A Very Brief Introduction to Neural Networks and Deep Learning

Ashley Lee, Isabel Restrepo, & Paul Stey

December 8, 2017

Table of Contents

1. Background
   - Neural Network Basics
   - History
   - Mechanics of Neural Networks
2. CNNs
   - Convolutional Neural Network (CNNs)
   - Real-World Constraints
   - Pooling Layer
   - Normalization Layer
3. RNNs
   - RNNs and LSTMs
4. Autoencoders
   - Autoencoder Basics
   - Stacked and Denoising Autoencoders
5. Summary
   - When to use Neural Networks

What is a neural network?

1. A species of directed acyclic graphs (usually)
2. "Universal function approximator"
3. "An engineering solution to a statistics problem"

What do neural networks do?

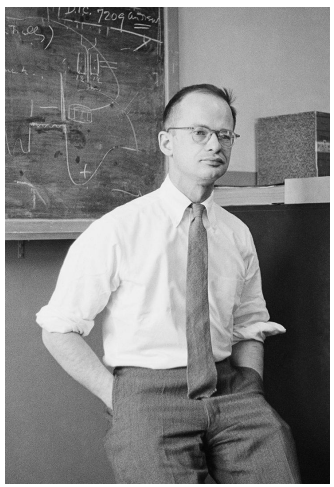
Like many other statistical or machine learning models (e.g., GLM, random forests, boosting), neural networks:

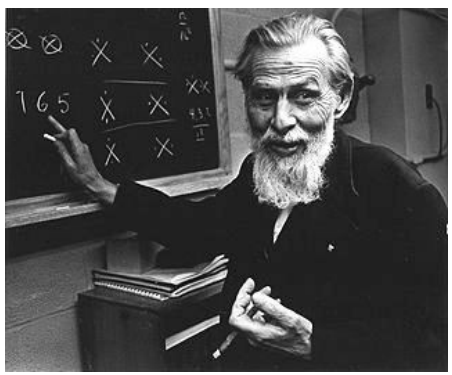
1. Attempt to approximate a data-generating mechanism
2. Can be used for classification problems
3. Can be used for regression problems
4. Can also be used for dimension reduction like principal components analysis (PCA)

History of Neural Networks

The history of neural networks is long and tumultuous.

1. McCulloch and Pitts (1943) "*A Logical Calculus of Ideas Immanent in Nervous Activity*"
2. Rosenblatt (1958) "*The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain*"



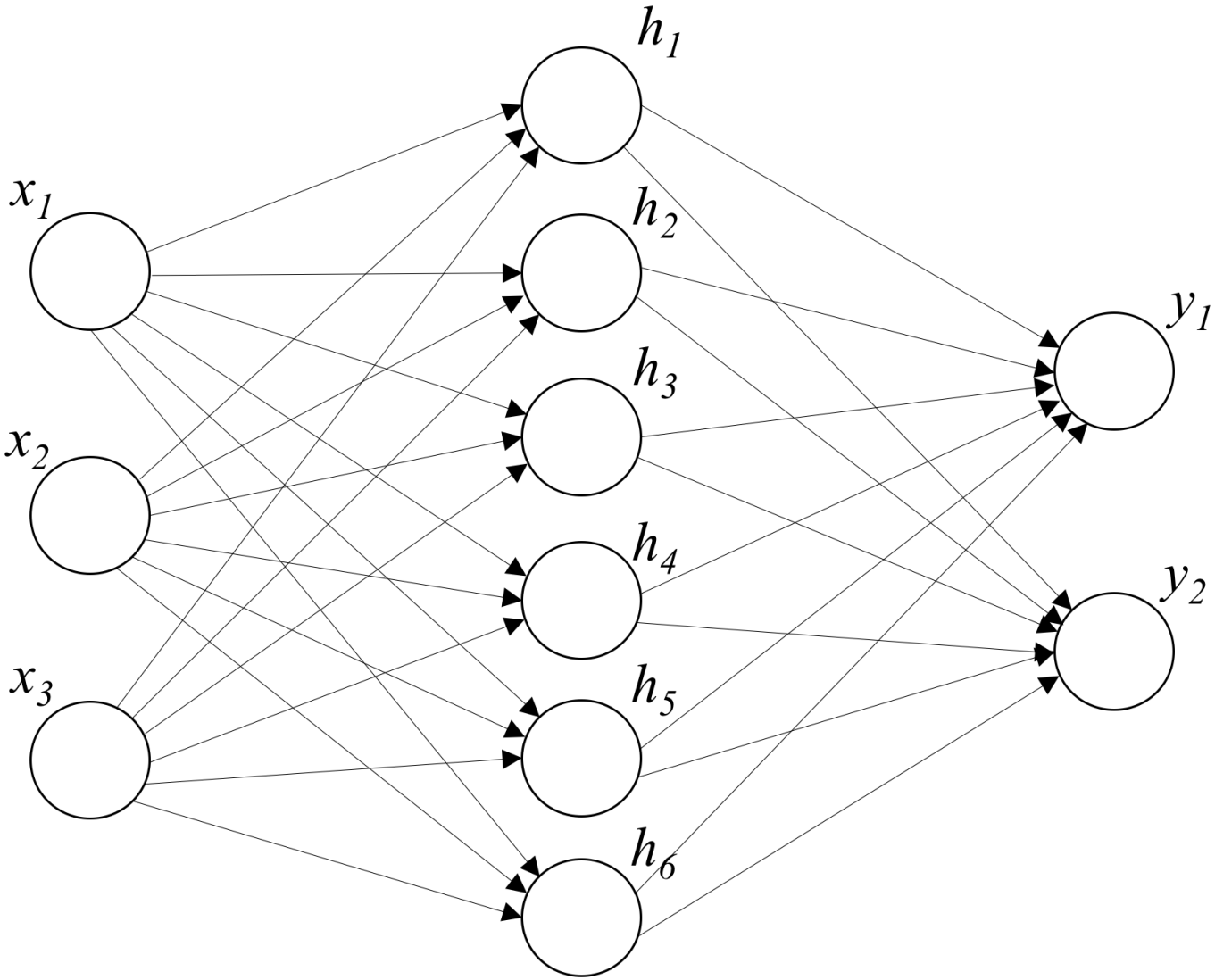


More Recently

Neural networks are experiencing a major resurgence. There are at least two reasons.

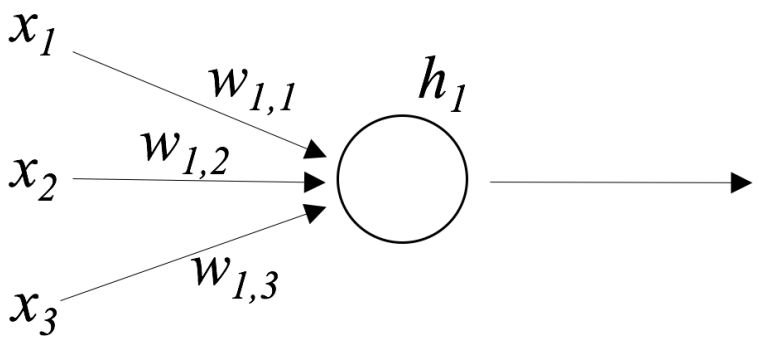
1. Better algorithms for back-propagation
2. GPUs are well suited to building neural networks
   - Matrix multiplies can be made embarrassingly parallel
   - GPUs have much better memory bandwidth
3. More labeled data

The Multilayer Perceptron



Single Neuron

A single neuron takes inputs, x_j, and applies the weights, $w_{.j}$ to the input by computing the dot product of the vectors x and w. The result is the input to the "activation" function.



Activation Functions

The notion of an activation function comes again from the theoretical relationship to neurons in the brain.

Activation functions are analogous to "link" functions in generalized linear models (GLMs).

In fact, one common activation function is the sigmoid function, which is just our old friend the logistic function which you are using when you fit logistic regression models.

Purpose of Activation Functions

There are a few reasons we use activation functions.

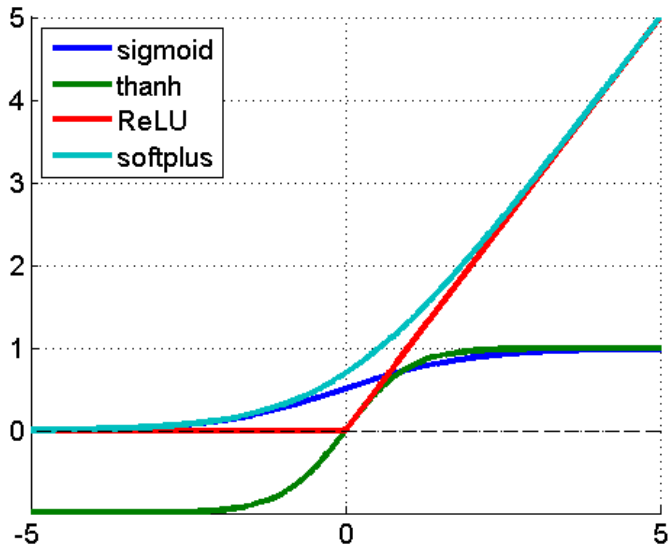
The most basic reason is that—like link functions in GLMs—we want to take some linear predictor and transform it so that it is bounded appropriate. For instance, the value of logistic function is in the range $(0, 1)$.

And the second key reason is that this allows us to introduce non-linearities. Recall that a neural network (like many statistical or machine learning models) is trying to approximate a data-generating mechanism. So we are trying to approximate a function that might be very complex and include many non-linearities.

Common Activation Functions

Some common activation functions include the following:

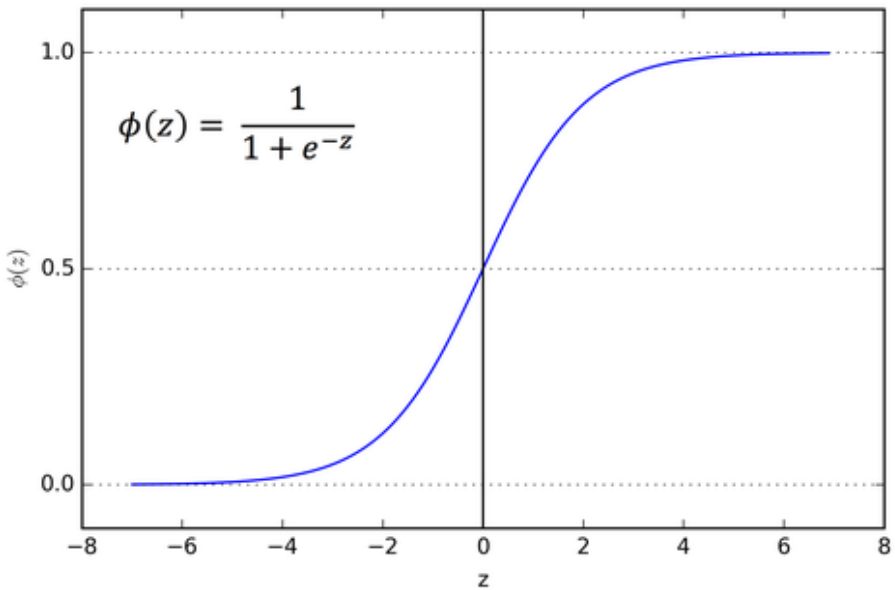
1. Sigmoid (i.e., logistic)
2. Hyperbolic tangent: $tanh$
3. Rectified linear unit (ReLU)
4. softplus



Sigmoid Functions

Sigmoid function: $\phi(z) = \frac{1}{1+e^{-z}}$

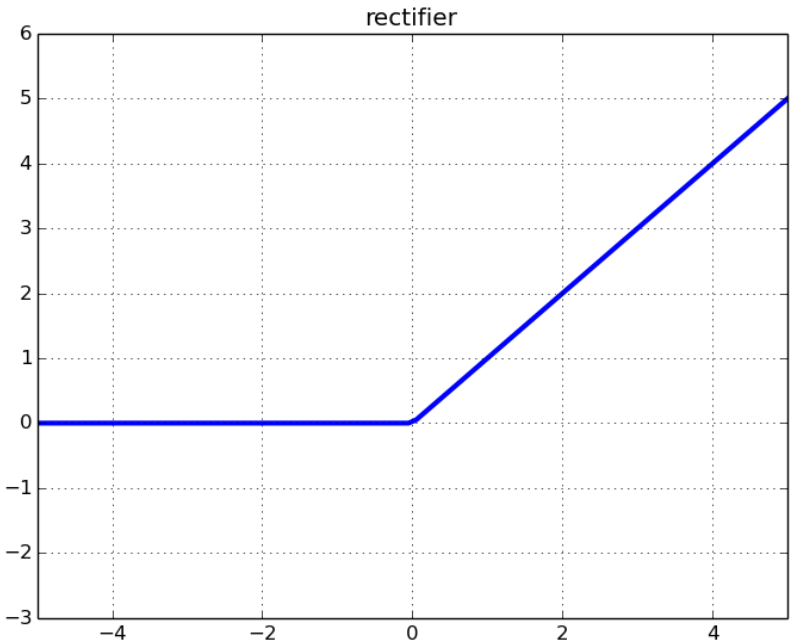
1. Range between 0 and 1
2. Sometimes interpreted as probability
3. Special case of the softmax: $\psi(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$, which is used for vector-value outcome



Rectified Linear Unit (ReLU)

ReLU: $f(z) = max(0, z)$

1. Range between 0 and $\inf$
2. Biological justification



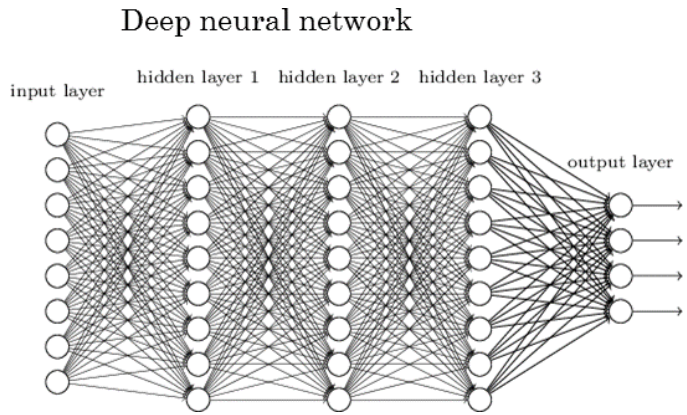
Activation Functions

For the hidden layers of neural networks, ReLUs—or some variation thereof—tend to be the most common.

Sigmoid (and probably $tanh$) are not frequently used in the hidden layers more because of the vanishing gradient problem (discussed later).

Convolutional Neural Networks

Why do we need ConvNets?

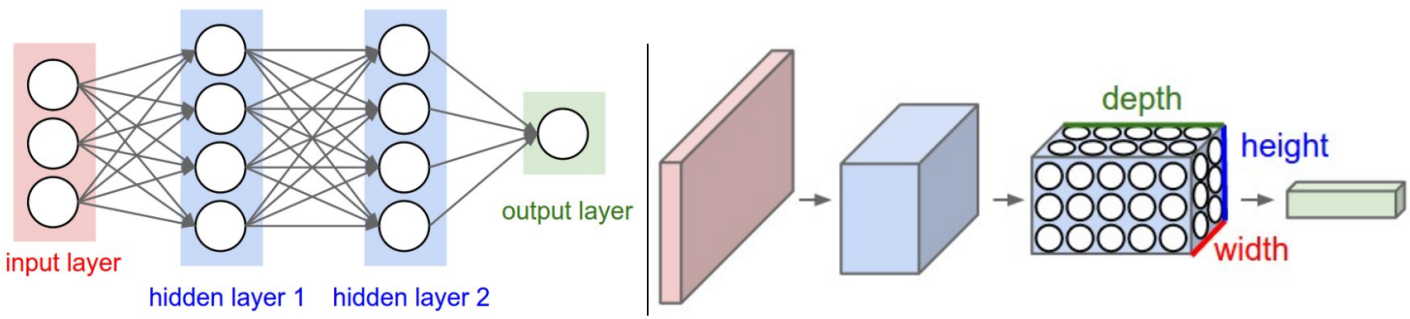


1. Regular neural nets don't scale well to images
   - For images of size $32 \times 32 \times 3$, a *single* fully-connected neuron in the first layer would have 3072 weights.
   - Images of size $200 \times 200 \times 3$, a *single* gives 120000 weights.
2. Full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

CNNs

What are ConvNets?

1. ConvNets are very similar to neural networks discussed thus far. Dot product, followed by non-linearity, and loss function at the end.
2. Explicit assumption that input are images.
3. Layers have neurons arranged in 3 dimensions (width, height, depth) to form an **activation volume**



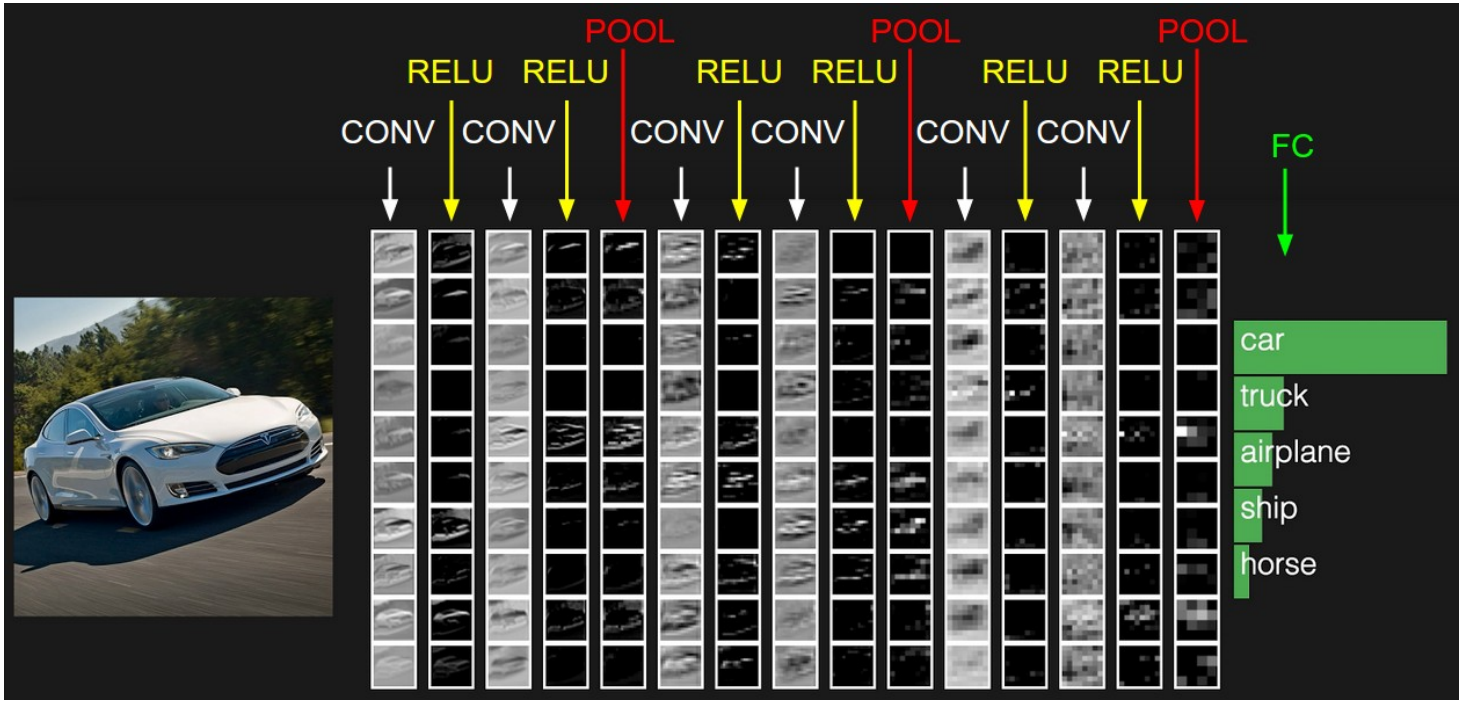
Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

CNN Architecture

Types of layers used to build ConvNets

1. Convolutional Layer
   - Input: 3-d volume
   - Output: 3-d volume
   - Convolve "filters" with small regions in the image
   - Output depth, depends on the number of filters
2. Pooling Layer
   - Downsampling along spatial dimensions (width, height)
3. Fully-Connected Layer (what we've seen so far)
   - Compute class score. Dimensions are transformed to $1 \times 1 \times k$, where k is number of classes

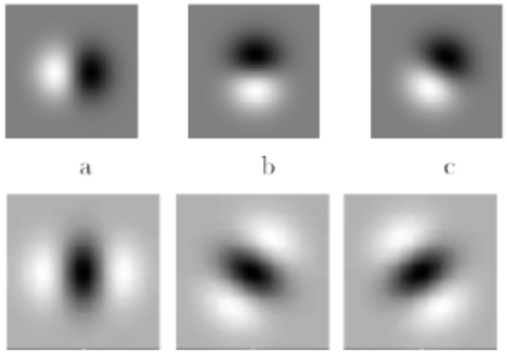
CNN Architecture



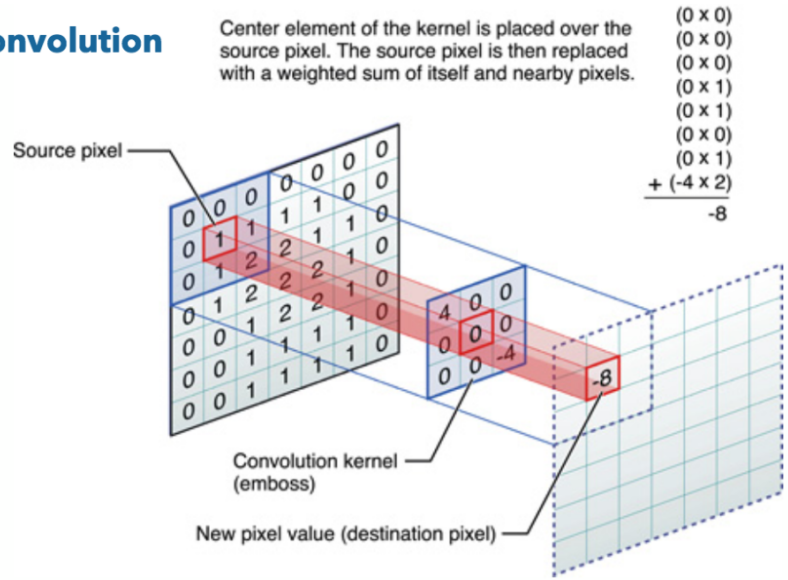
1. CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (weights and biases of neurons).
2. RELU/POOL layers will implement a fixed function (e.g., ReLU $= max(0, x)$)

Classical Computer Vision

1. Filter Design. E.g., Steerable filters - Gaussian and it's derivatives

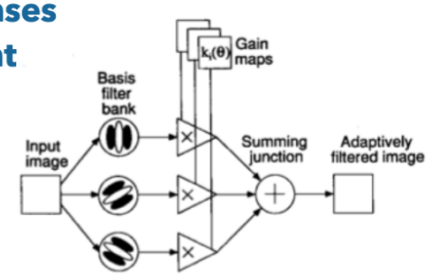


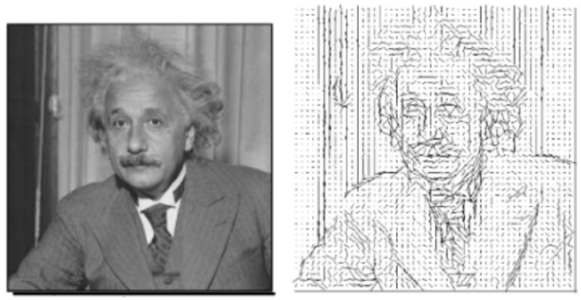
2. Convolution



3. Individual responses (apply a, b, c)

4. Compose responses (optional) - gradient orientation



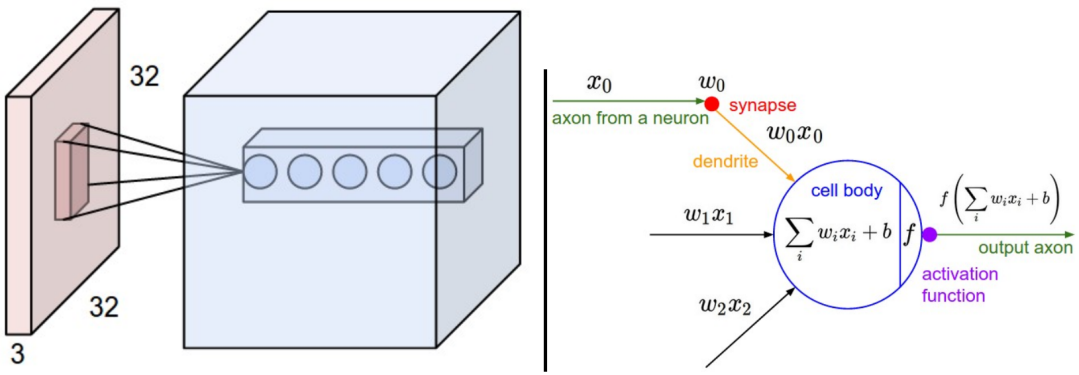


Convolutional Layer - Local Connectivity

Connecting each Neuron to the Input Volume

1. **Receptive Field (hyper-parameter):** Spatial extend of each neuron = filter size
2. Connections are local in space (along width and height), but always full along the entire depth of the input volume.

   - Example: Suppose an input volume had size $16 \times 16 \times 20$. Then using an example receptive field size $3 \times 3 \times 20 = 180$ connections to the input volume. Notice that, again, the connectivity is local in space (i.e., 3×3), but full along the input depth (20)



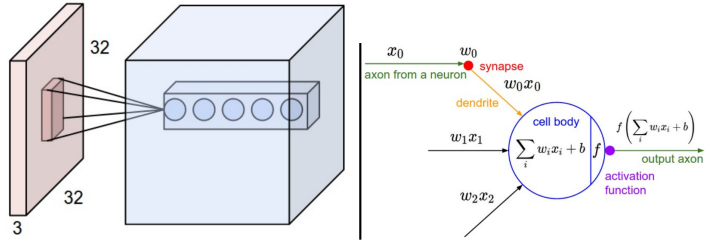
Convolutional Layer - Spatial Arrangement

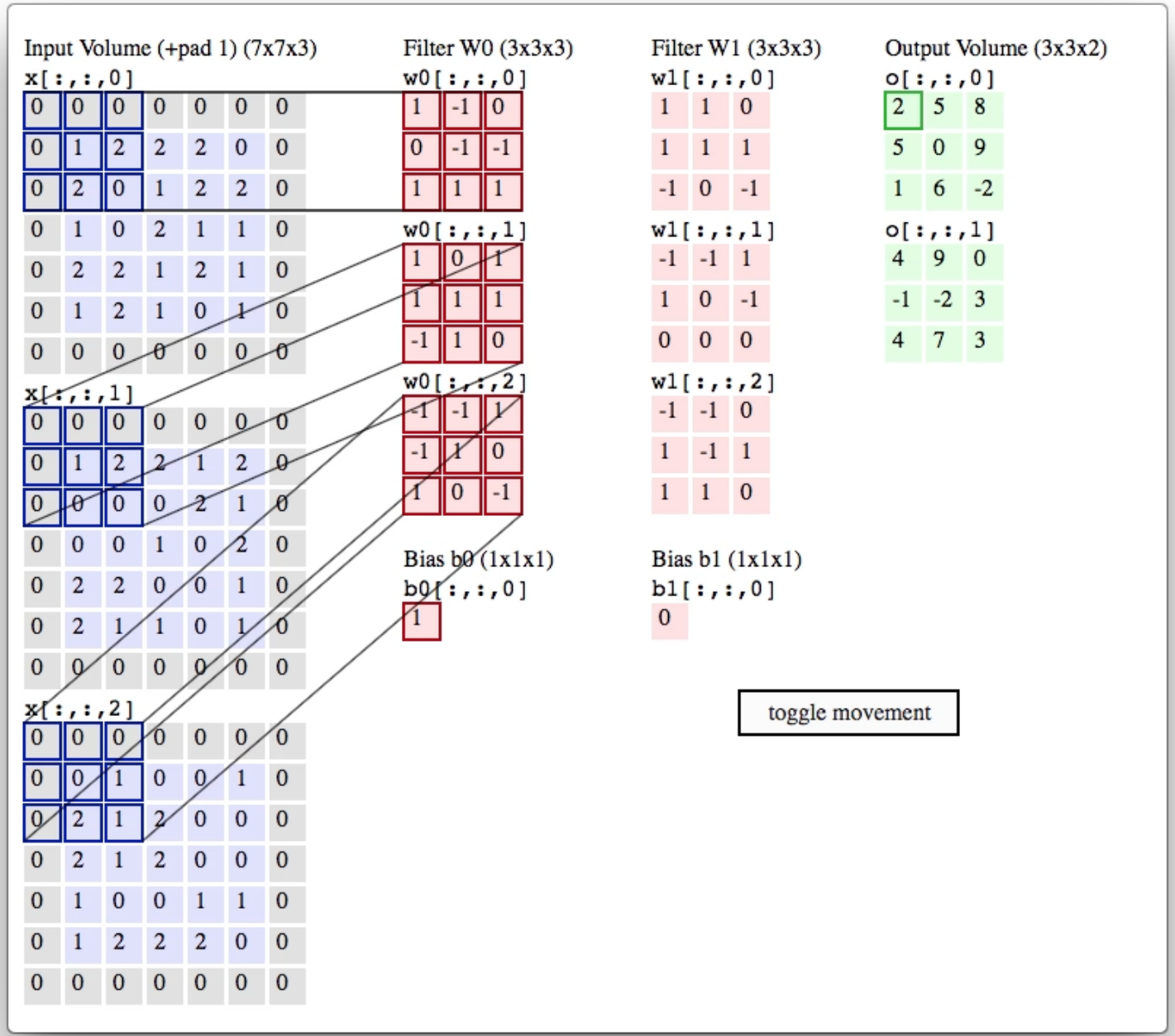
Number of Neurons in the Output Volume

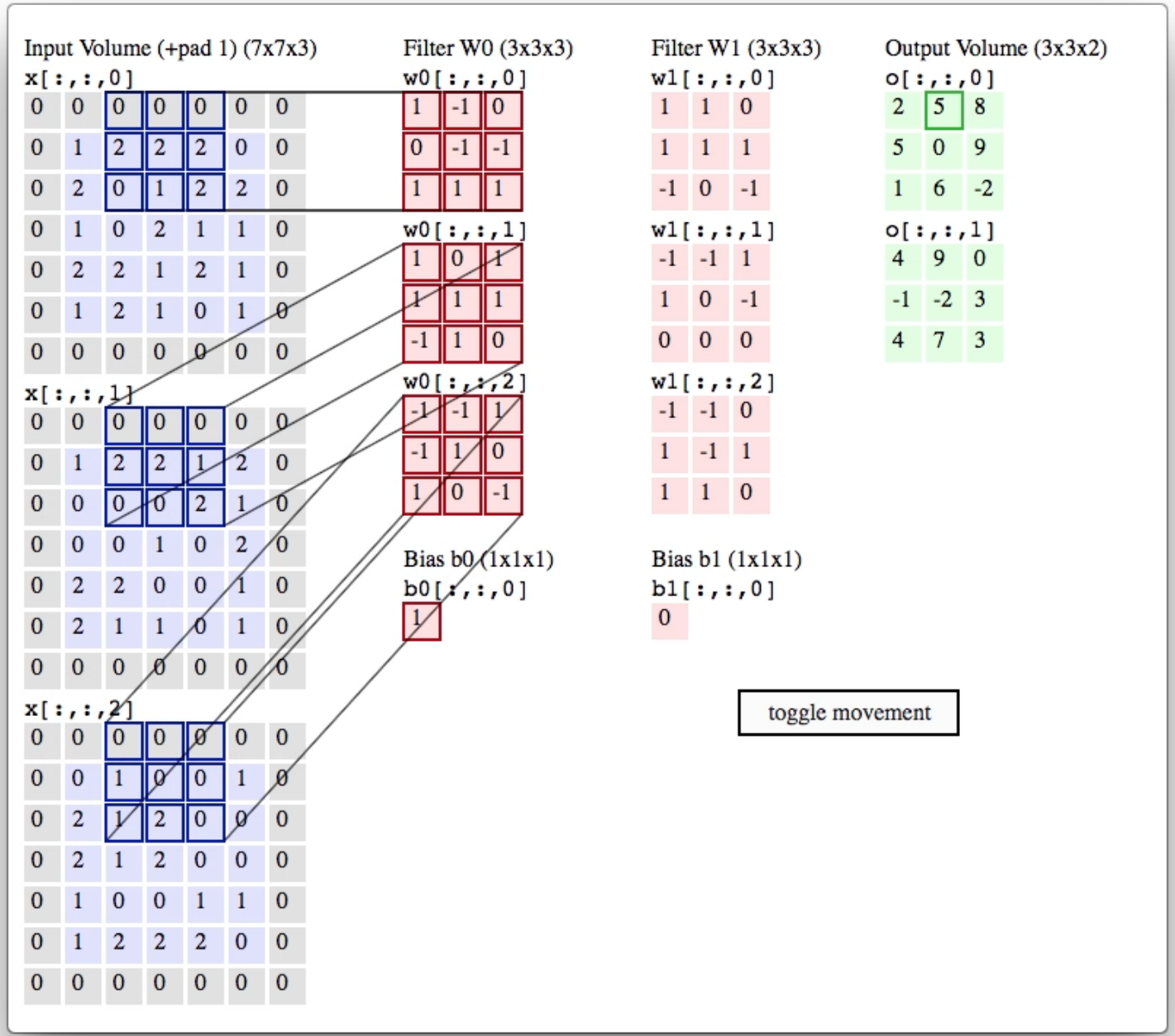
1. **Depth (hyper-parameter):** Number of filters: Number of patterns to look for in input. Think back to steerable filters. The set of neurons looking at the same region may be referred to as **depth column** or **fiber**.

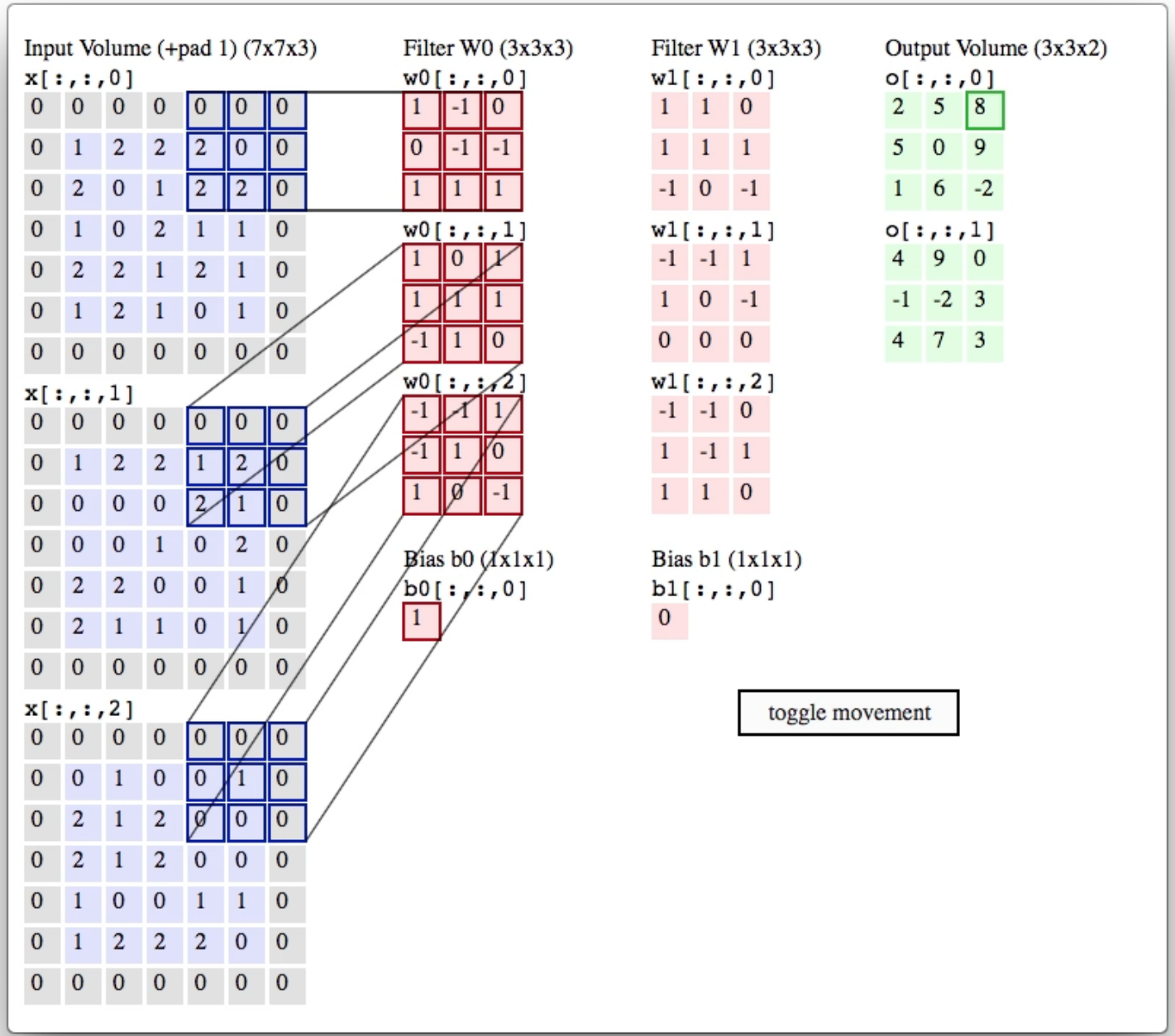
2. **Stride (hyper-parameter):** Number of pixels used when sliding the filter. This controls the overlap between neurons. If stride is 1, we move one pixel at a time (common, but depends on the width of filter).

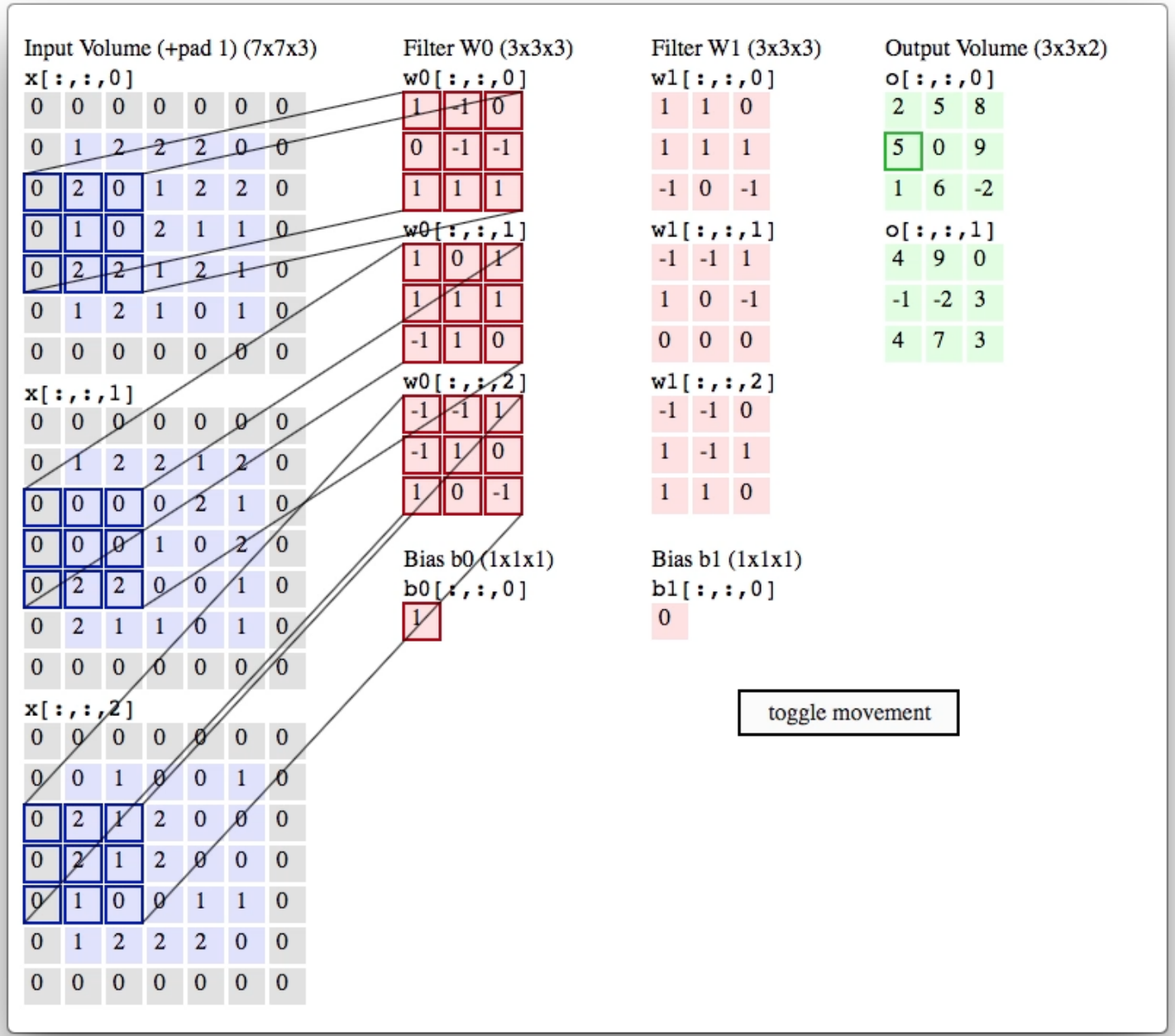
3. **Padding (hyper-parameter):** Number of zeros to add to the border of the volume. Convenient for convolution.

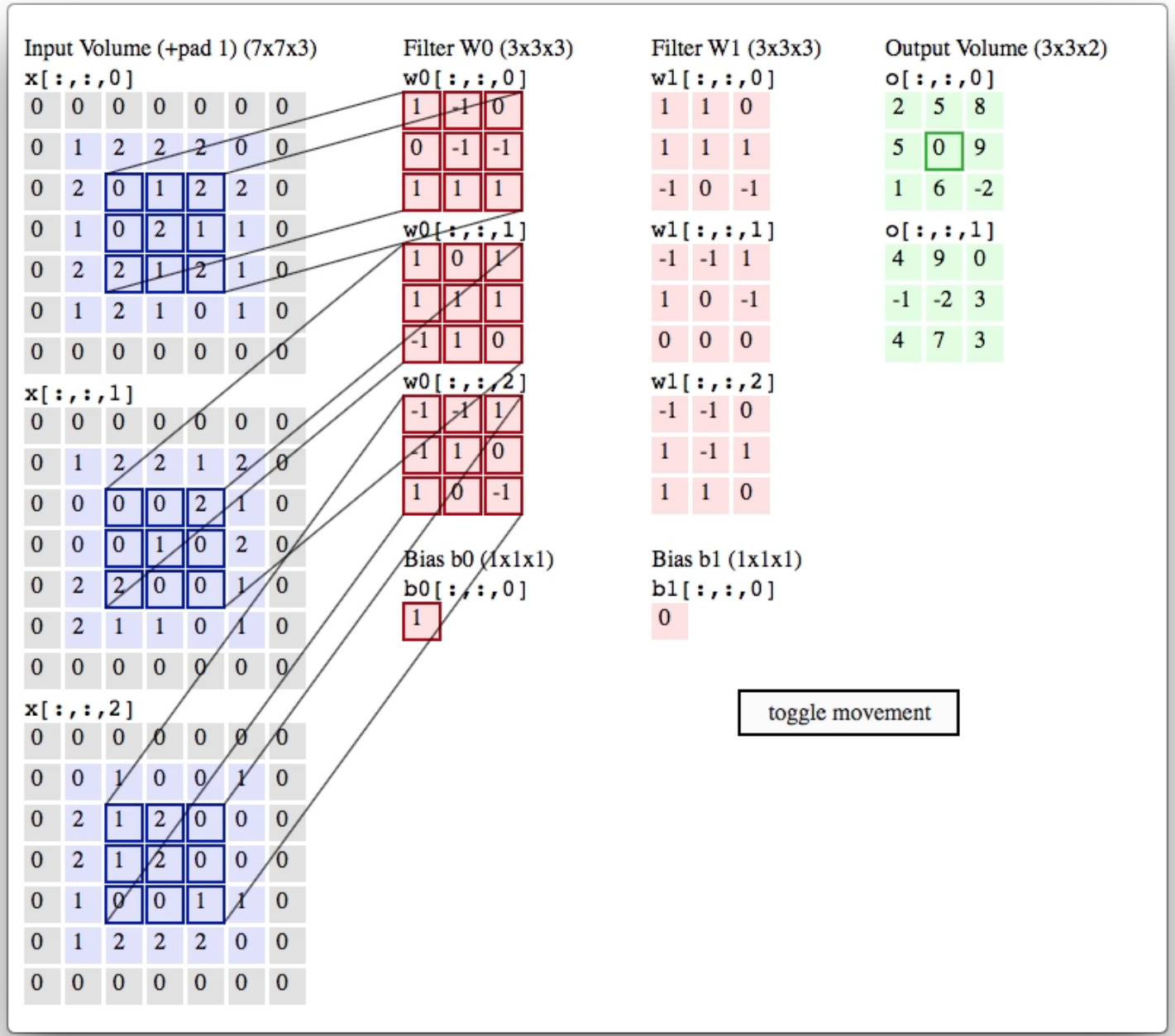


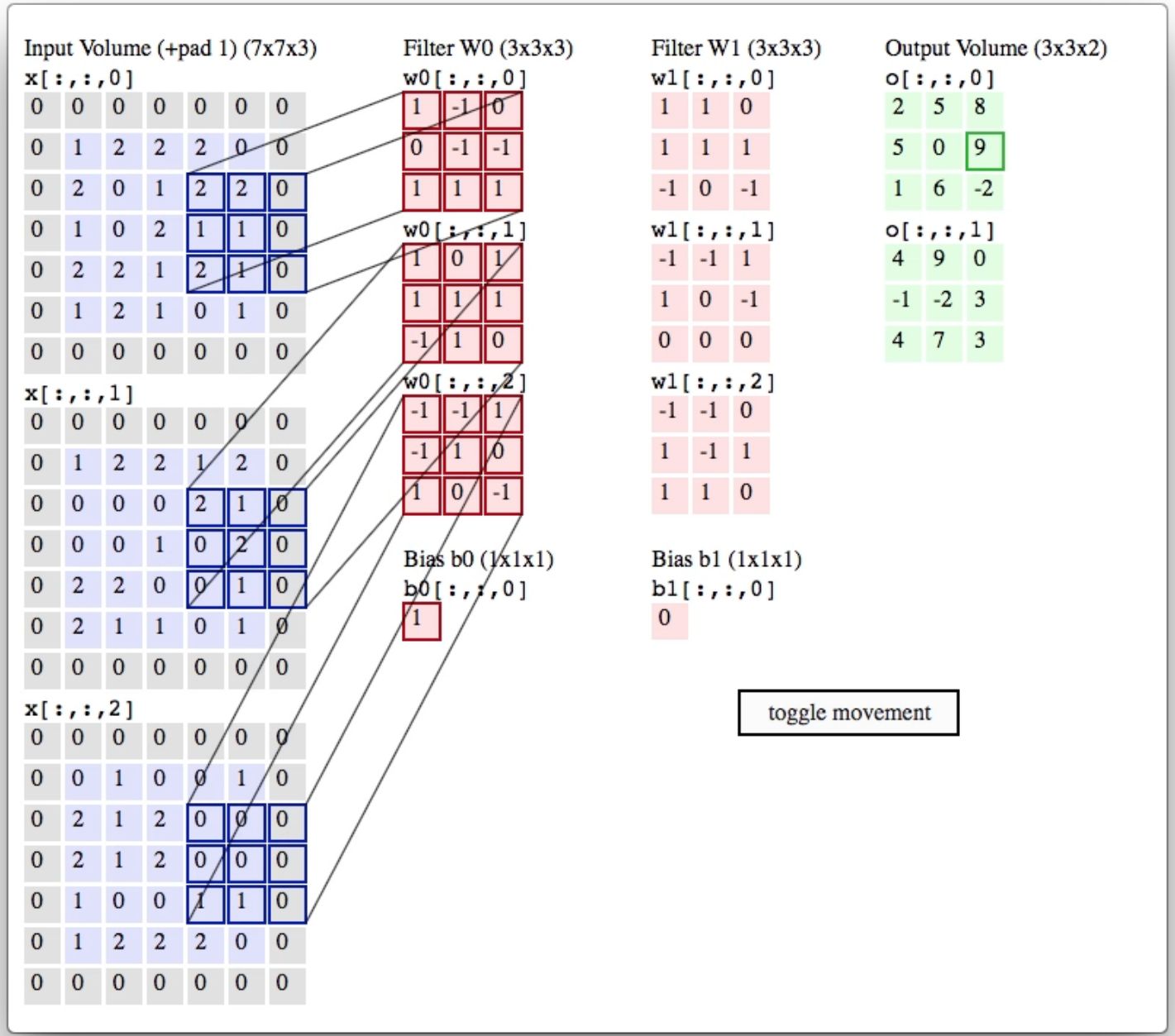

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

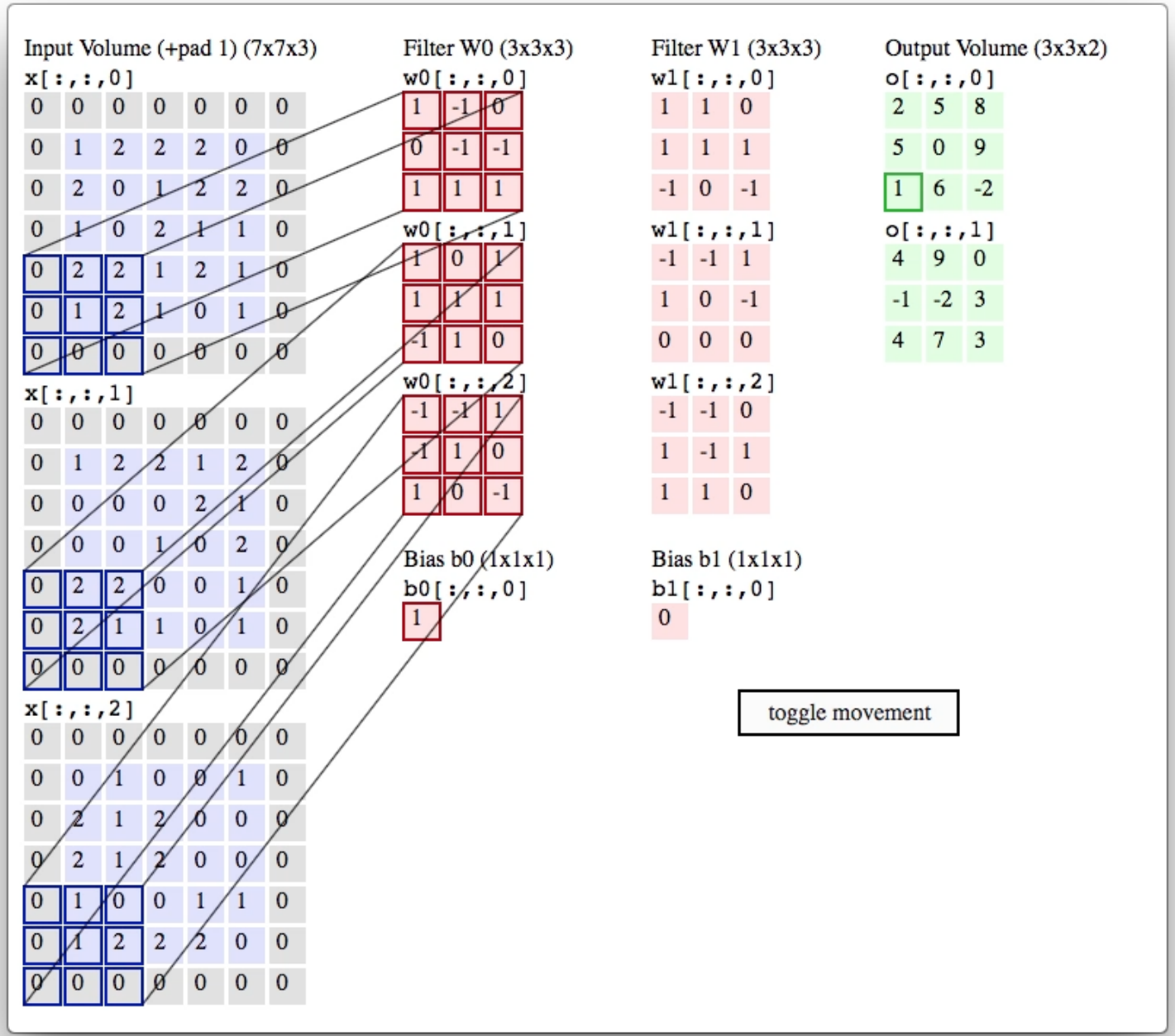

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

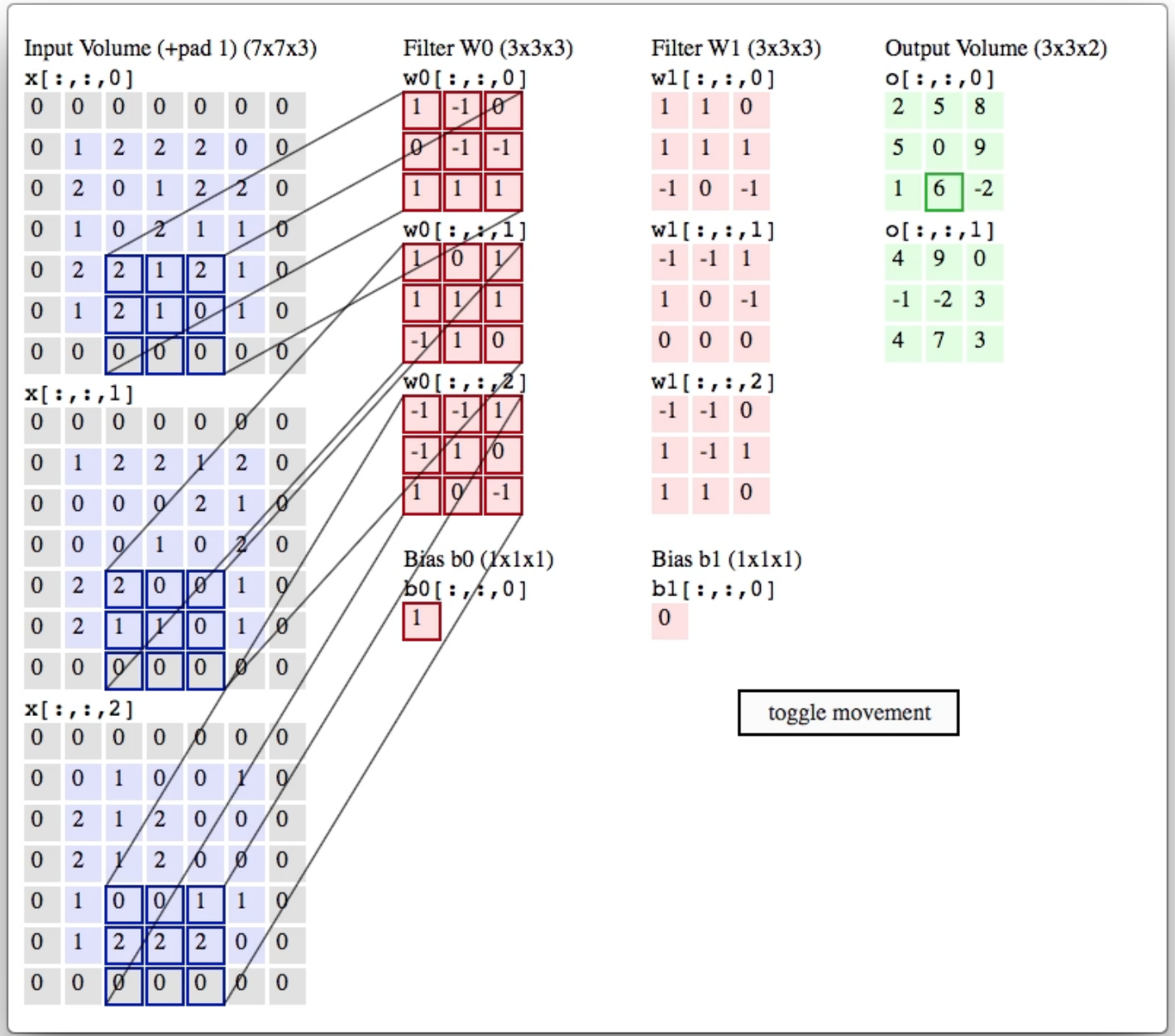

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

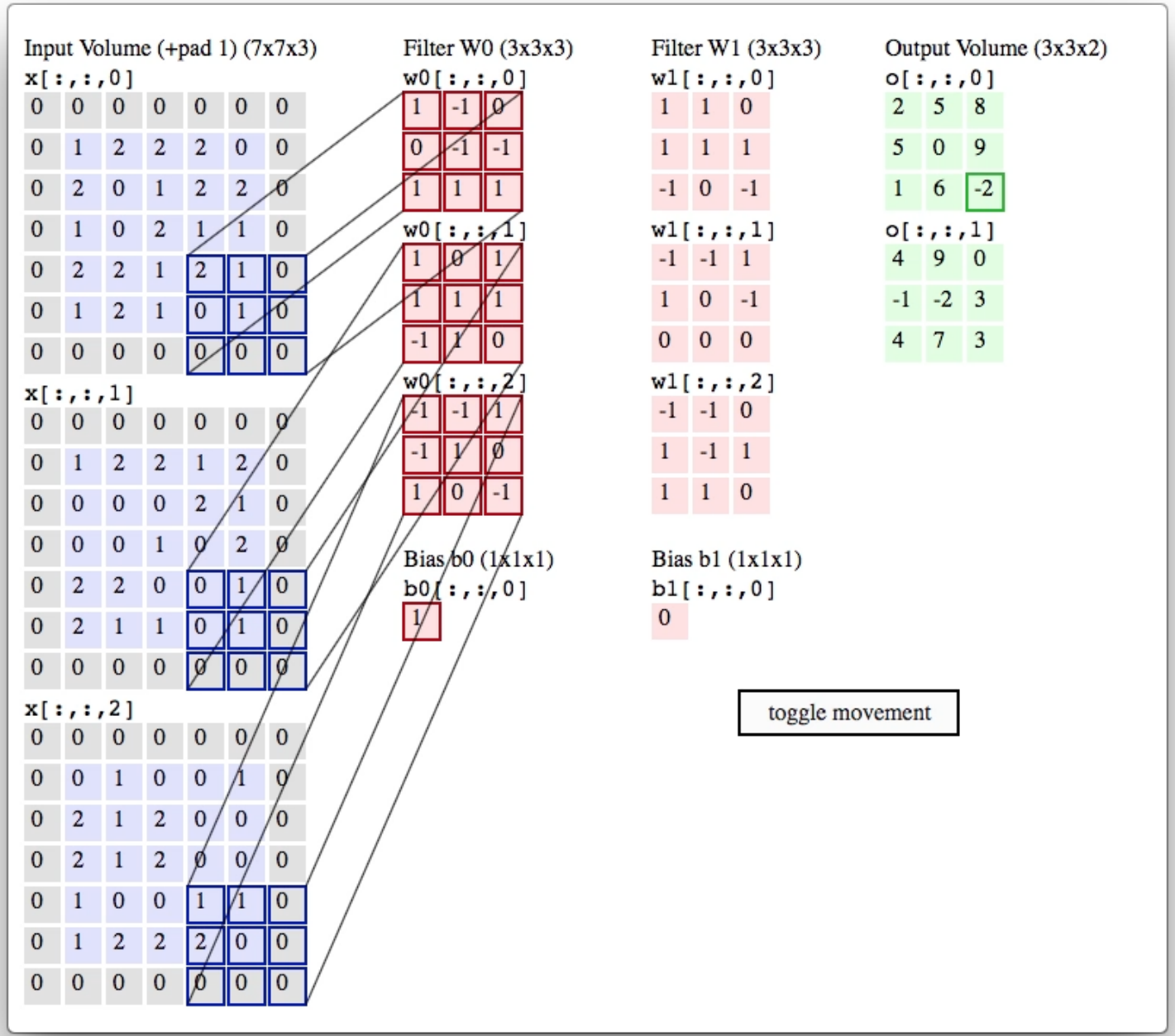

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

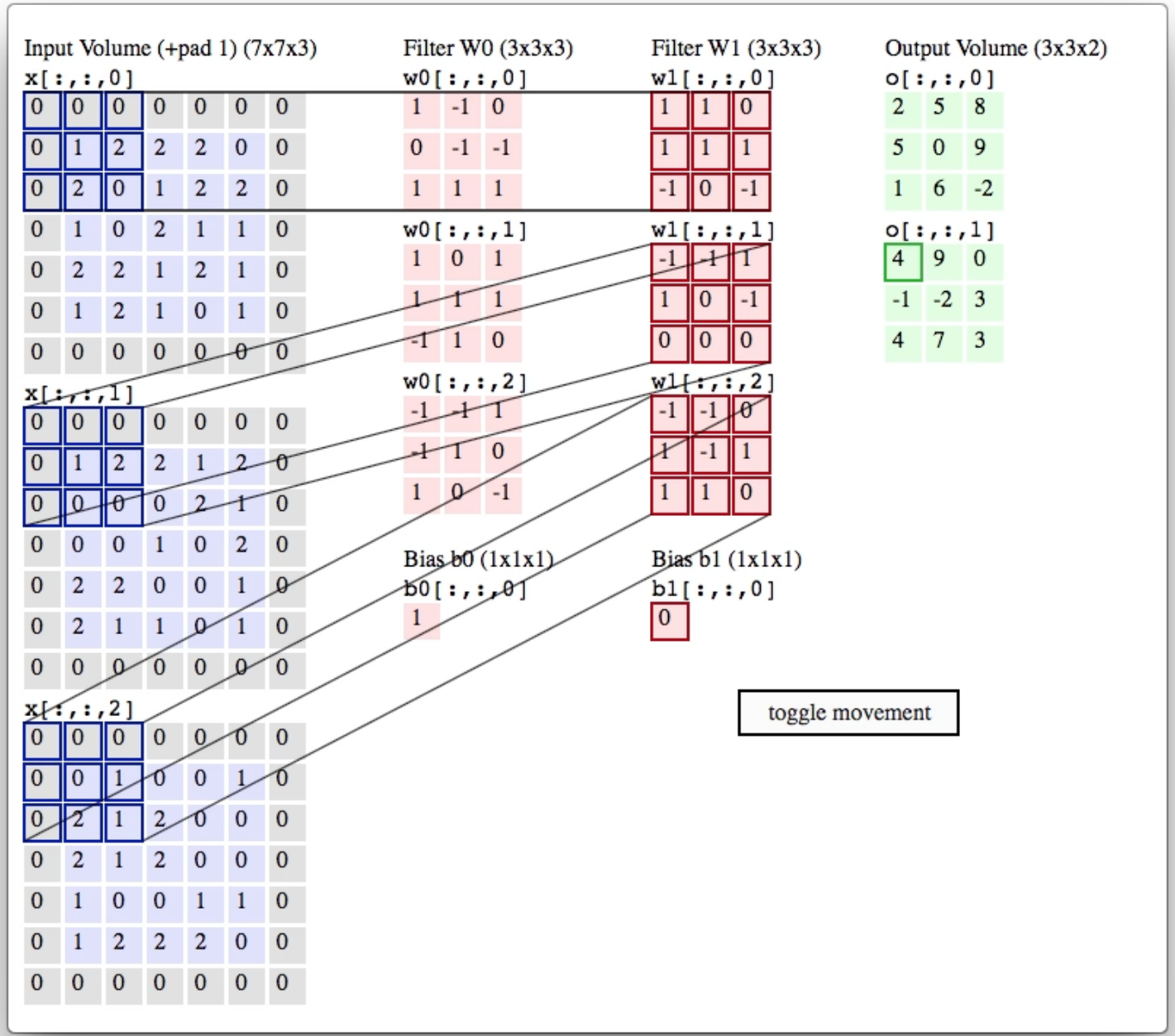

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

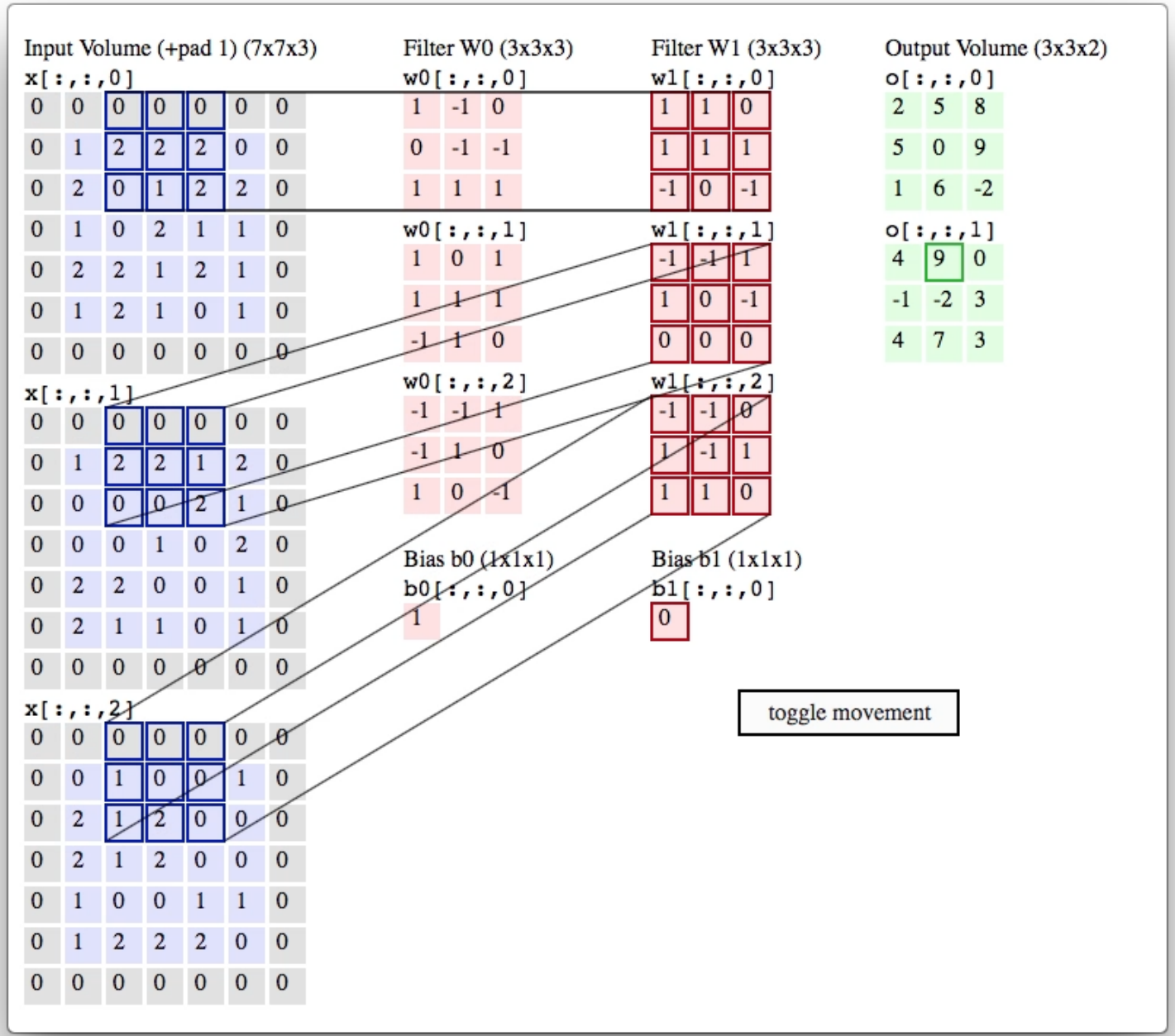

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

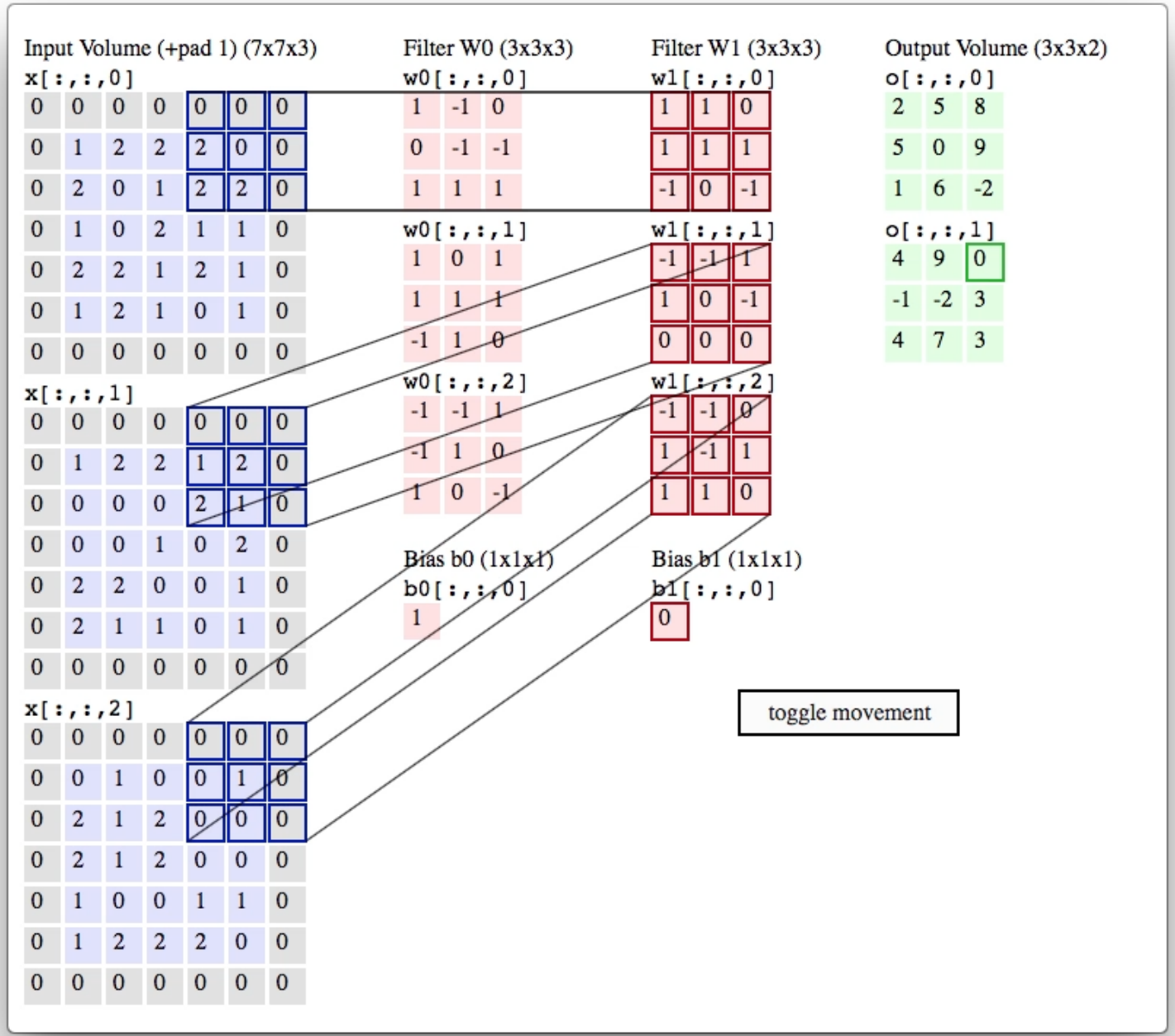

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

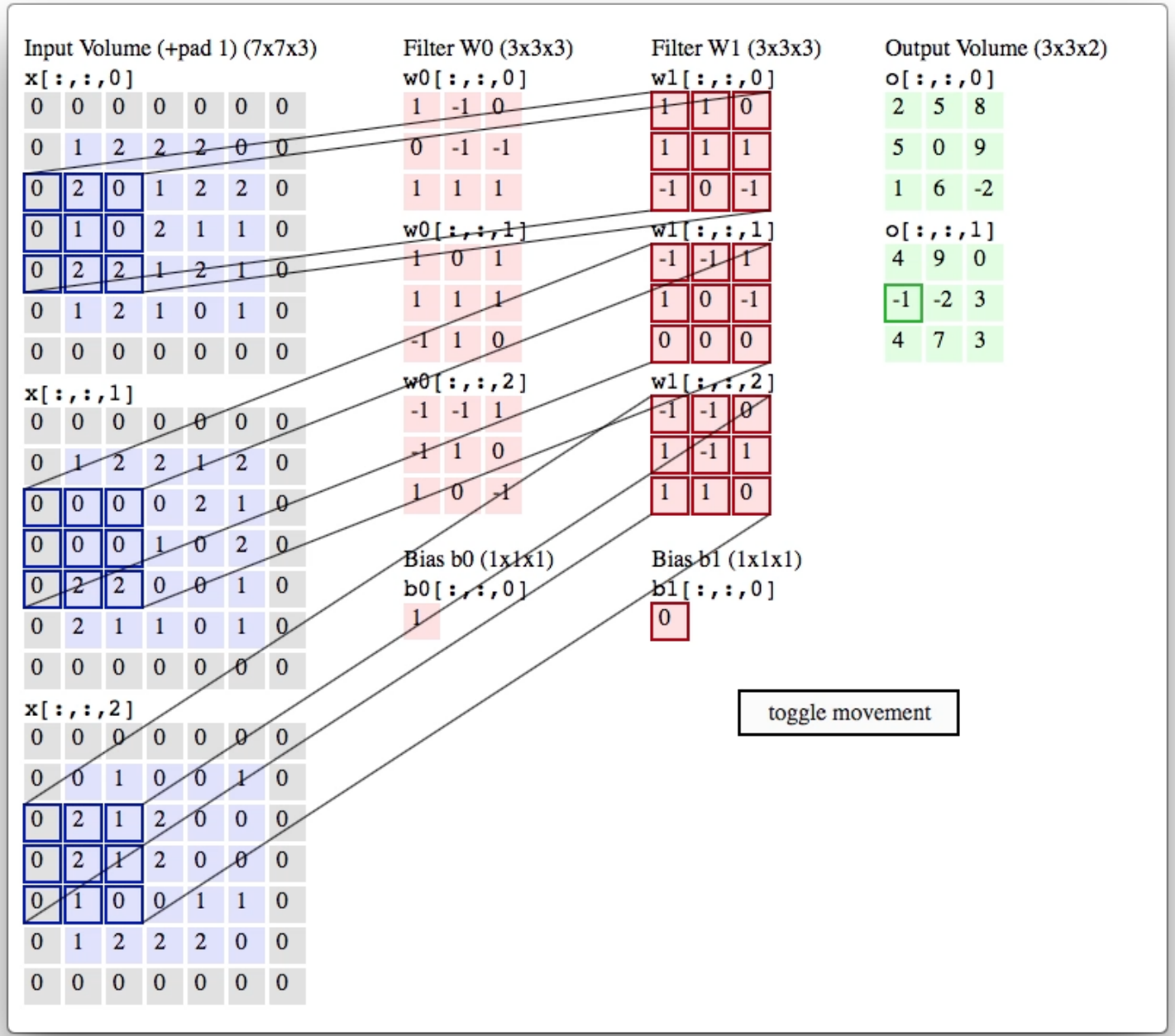

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

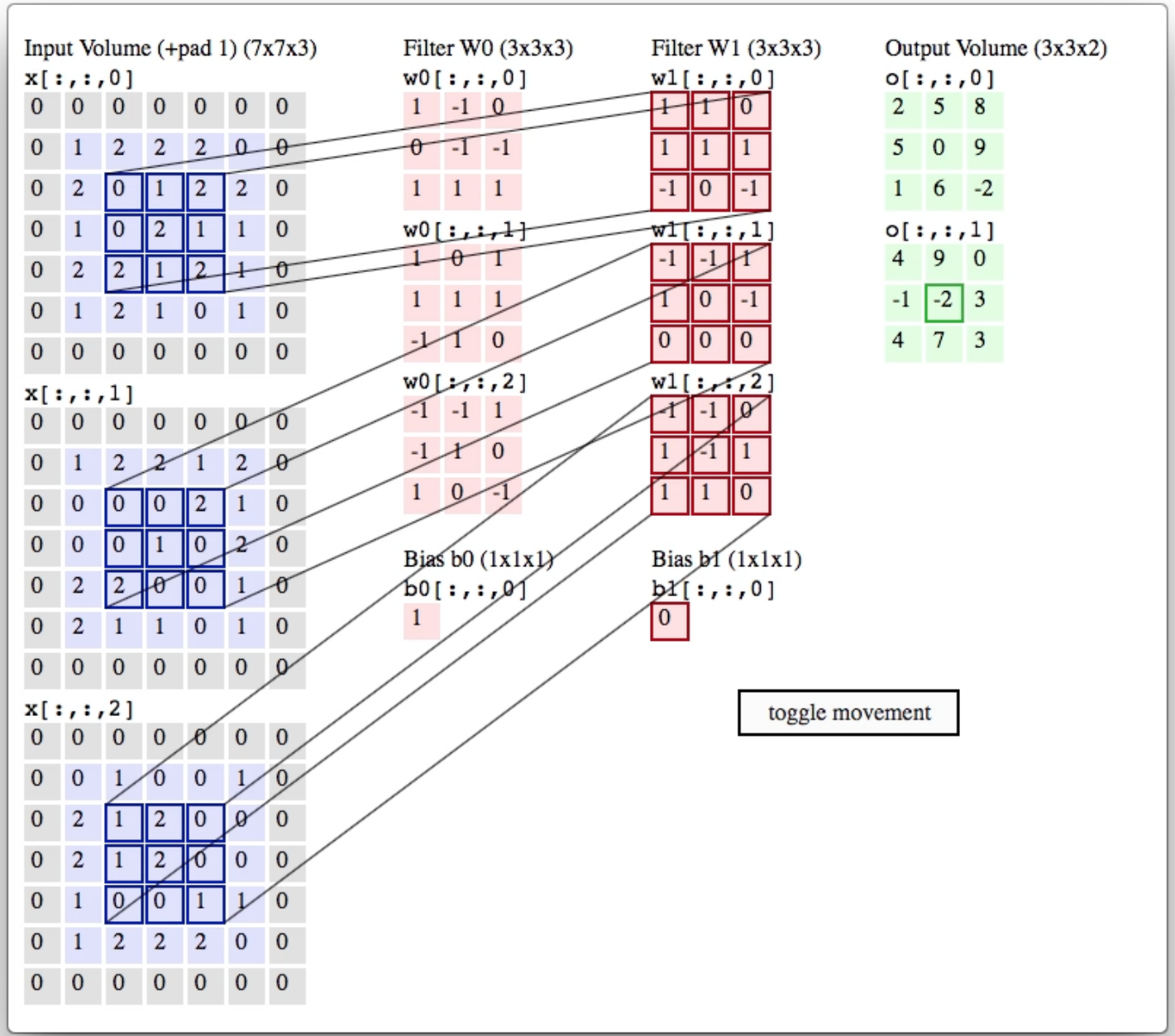

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

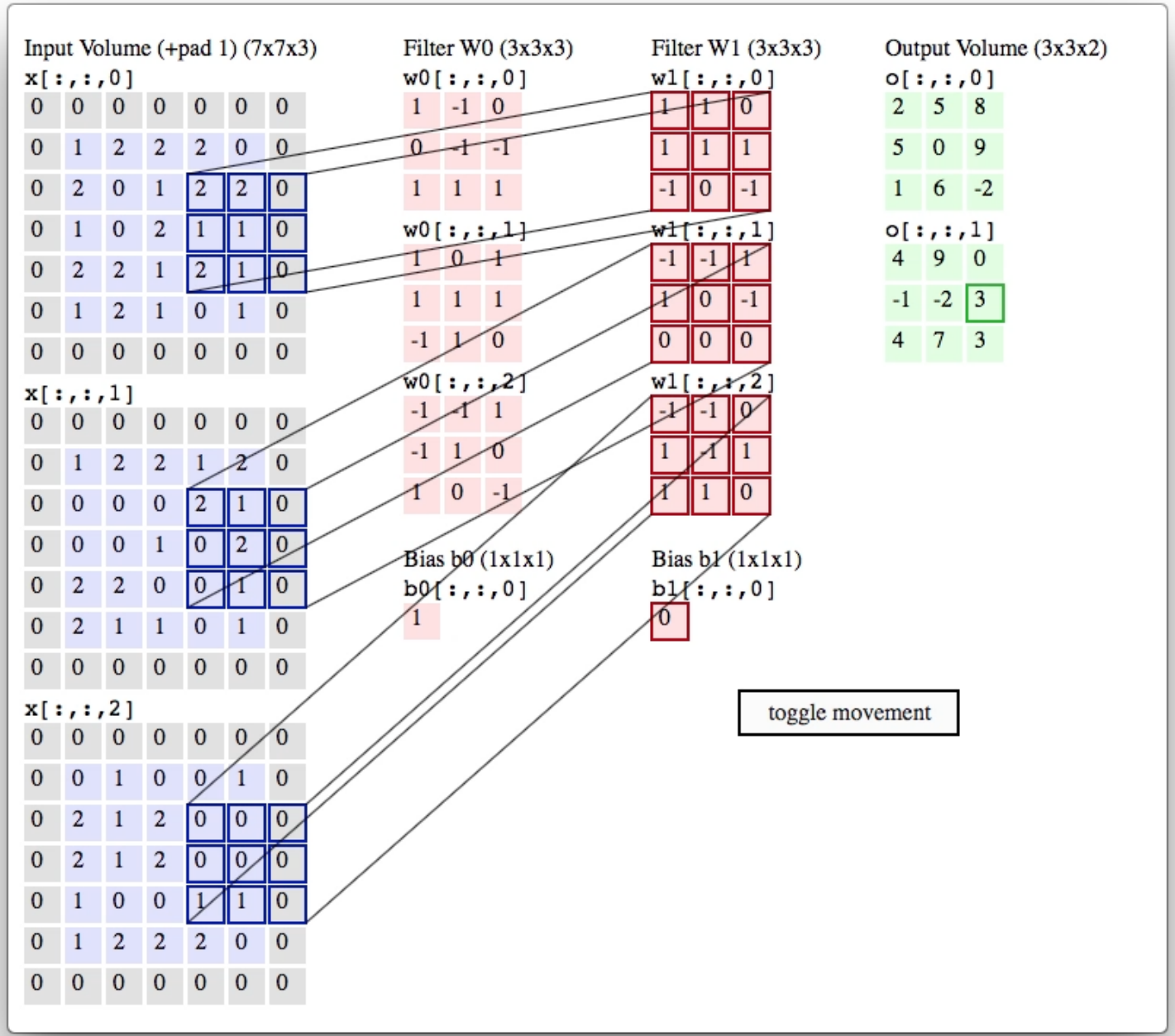

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

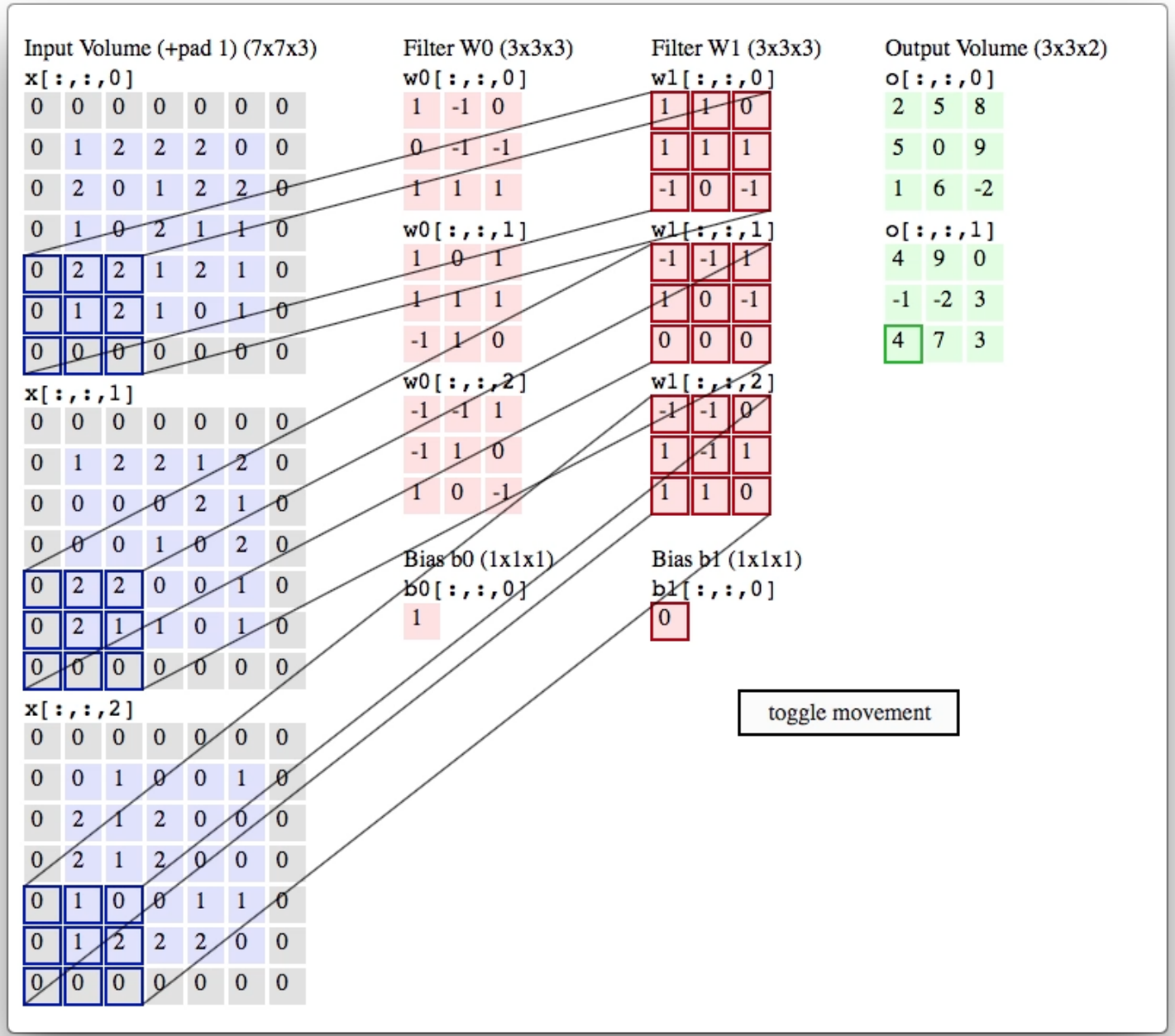

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

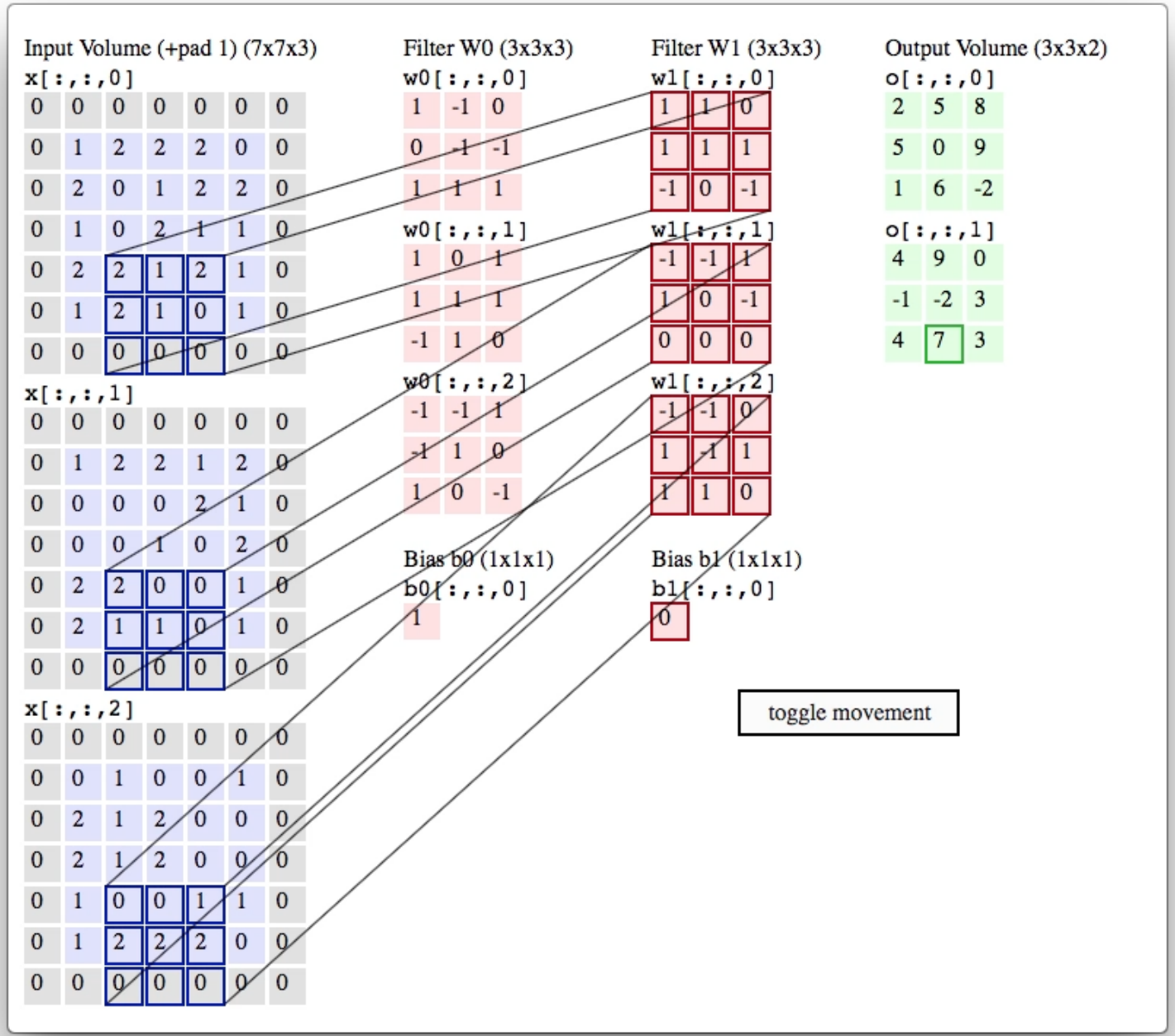

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

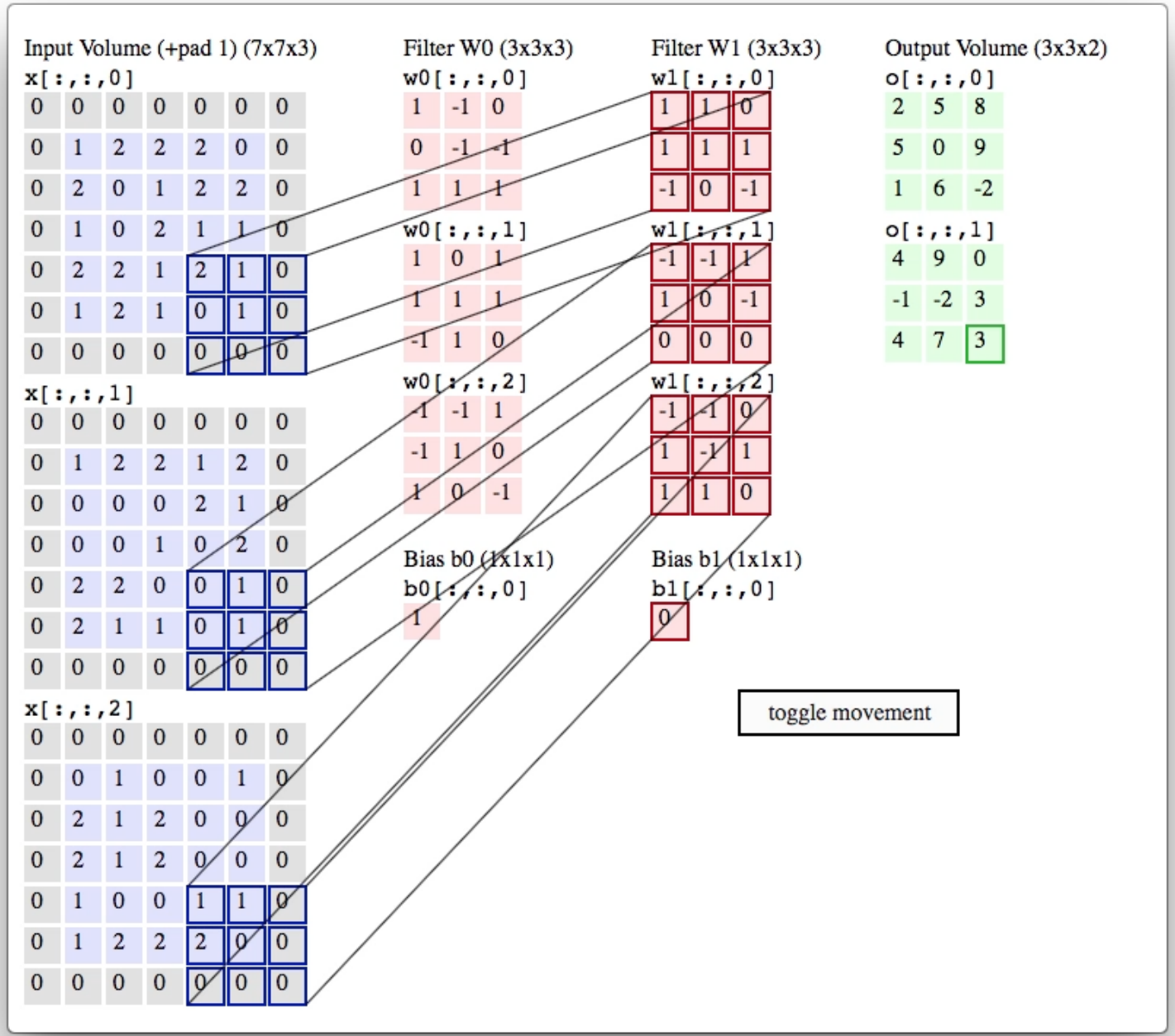

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 2 2 0 0
0 2 0 1 2 2 0
0 1 0 2 1 1 0
0 2 2 1 2 1 0
0 1 2 1 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 2 2 1 2 0
0 0 0 0 2 1 0
0 0 0 1 0 2 0
0 2 2 0 0 1 0
0 2 1 1 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 0 0 1 0
0 2 1 2 0 0 0
0 2 1 2 0 0 0
0 1 0 0 1 1 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 0
0 -1 -1
1 1 1
w0[:,:,1]
1 0 1
1 1 1
-1 1 0
w0[:,:,2]
-1 -1 1
-1 1 0
1 0 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 0
1 1 1
-1 0 -1
w1[:,:,1]
-1 -1 1
1 0 -1
0 0 0
w1[:,:,2]
-1 -1 0
1 -1 1
1 1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 5 8
5 0 9
1 6 -2
o[:,:,1]
4 9 0
-1 -2 3
4 7 3
toggle movement

Mutual Constraints of Hyper-Parameters

1. If stride is one, $S = 1$, it's common to set padding, $P = \frac{F-1}{2}$, to ensure that input and output volumes have the same size.

2. The number of neurons that fit is $\frac{W-F+2P}{S} + 1$. Must set hyper-parameters so there number of neurons is an integer!

3. **Real-World Example:** (Winner of ImageNet 2012) Images $227 \times 227 \times 3$, $F = 11$, $S = 4$, $P = 0$, $K = 96$.

   - Number of neurons in *first layer*

     $\frac{227-11}{4} + 1 = 55$ (per width)

     $55 \times 55 \times 96 = 290400$ (width $\times$ height $\times$ depth)

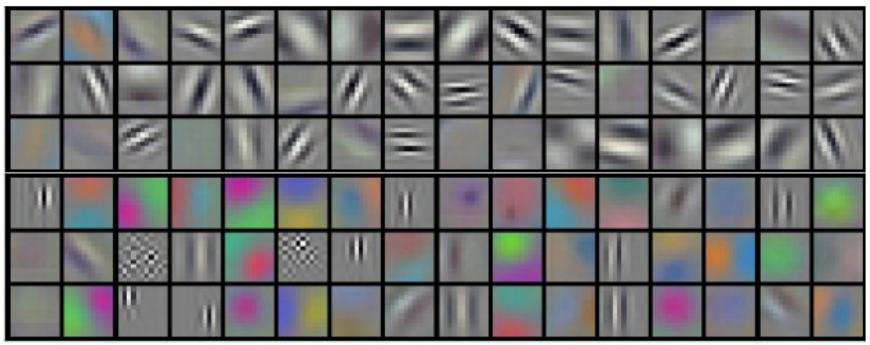
     Each neuron has $11 \times 11 \times 3 = 363$ weights plus 1 bias

     $290400 \times 364 = 105705600$ parameters *in the first layer!*

Parameter Sharing

If one feature (filter/kernel/depth slice) is useful at pixel (x_1, y_1), then it is also useful at pixel (x_2, y_2)

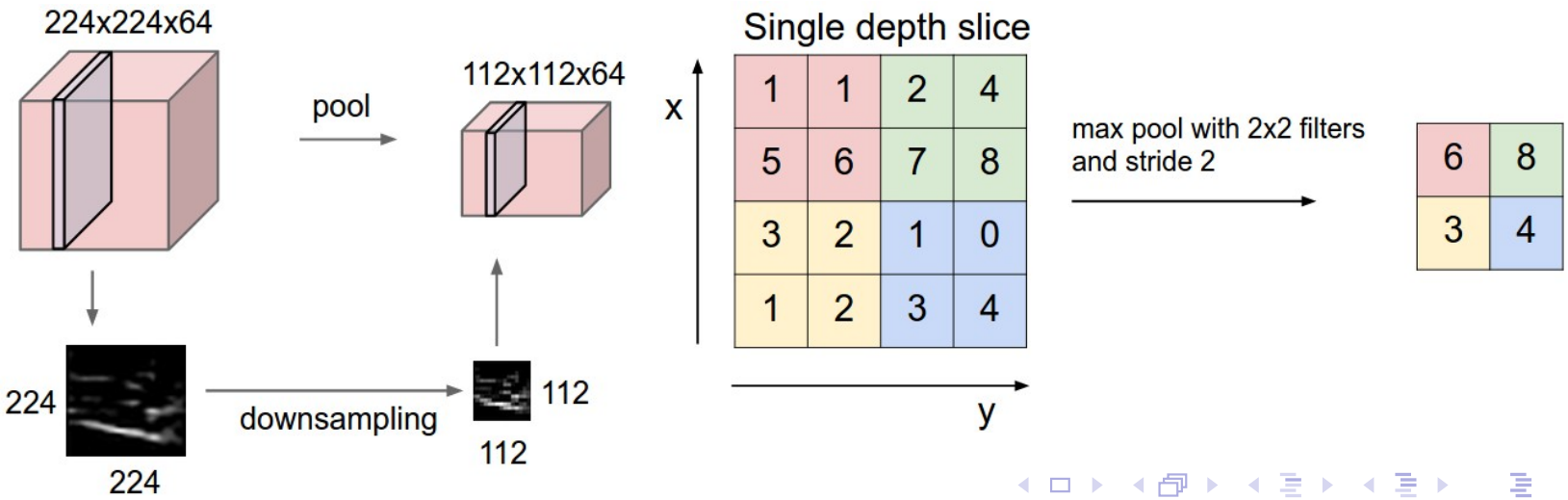
1. All neurons in the same depth slice have same weights and bias.
2. Example before $96 \times 11 \times 11 \times 3 = 34848$ unique weights $+96$ biases $= 34944$ parameters
3. Then forward pass can be seen as the convolution of that slice neuron's weights with the input volume.
4. Weights $=$ filter/kernel that is convolved with the input



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size $11 \times 11 \times 3$

Pooling Layer

1. Inserted in-between convolutional layers
2. Goal: To reduce spatial size, amount of parameters, computation requirements, and overfitting
3. Operates independently on every depth slice and resizes spatially using $max(\cdot)$ (or could be $mean(\cdot)$ or $L2$)
4. Introduces no parameters, since it is a fixed operation
5. Hyper-parameters: F (spatial size) and S (stride)



Normalization Layer

Future - Hope of Getting Rid of Pooling

1. Striving for Simplicity: Newer works propose discarding pooling. Suggest using larger strides
2. Variational auto-encoders
3. Generatie Adversarial Networks

Normalization Layer

1. Have show no real advantages

Practice

Hyper-Parameters and Layer Sizing

1. Input layer should be divisible by 2 many times
2. Conv layer use small filters (3×3), (5×5), (notice odd) and stride $S = 1$ (reduce sizing headaches - downsampling done by pooling)
3. Use padding that doesn't alter dimension
4. Pool layers (2×2), $F = 2$, and $S = 2$. Discards 75% of activations
5. Compromise based on memory constraints (GPU). May have to sacrifice at the 1st layer.

Popular Networks

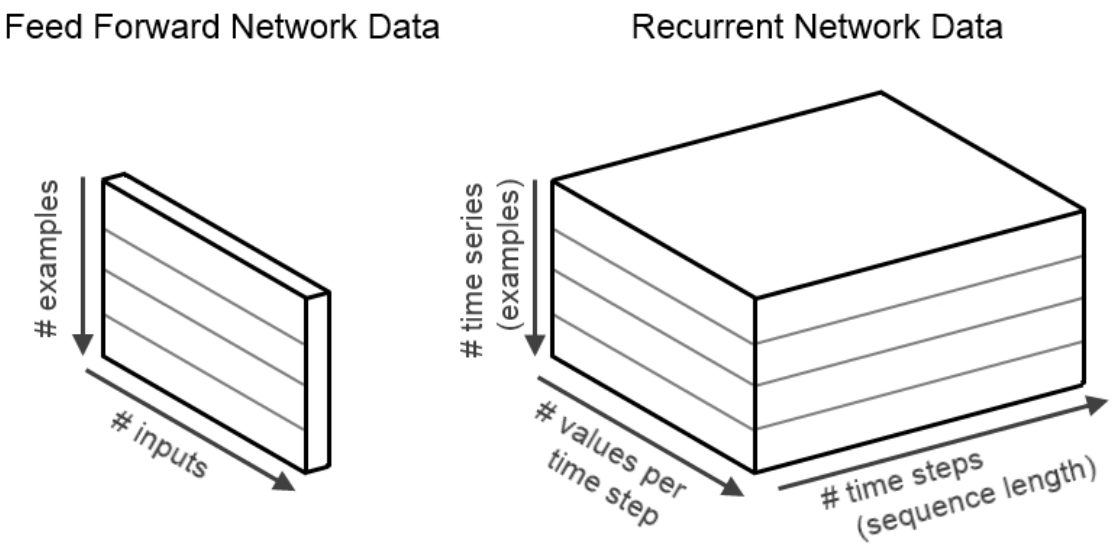
1. LeNet: LeCun 90's - read zip code (first successful ConvNet)
2. AlexNet: Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Popularize ConvNets for computer vision. ImageNet. Deeper, bigger than LeNet
3. ZF Net: Matthew Zeiler and Rob Fergus. ILSVRC 2013 Winner. Improved by expanding the size of the middle convolutional layers making the stride and filter size on the first layer smaller
4. GoogLeNet: Szegedy et al. from Google. ILSVRC 2014 winner. Reduced the number of parameters in the network from 60M to 4M (inception model)
5. ResNet: Residual Network. Kaiming He et al. ILSVRC 2015 winner. It features special skip connections and heavy use of batch normalization. State of the art as of 2016

Recurrent Neural Networks (RNNs)

RNNs are a type of neural network that share parameters across time. They are used frequently for natural language processing (NLP) and time series data.

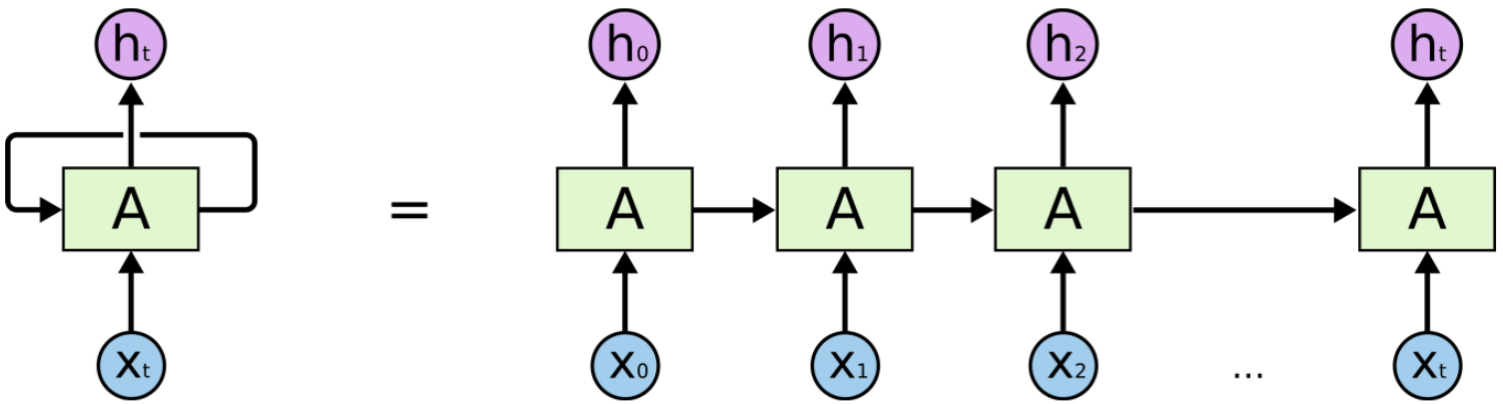
RNNs

1. Unlike standard neural networks or CNNs, RNNs introduce a temporal or sequential component to the system.
2. While CNNs introduce idea of spatial dependence, RNNs rely on the notion that *the current input relies on previous inputs*



RNN Architecture and Notation

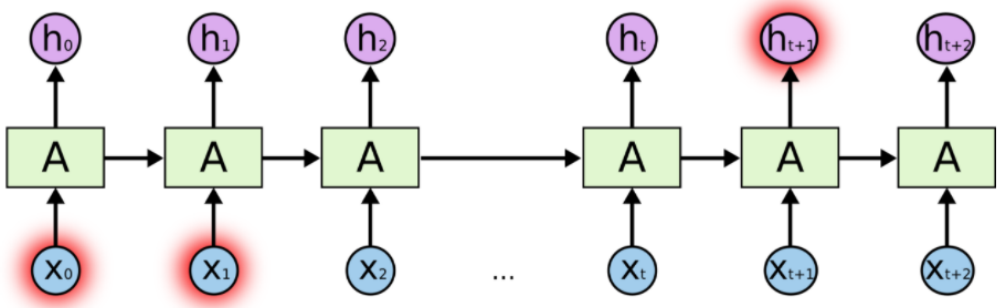
1. Current input: x_t
2. Current hidden state: h_t
3. Non-linear activation function σ or $tanh$
4. Predicted output: $\widehat{y_t}$



Long-Term Dependencies

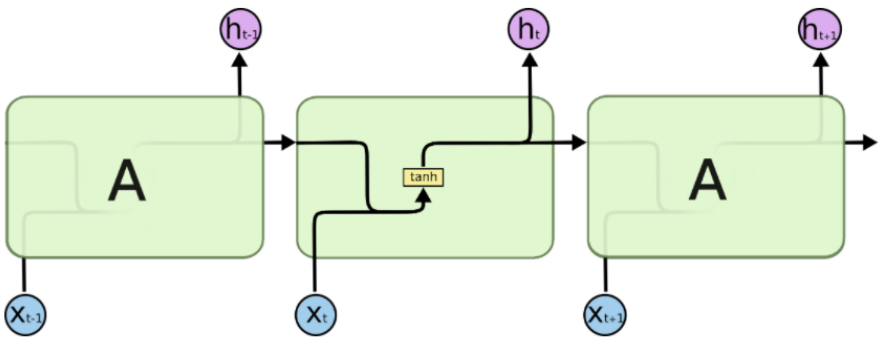
1. Suppose we want a model that predicts the next word in this sentence:
   - "*The clouds are in the* <WORD>"

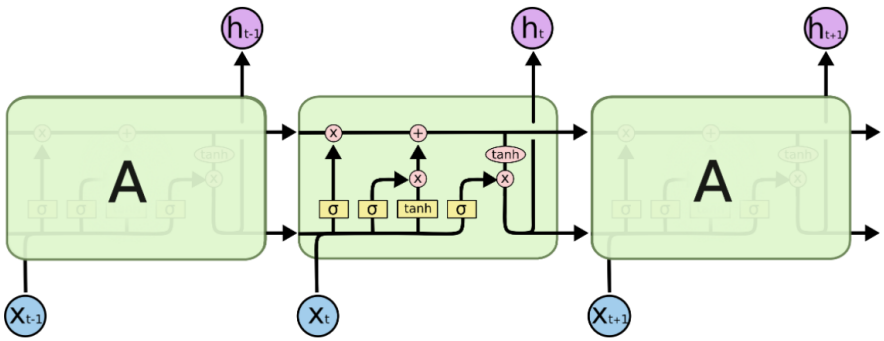
2. Now suppose we to model the next word in this sentence:
   - "*I grew up in France and lived there most of my life. As a result I am a native speaker of* <WORD>"



Long Short Term Memory Networks (LSTM)

LSTMs are a species of RNN that is designed for learning long-term dependencies. While standard RNNs are able to model temporal dependencies, LSTMs have a much more elaborate structure.

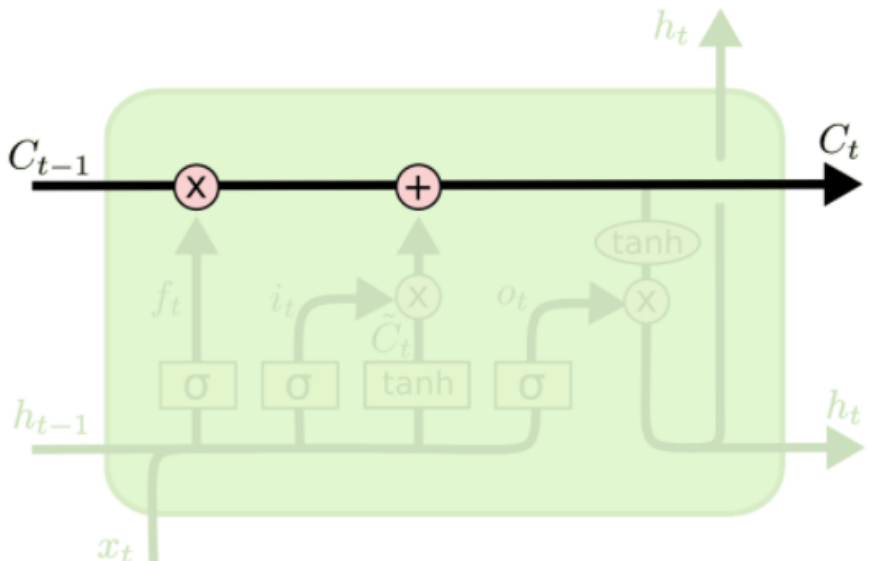




Cell State

The key idea of LSTMs is that of cell state to propagate information.

1. Cell state is subject to linear operations
2. Information in cell state can be "forgotten" using a system of gates

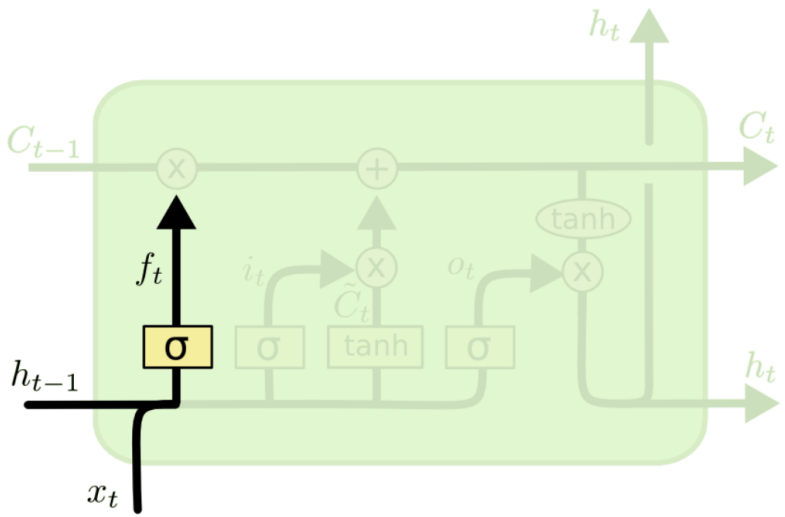


LSTM Forget Gate Layer

Forget gate layer is a sigmoid layer that decides the values to be updated with

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

1. This layer looks at h_{t-1} and x_t and updates the cell state
2. f_t has range $(0, 1)$
3. 0 indicates "complete get rid of this", and 1 indicates "completely keep this".

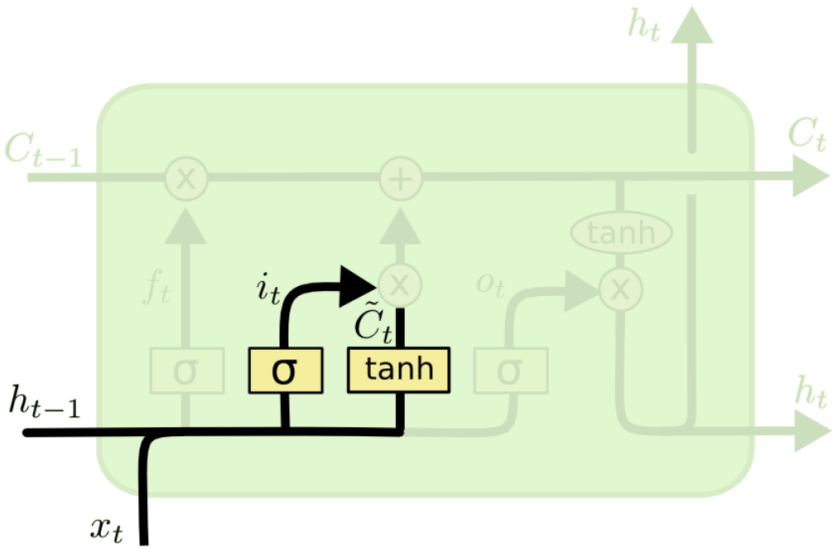


Input Gate Layer

Input gate layer is a combination of a sigmoid layer and $tanh$ layer that decide the new information to keep with

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\widetilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

1. Sigmoid layer is our "input gate layer"
2. $tanh$ layer creates candidate values

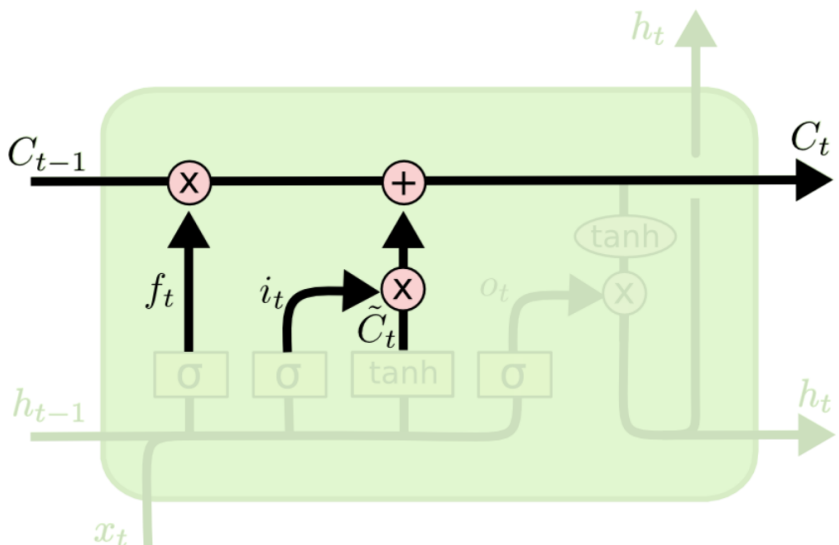


Update the Cell State

Now we update the cell state using

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C}_t$$

1. Multiply old state by f_t, forgetting what we had decided
2. And add $i_t \cdot \widetilde{C}_t$, our new candidate values



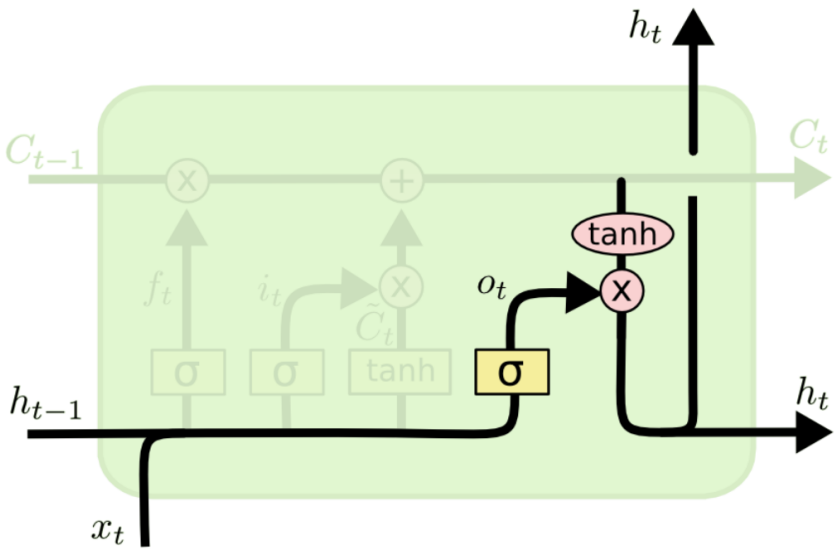
Output Layer

Now we decide what to output with

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

1. Take updated cell state to help compute output
2. Apply sigmoid layer and $tanh$ layer



Autoencoder Basics

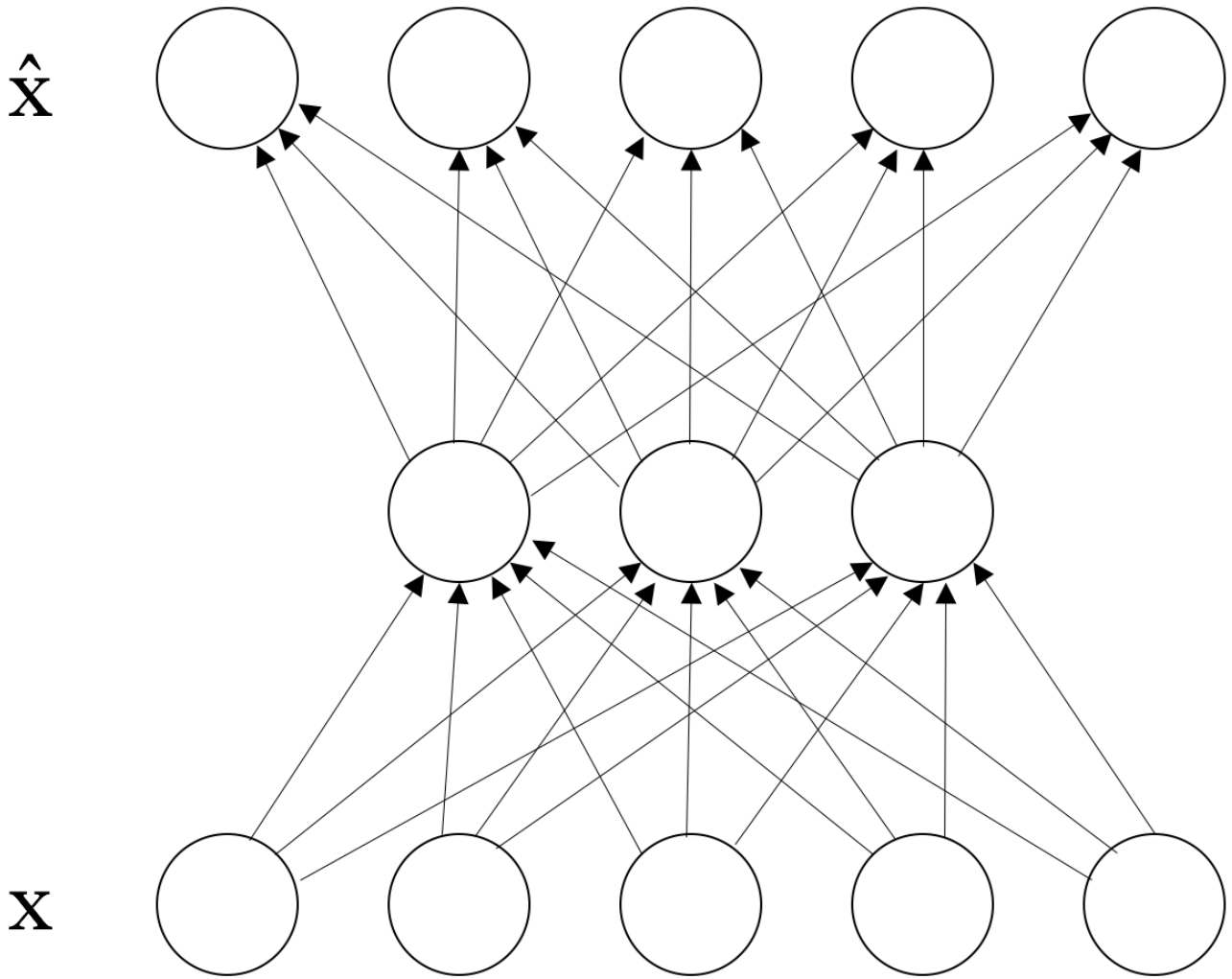
1. Feed-forward neural network
2. Used for unsupervised learning
3. Trained to reproduce input layer in the output layer
4. Training uses gradient descent

Autoencoder Basics cont.

1. Similar to PCA*, but more flexible
2. Autoencoders can accommodate non-linear transformations
   1. Consequence of flexible activation functions
   2. Turns out to be hugely useful

* Actually, with squared error loss and no sigmoid transformation, it is equivalent to PCA (Baldi & Hornik, 1989)

Simple Autoencoder



Simple Autoencoder cont.

From input layer to hidden layer is "encoder":

$$f(x) = \sigma(Wx + b)$$

Hidden layer to the output is the "decoder":

$$g(x) = \sigma(W'x + b')$$

where σ is the activation function (often sigmoid) and W' is often $W^\top$ (known as having "tied" weights).

Simple Autoencoder cont.

1. Input layer $\rightarrow$ hidden layer (fewer units) $\rightarrow$ output layer
2. Hidden layers values can be thought of as lossy compression of input layer
3. Hidden layer with fewer units than input is often called "bottleneck" or under-complete layer

Question

In a sense, autoencoders want to learn an approximation to the identity function.

Why would this be useful?

Answer

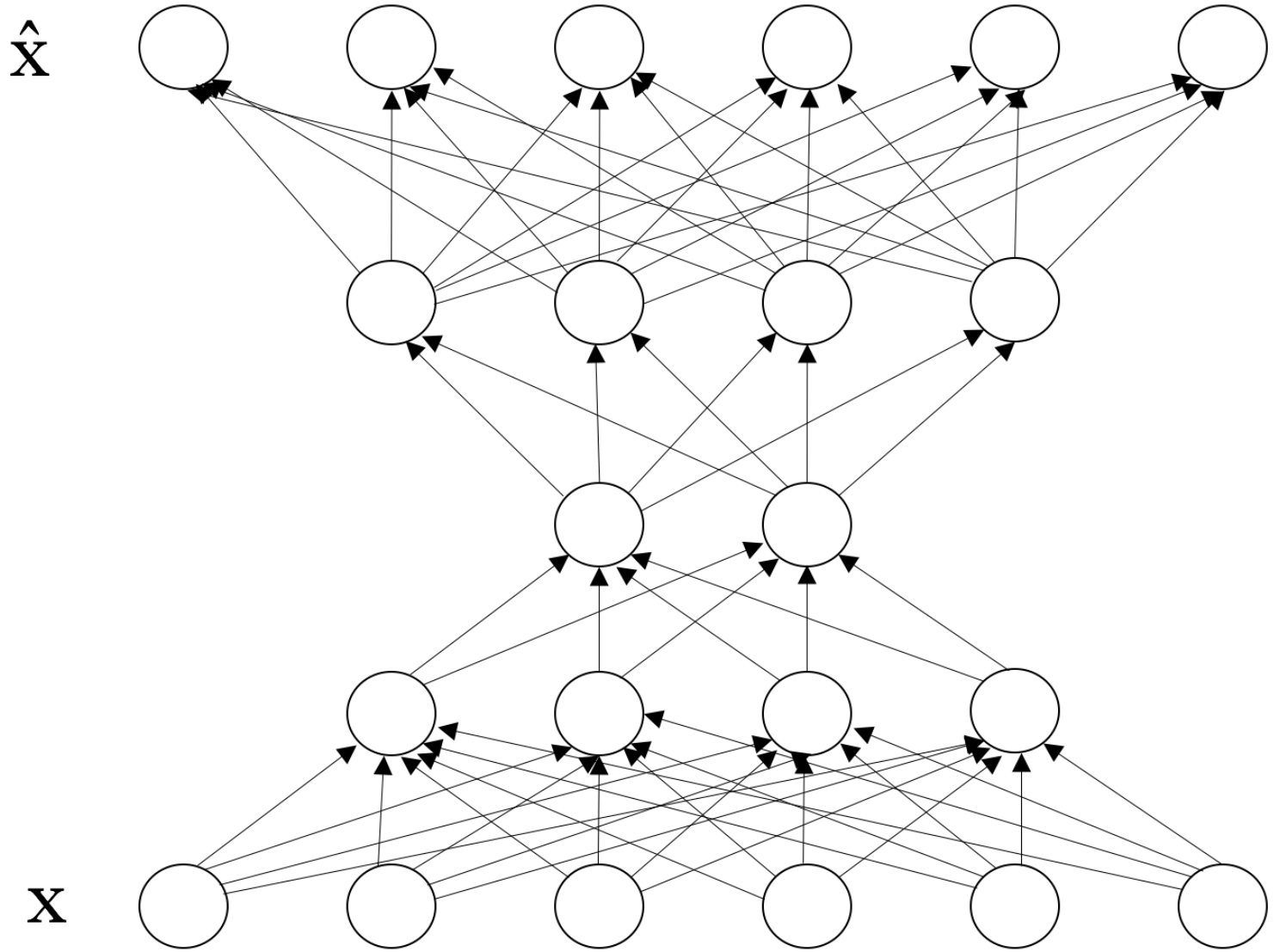
This turns out to have several practical uses.

1. Autoencoders learn a kind of latent representation of the input
2. Can be used for initializing weights and biases prior to fitting neural net
3. Very well suited to dimensionality reduction in NLP tasks

Stacked Autoencoder

1. Sometimes called deep autoencoder
2. Feed-forward neural network
3. Has additional layers
4. Still "unsupervised" in the sense of no labels or real-valued outcome variable
5. Output layer is still $\hat{x}$
6. Trained in stages

Stacked Autoencoder



Stacked Autoencoder cont.

1. Beneficial for data with hierarchical organization
2. Can be difficult to train using back propagation

Denoising Autoencoder

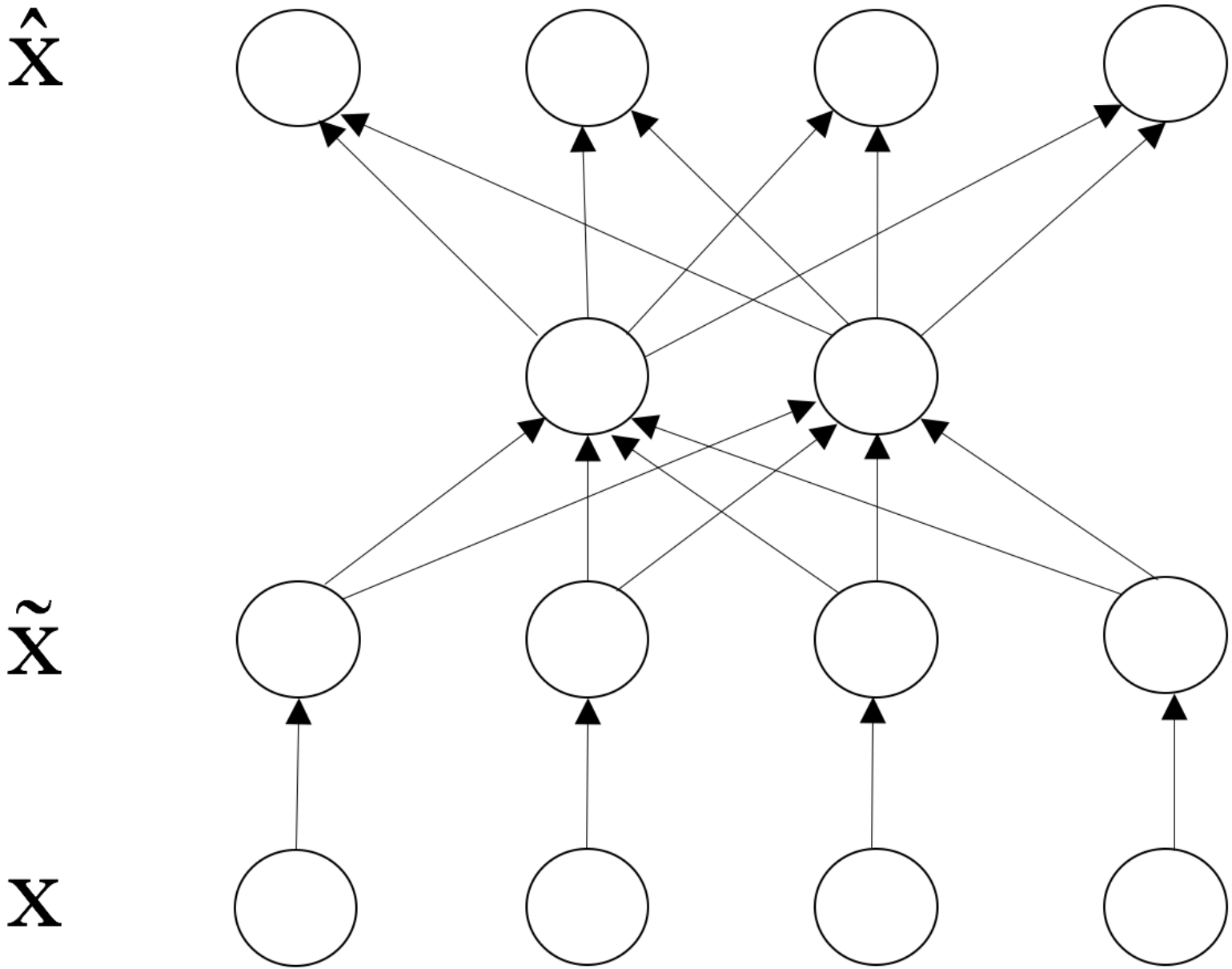
1. Stochastic version of autoencoder
2. Random noise injected into the input layer
   1. *Masking noise*: some fraction of elements* in x set to 0
   2. *Gaussian additive noise*: $\tilde{x}|x \sim \mathcal{N}(x, \sigma^2 I)$
   3. *Salt-and-pepper noise*: fraction of elements* in x set to either min or max possible value (often 0 or 1) according to coin toss

* chosen at random for each example

Denoising Autoencoder

1. Output layer is evaluated against original (uncorrupted) data
2. Goal is slightly different from simple autoencoder
   1. Not attempting to learn identity function
   2. We want robustness of representation
   3. Learned representation is insensitive to perturbations in the input

Denoising Autoencoder



Denoising Autoencoder cont.

1. Training proceeds fairly similarly to standard autoencoder
2. Crucial difference is the loss function is calculated using *uncorrupted* input layer

Extending Denoising Autoencoder

1. Some noise types only corrupt subset of the input's components
   1. Masking noise
   2. Salt-and-pepper noise
2. For these, we can extend denoising by adding emphasis on corrupted components.
3. Use hyperparameters α and β to control emphasis; for squared error loss, this yields

$$L(x, \hat{x}) = \alpha \left(\sum_{i \in \mathcal{I}(\tilde{x})} (x_i - \hat{x}_i) \right) + \beta \left(\sum_{i \notin \mathcal{I}(\tilde{x})} (x_i - \hat{x}_i) \right)$$

where $\mathcal{I}(\tilde{x})$ denotes indexes components of x that were corrupted.

When to use neural networks

Conditions under which you might consider using neural networks:

1. Have a huge amount of labeled training data
2. Image classification (with huge amount of labeled images)
3. Certain NLP tasks
4. Some signal processing problems

When *not* to use neural networks

Probably should **not** use neural networks when:

1. You have specific hypotheses you want to test
   - E.g., "*Drug* X *improves condition* Y".
2. Interested in estimating the effect of some variable(s) on some outcome variable
3. You have highly structured data and/or few features

In the case of (1.) and (2.), a traditional statistical model is better. In the case of (3.), using some ensemble-of-trees method will give as-good or better results with minimal tuning.

Sources

1 http://cs231n.github.io/convolutional-networks/
2 https://colah.github.io/posts/2015-08-Understanding-LSTMs/
3 Dave Berenbaum "Activation Functions" (2016) https://github.com/brown-data-science/deep_learning_group/blob/master/slides/activation_functions.p
4 Yinong Wang "Recurrent Neural Networks" (2016) https://github.com/brown-data-science/deep_learning_group/blob/master/slides/recurrent_neural_netwo
5 Vincent P., Larochelle, H., Bengio Y., Manzagol P.A. (2008) *Extracting and Composing Robust Features with Denoising Autoencoders*.
6 Vincent P., Larochelle, H., LaJoie I., Bengio Y., Manzagol P.A. (2010) *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*
7 Makhzani A., Frey B. (2015) *Winner-Take-All Autoencoders*
8 Ng A. *Sparse Autoencoders* CS294A Lecture notes