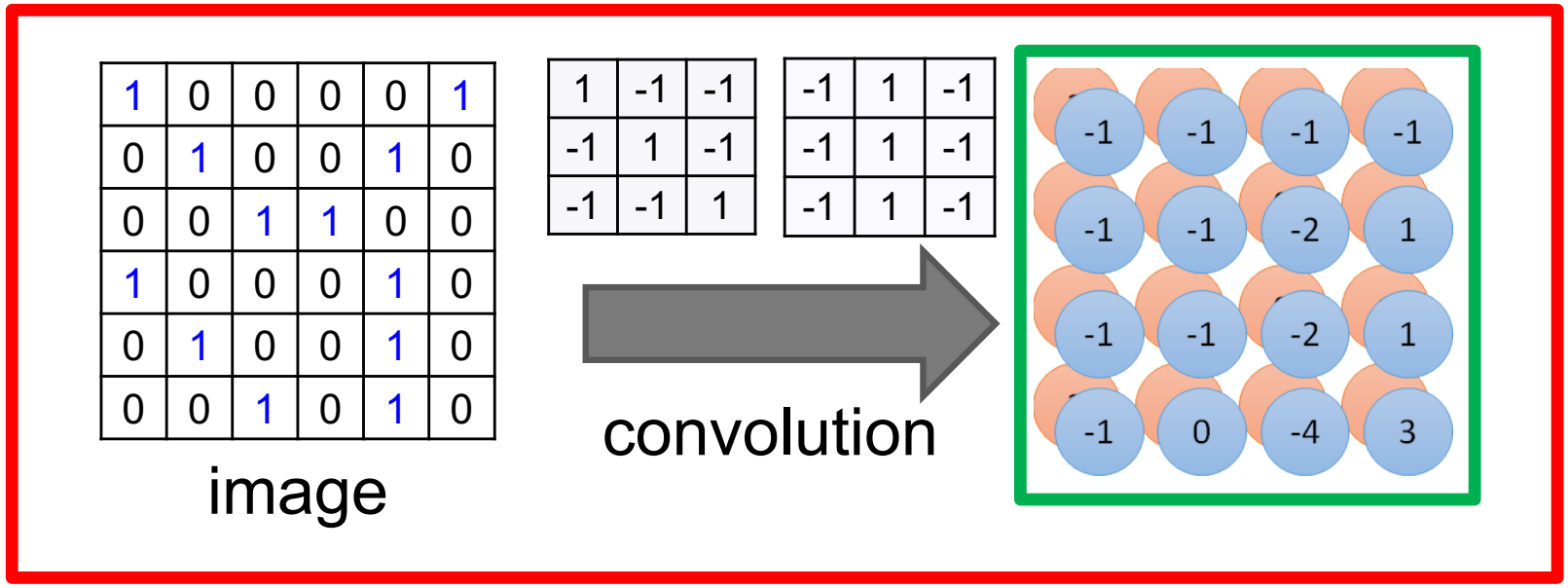

Filter 2

Color image

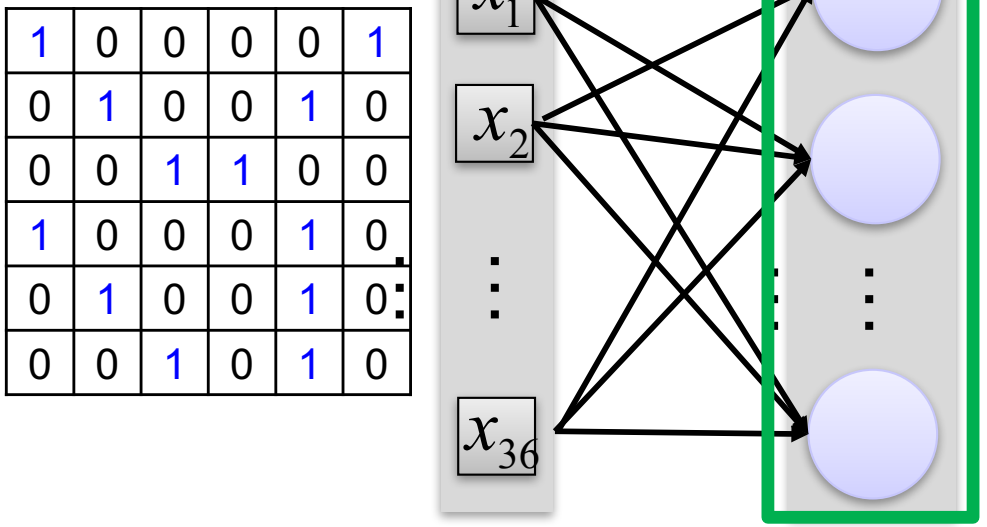


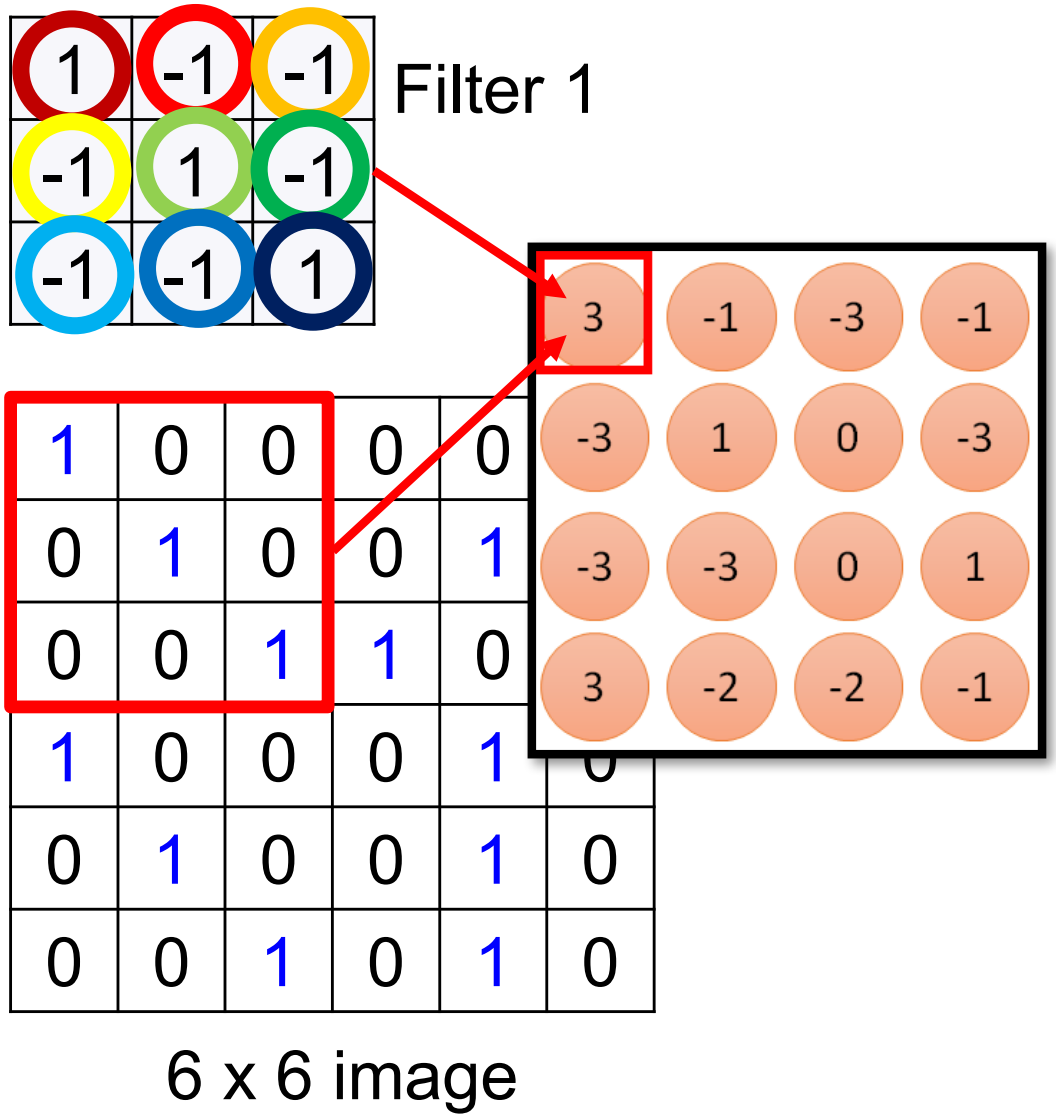


# Convolution v.s. Fully Connected

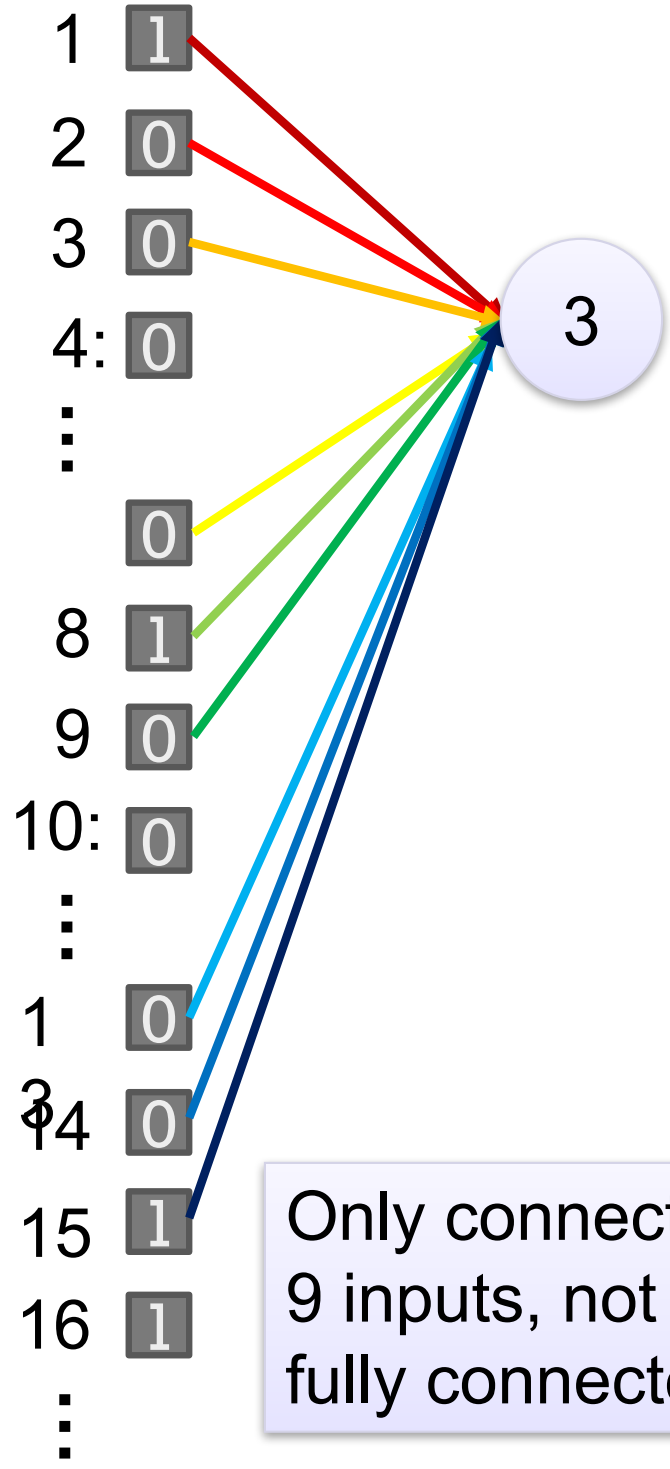



Fully-connected

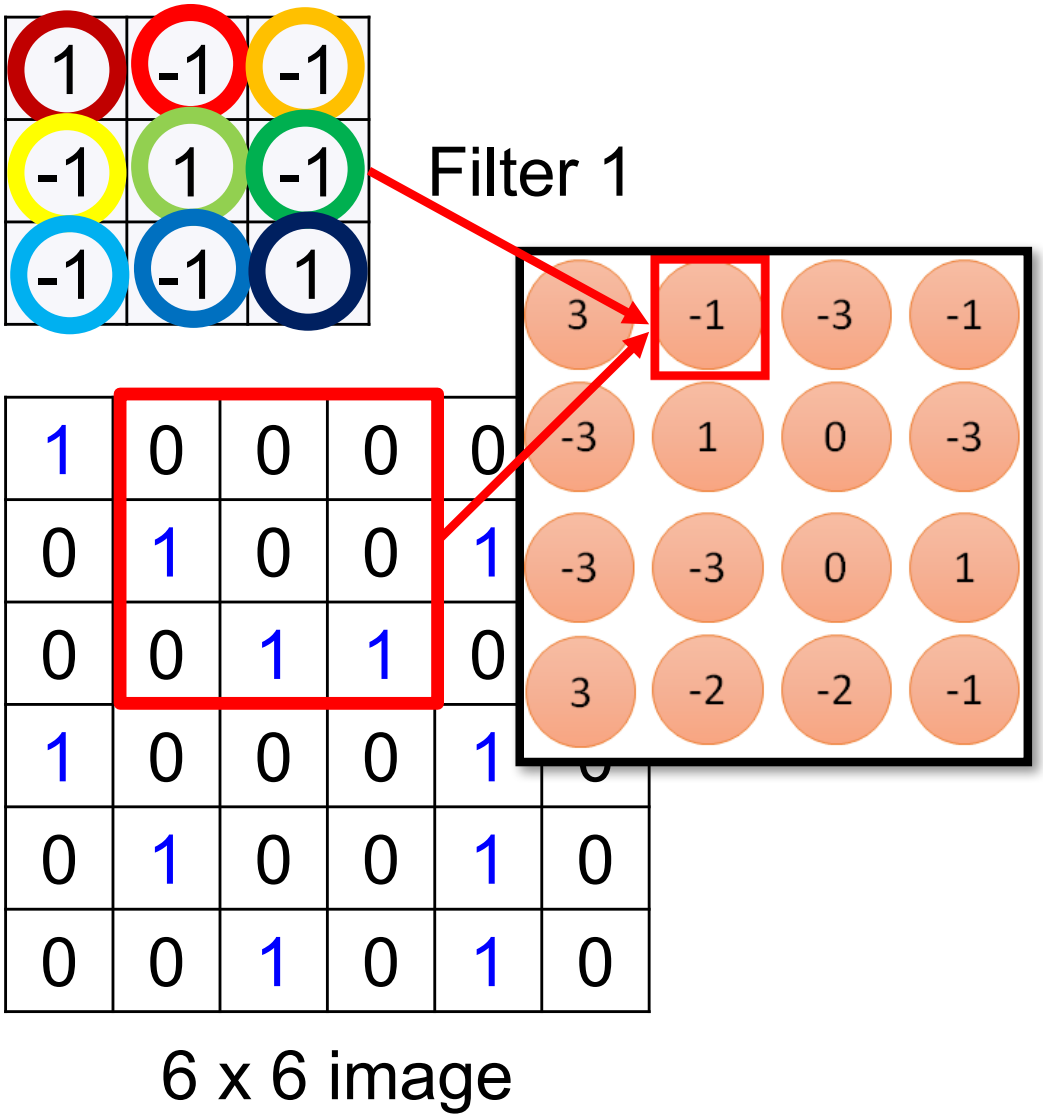





fewer parameters!

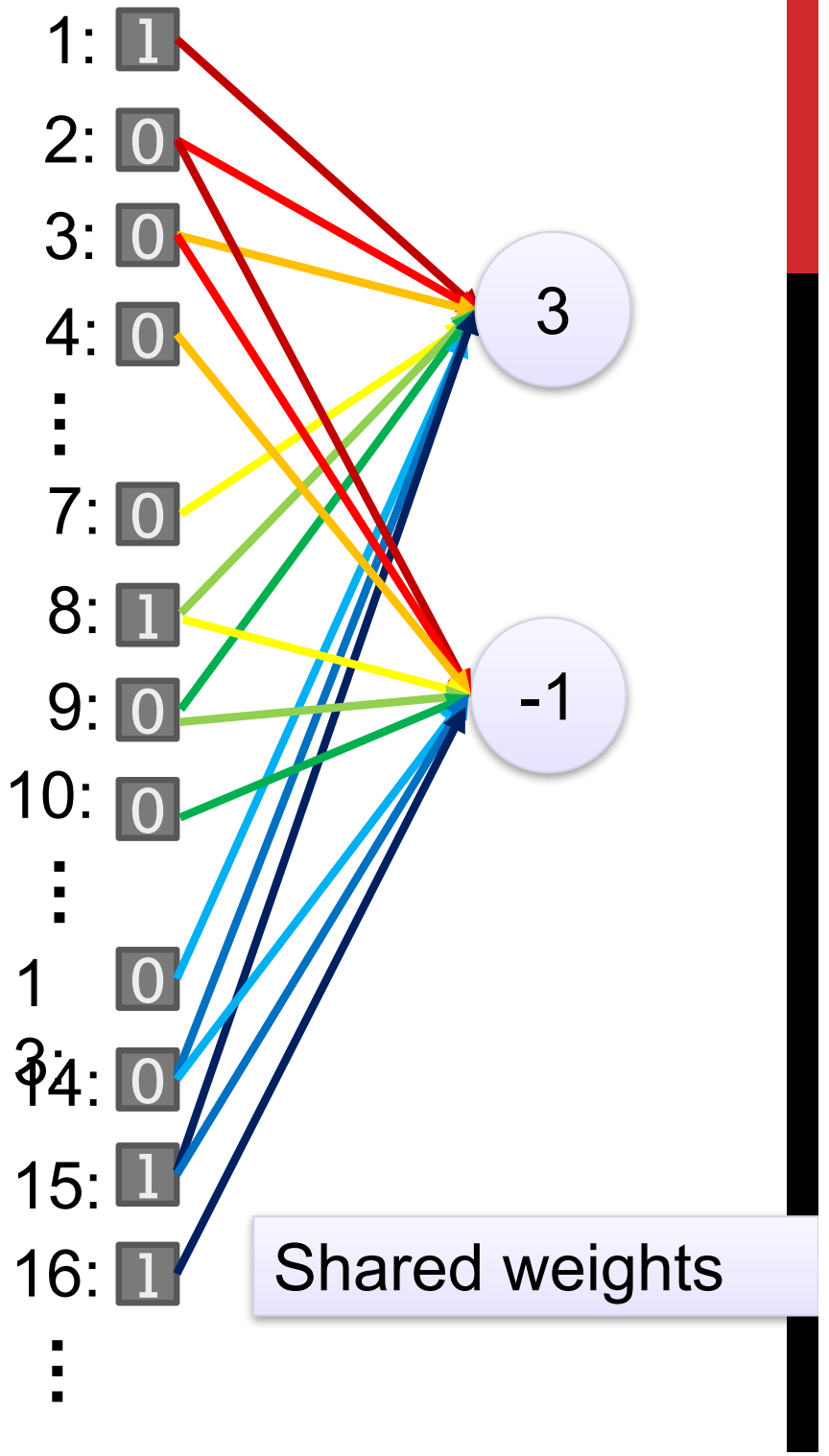



Only connect to 9 inputs, not fully connected

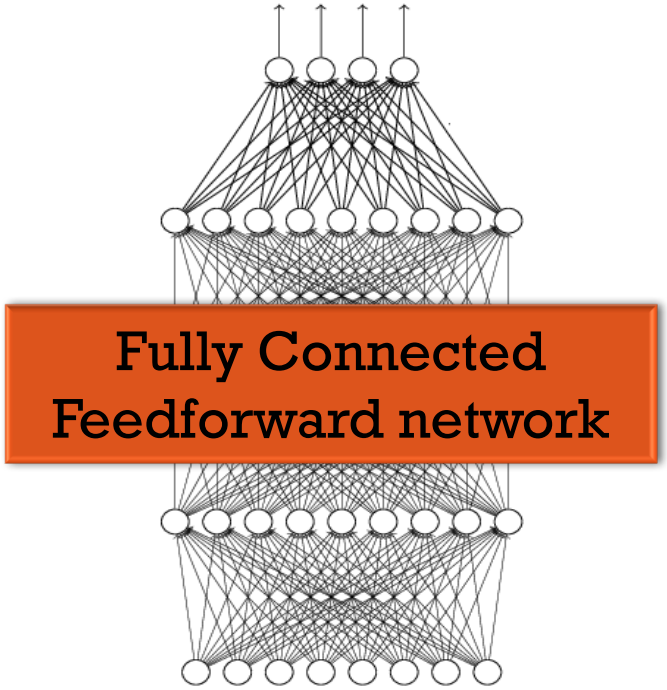



Fewer parameters

Even fewer parameters



# THE WHOLE CNN



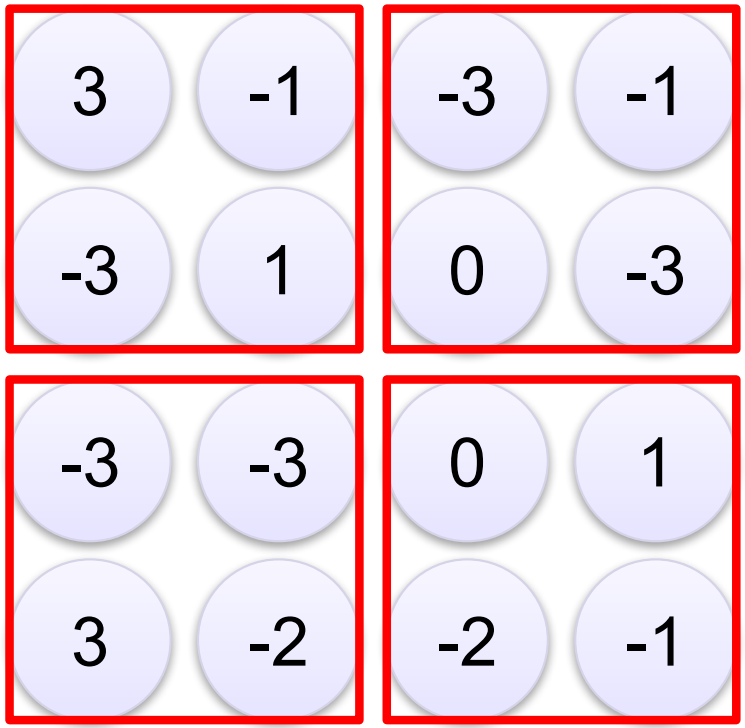
# MAX POOLING

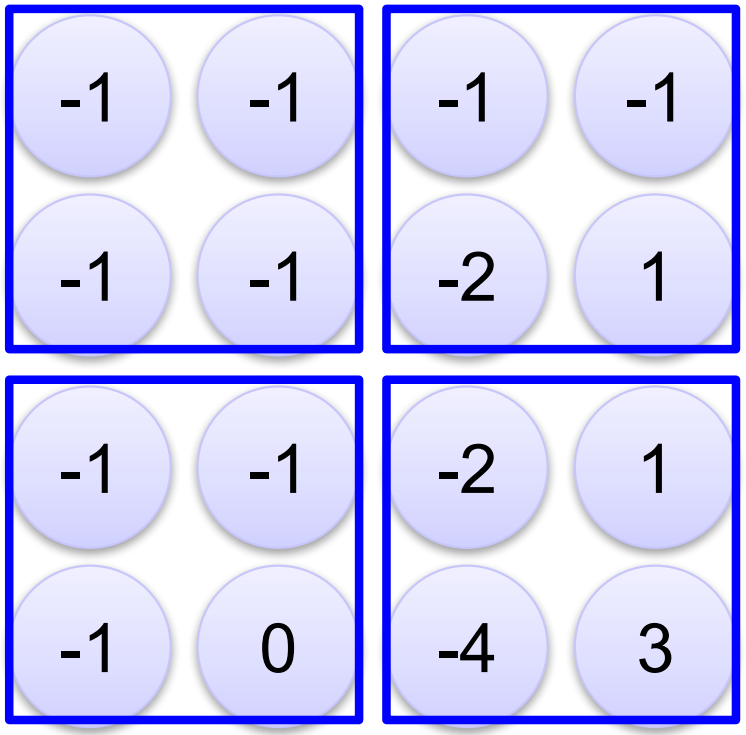
| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2



# WHY POOLING

**Subsampling pixels will not change the object**

bird

Subsampling

bird

We can subsample the pixels to make image smaller

→ fewer parameters to characterize the image

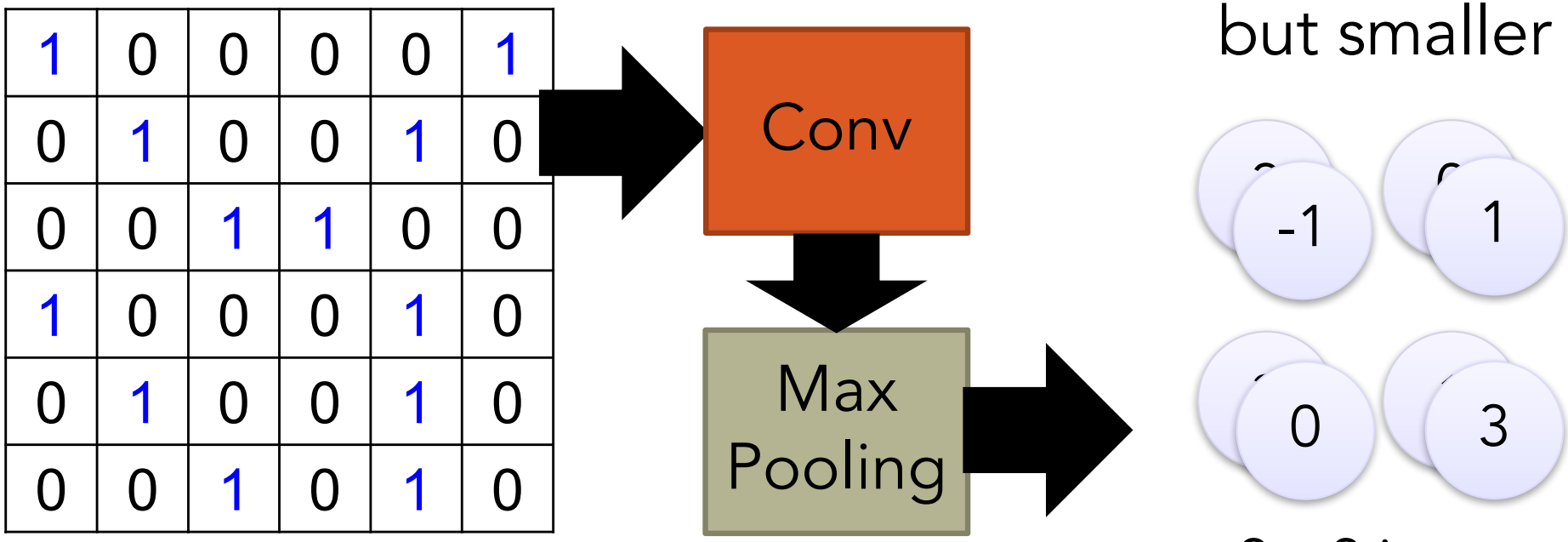
# A CNN COMPRESSES A FULLY CONNECTED NETWORK IN TWO WAYS:

Reducing number of connections

Shared weights on the edges

Max pooling further reduces the complexity

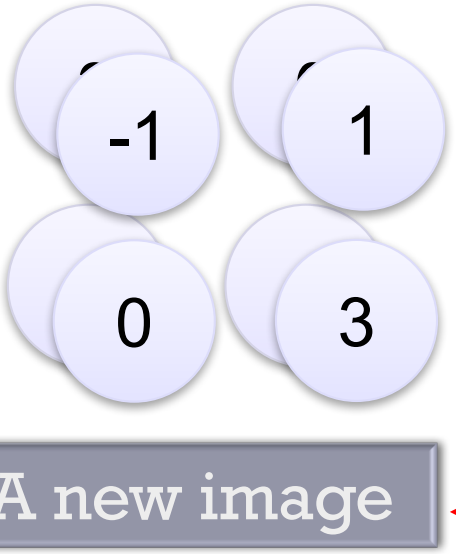
# MAX POOLING




6 x 6 image

Each filter is a channel

# THE WHOLE CNN




Smaller than the original image

The number of channels is the number of filters

# THE WHOLE CNN

Convolution

Max Pooling

A new image

Convolution

Max Pooling

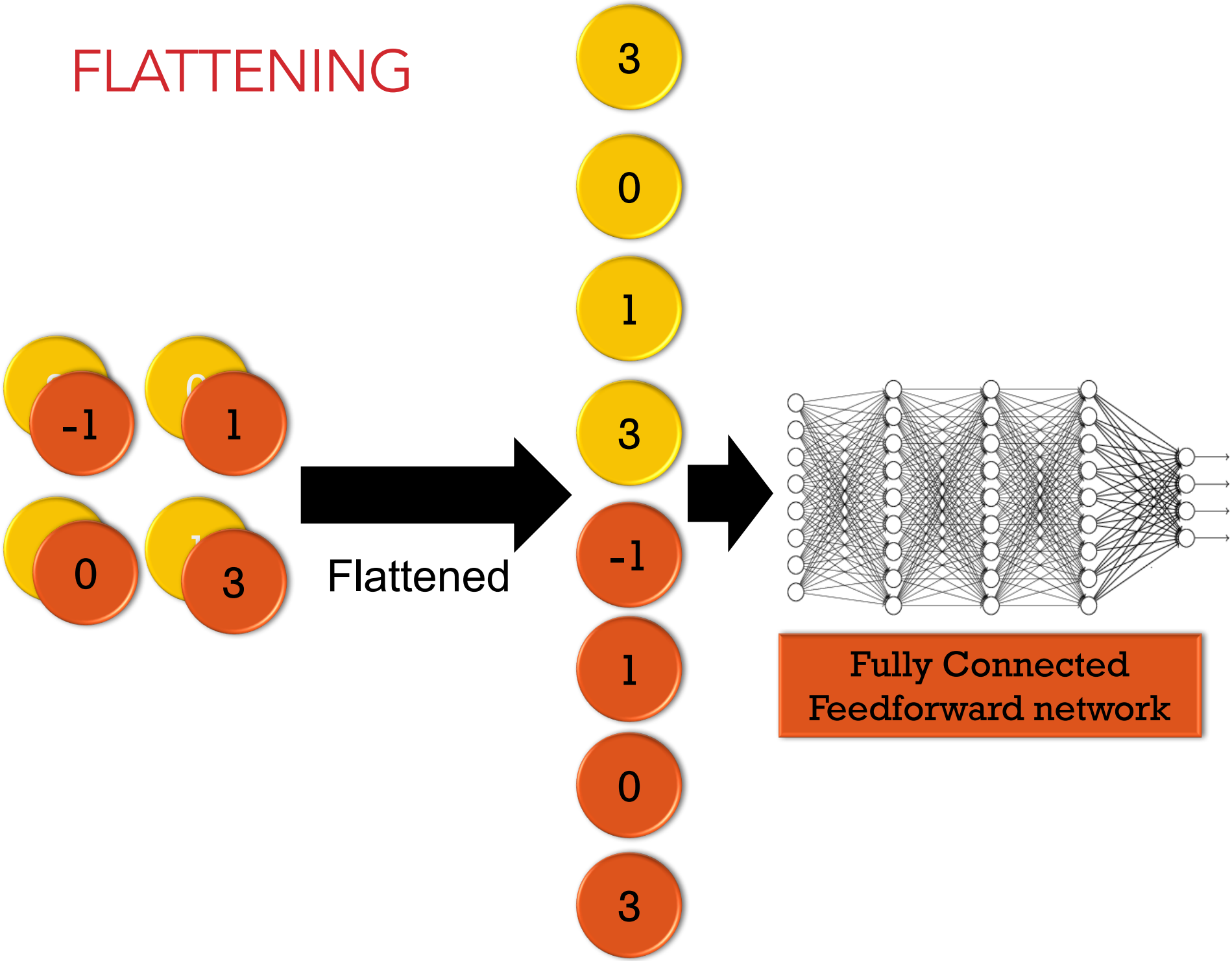
A new image

Flattened

Fully Connected Feedforward network

cat dog ……

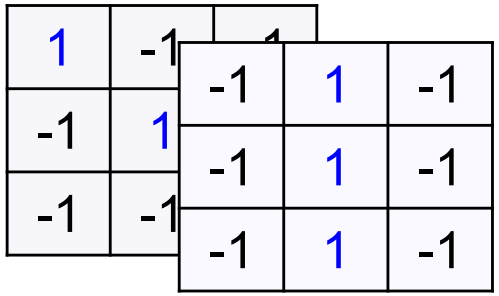
# FLATTENING


-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

# CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(28,28,1)) )
```



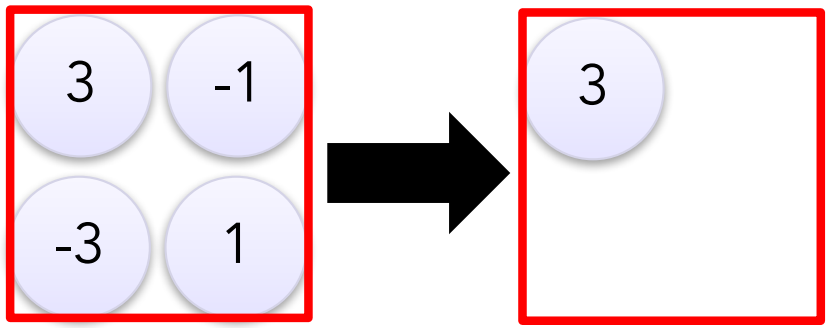

There are 25 3x3 filters.

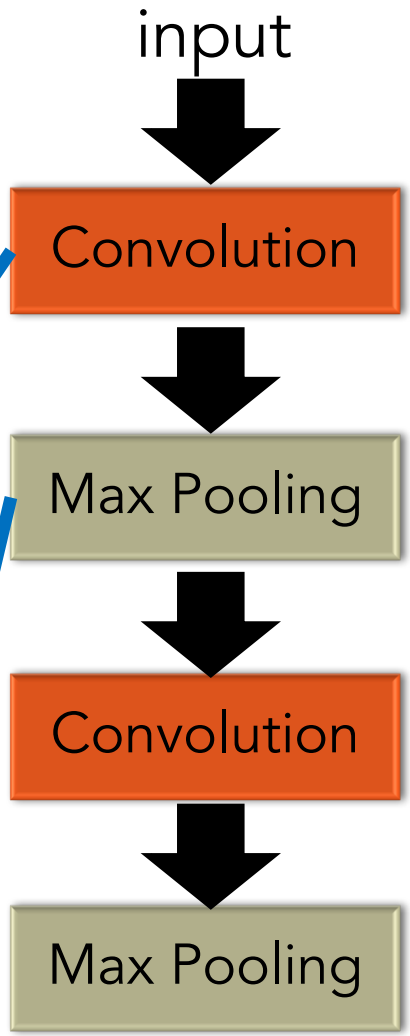
Input_shape = ( 28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

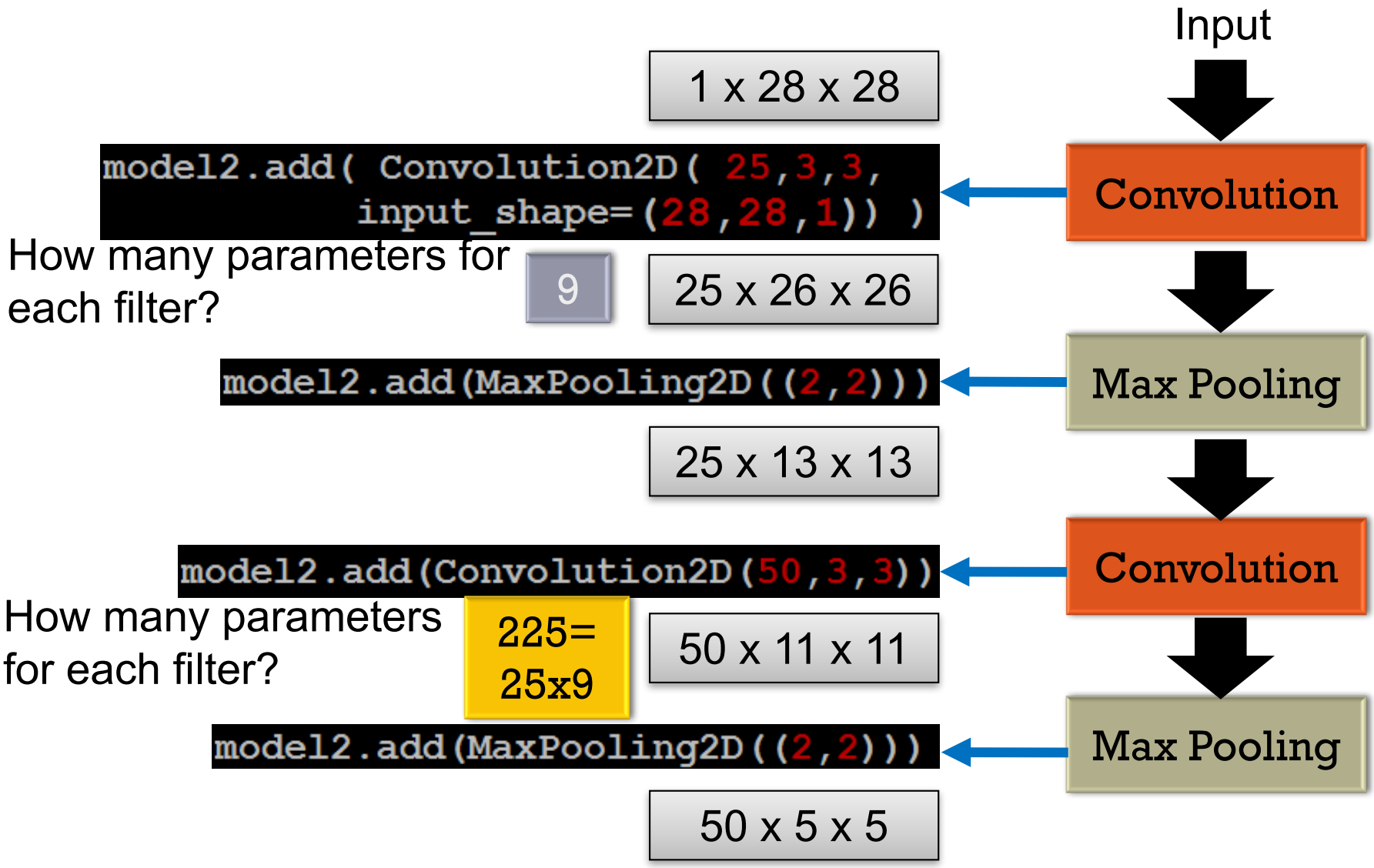
```
model2.add(MaxPooling2D((2,2)))
```



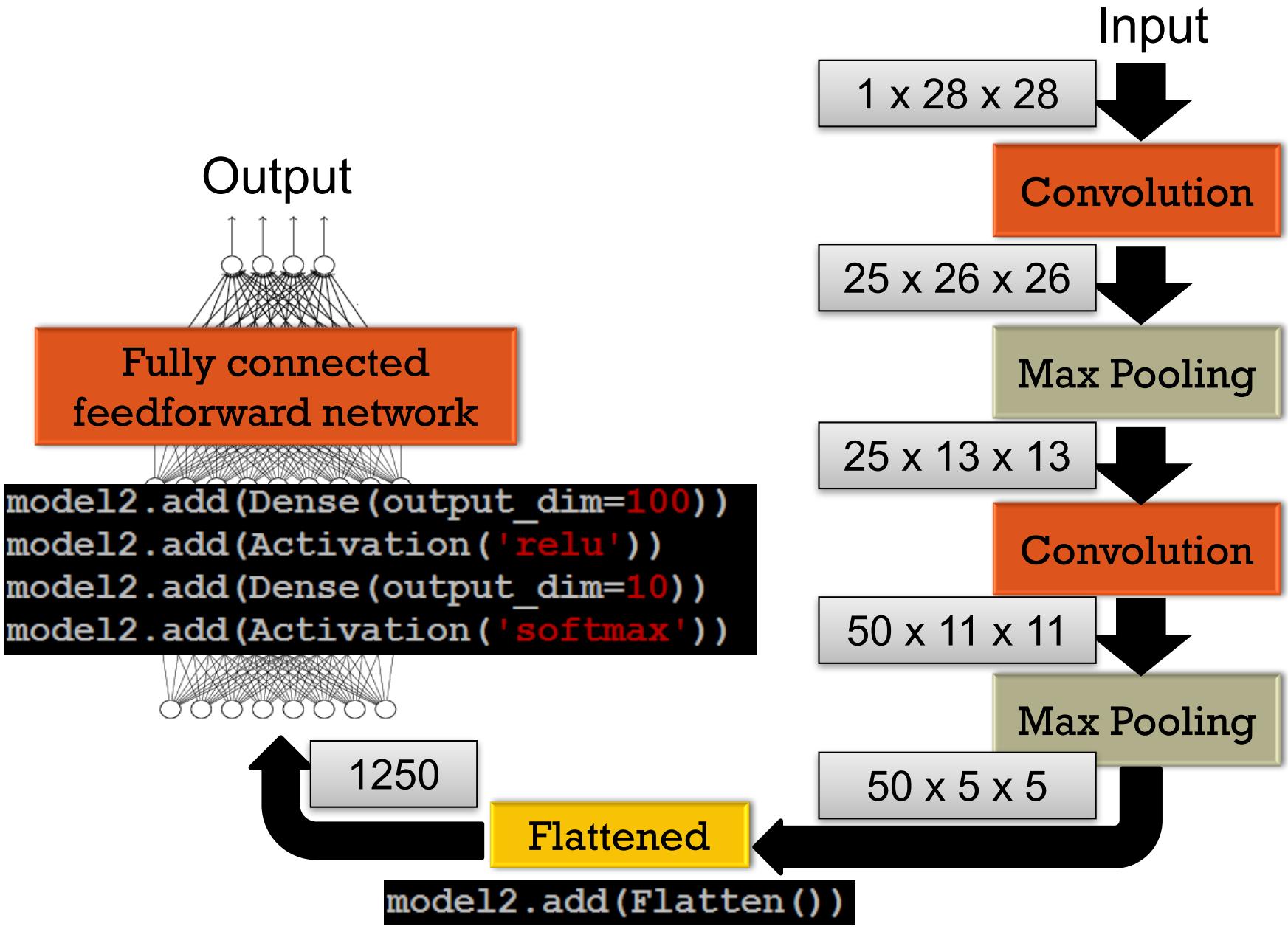
# CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***

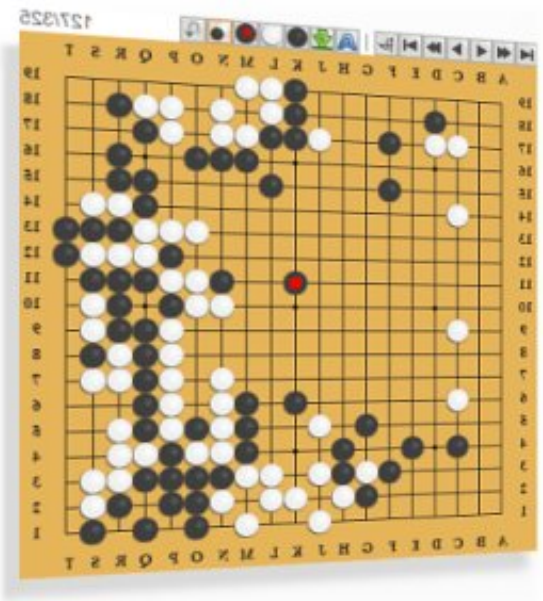


# CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***



# ALPHAGO



19 x 19 matrix

Black: 1

white: -1

none: 0

Neural Network

Next move

(19 x 19 positions)

Fully-connected feedforward network can be used

But CNN performs much better

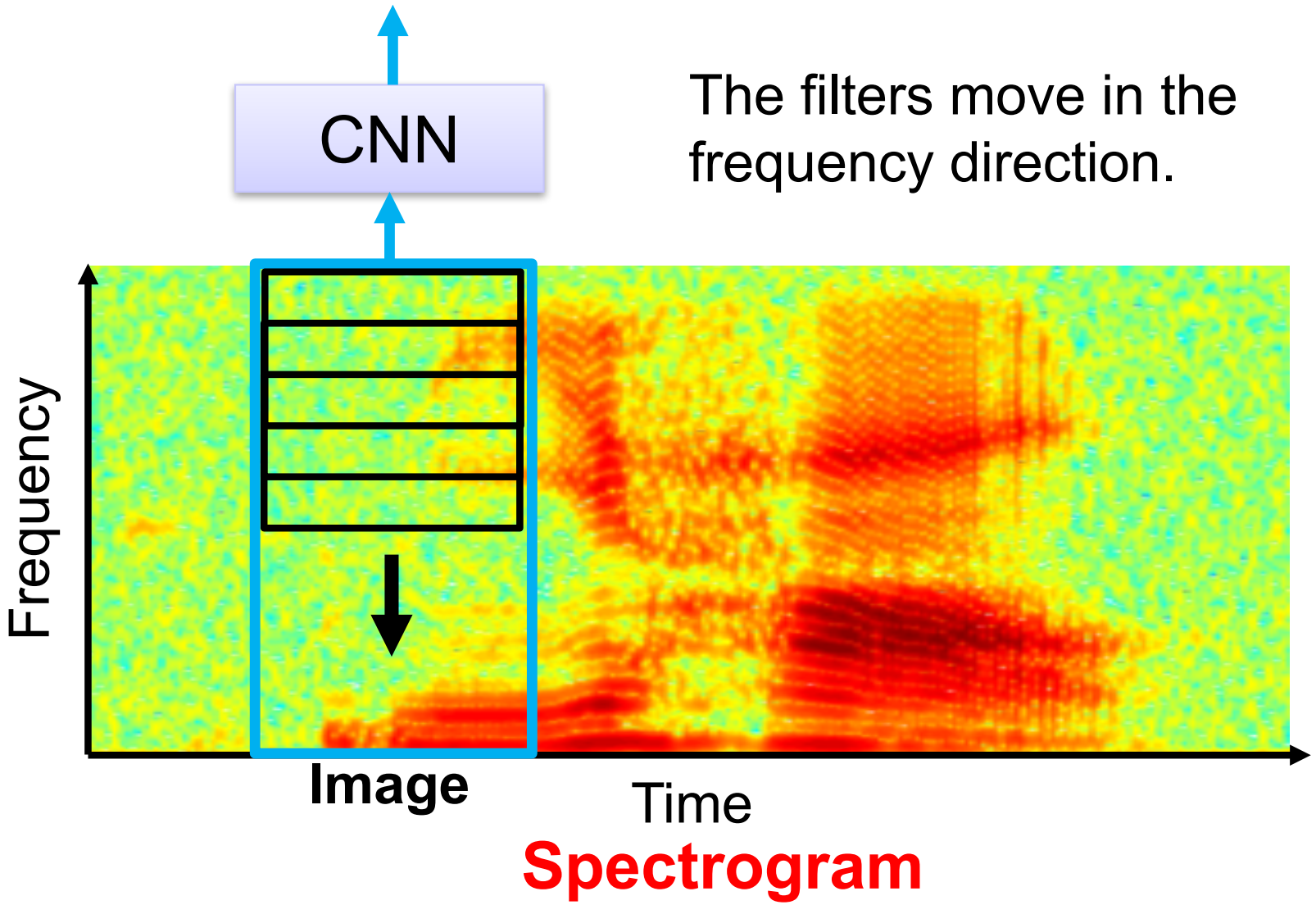
# ALPHAGO'S POLICY NETWORK

The following is quotation from their Nature article:

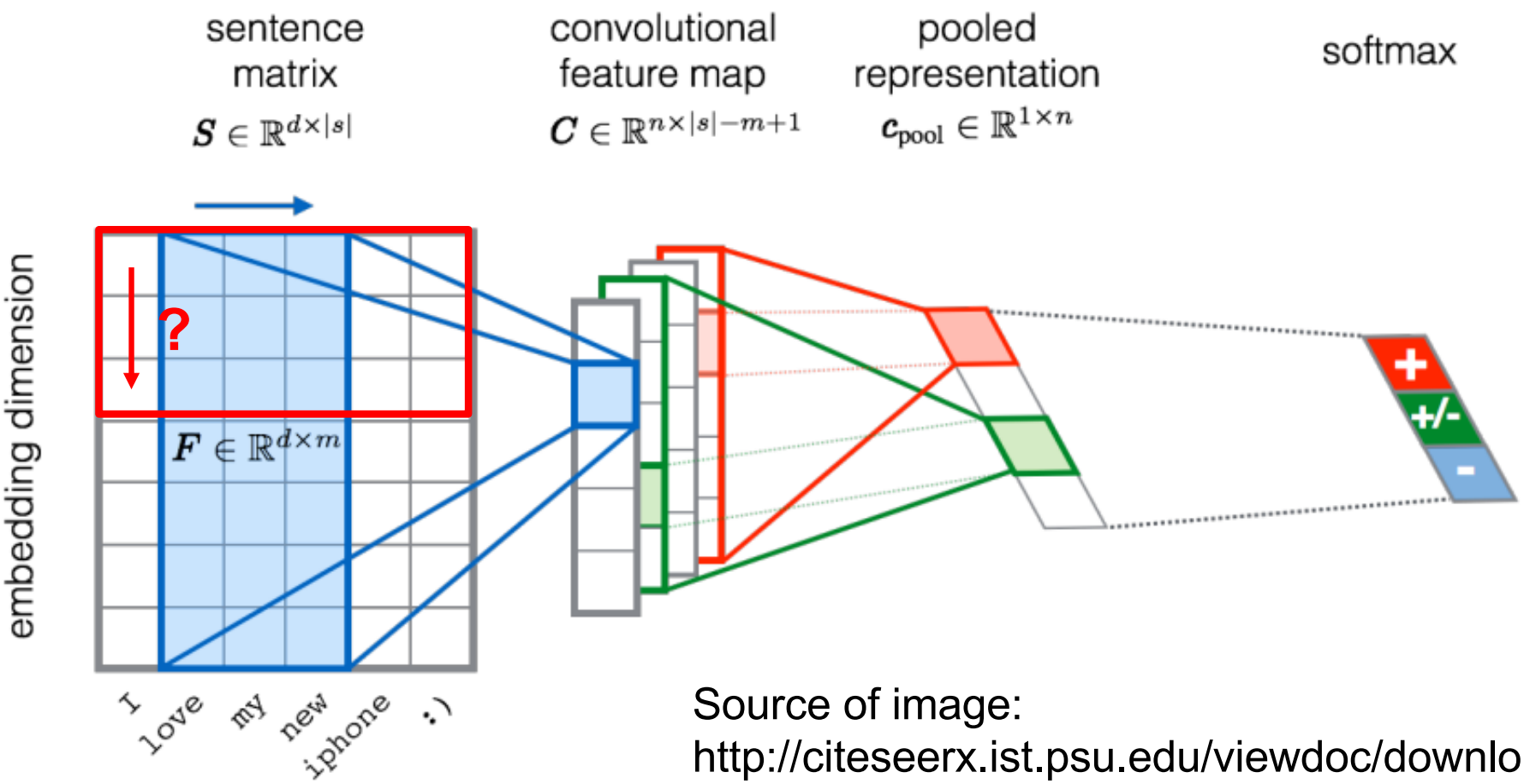
**Note: AlphaGo does not use Max Pooling.**

**Neural network architecture.** The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a $23 \times 23$ image, then convolves $k$ filters of kernel size $5 \times 5$ with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a $21 \times 21$ image, then convolves $k$ filters of kernel size $3 \times 3$ with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size $1 \times 1$ with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

# CNN IN SPEECH RECOGNITION



# CNN IN TEXT CLASSIFICATION




Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf