Convolutional Networks

EE 565: Pattern Recognition and Machine Learning

Matthew Martinez

November 15, 2016

What are Convolutional Neural Networks?

- Specialized Neural Network for processing data with a grid-like structure

What are Convolutional Neural Networks?

- Specialized Neural Network for processing data with a grid-like structure
  - Time series data: 1-D grid, i.e. samples occur at regular time intervals
  - Image data: 2-D grid of pixels
- Applications:

What are Convolutional Neural Networks?

- Specialized Neural Network for processing data with a grid-like structure
  - Time series data: 1-D grid, i.e. samples occur at regular time intervals
  - Image data: 2-D grid of pixels
- Applications:
  - Image recognition
  - Video analysis
  - Natural language processing
  - Speech recognition
  - Playing Go

What are Convolutional Neural Networks?

- Specialized Neural Network for processing data with a grid-like structure
  - Time series data: 1-D grid, i.e. samples occur at regular time intervals
  - Image data: 2-D grid of pixels
- Applications:
  - Image recognition
  - Video analysis
  - Natural language processing
  - Speech recognition
  - Playing Go
- Neural networks that use convolution in place of matrix multiplication in at least one layer

Convolution

Continuous Time Convolution:

Definition

$$s(t) = \int x(a)w(t-a)da \tag{1}$$

Convolution

Continuous Time Convolution:

Definition

$$s(t) = \int x(a)w(t-a)da \tag{1}$$

Discrete Time Convolution:

Definition

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2}$$

Convolution

Continuous Time Convolution:

Definition

$$s(t) = \int x(a)w(t-a)da \tag{1}$$

Discrete Time Convolution:

Definition

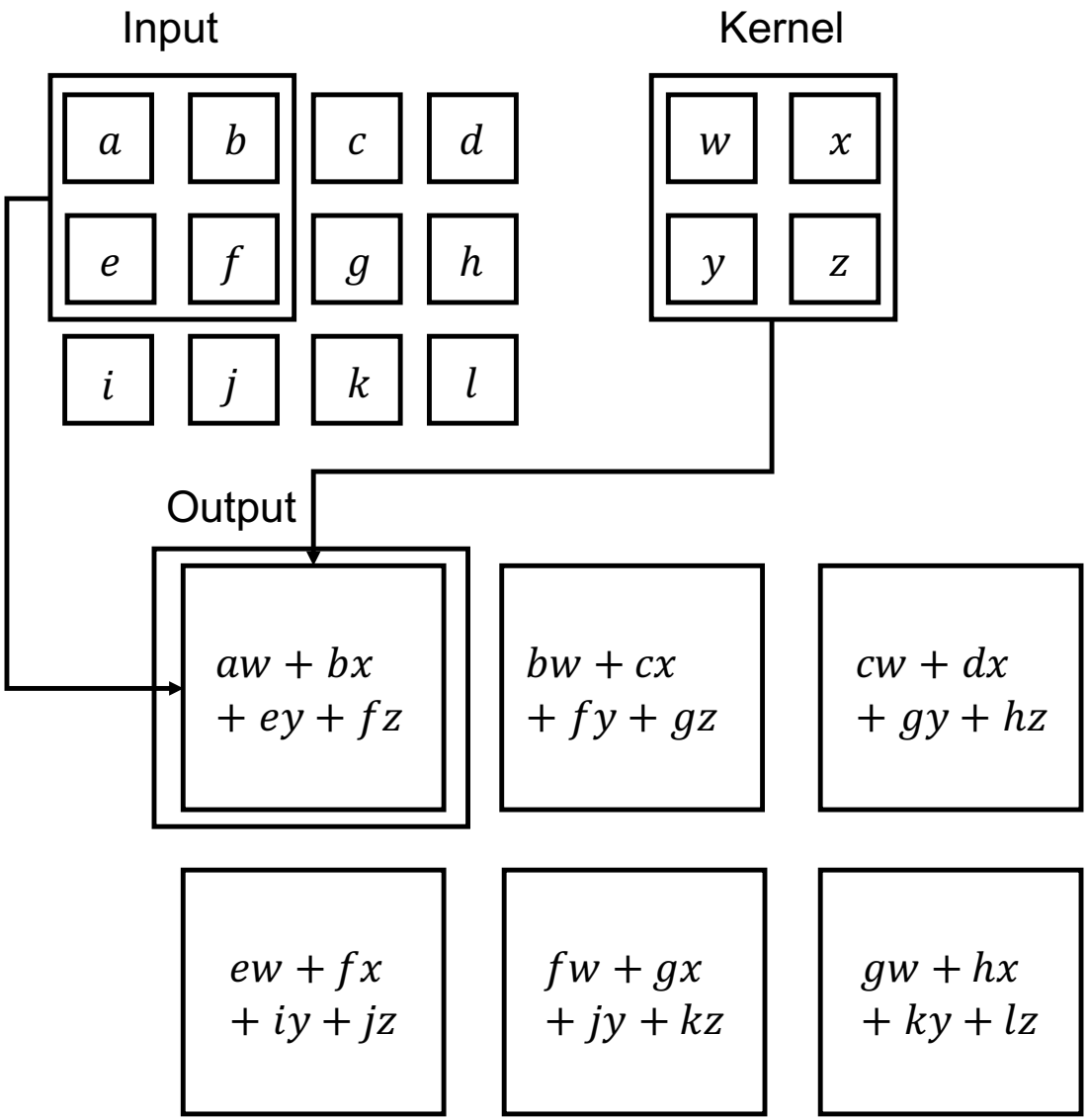
$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2}$$

2-D Convolution:

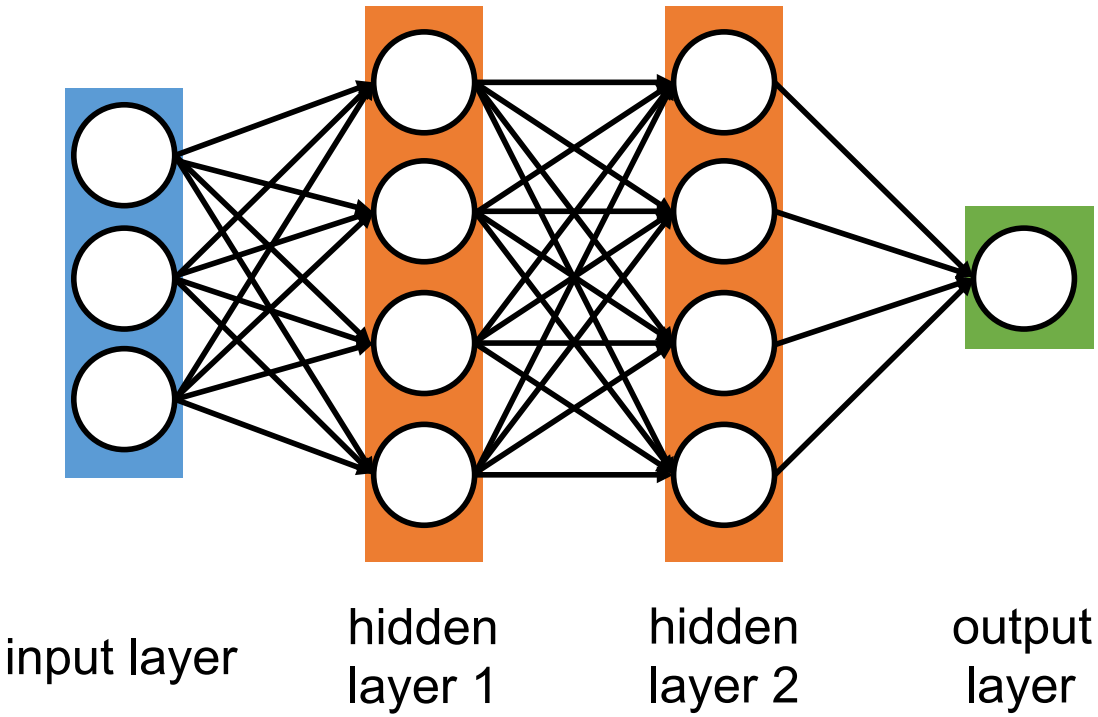
Definition

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{3}$$

Convolution Cont.

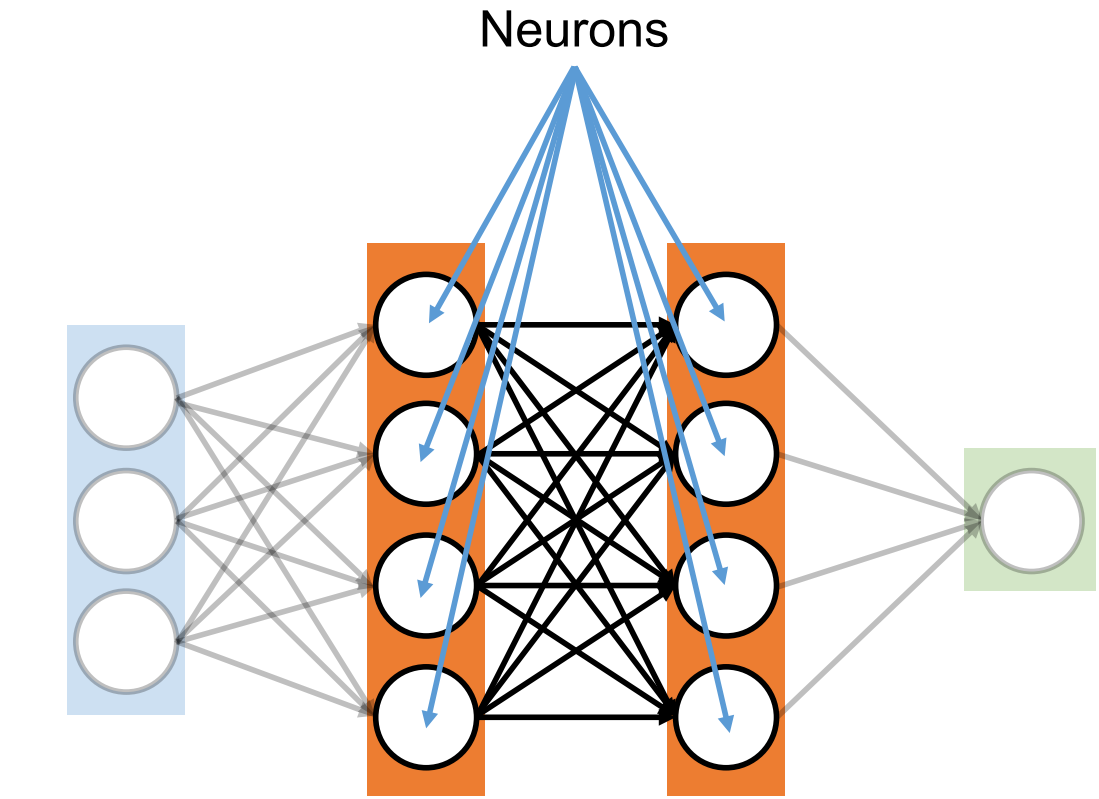


Feedforward Neural Networks



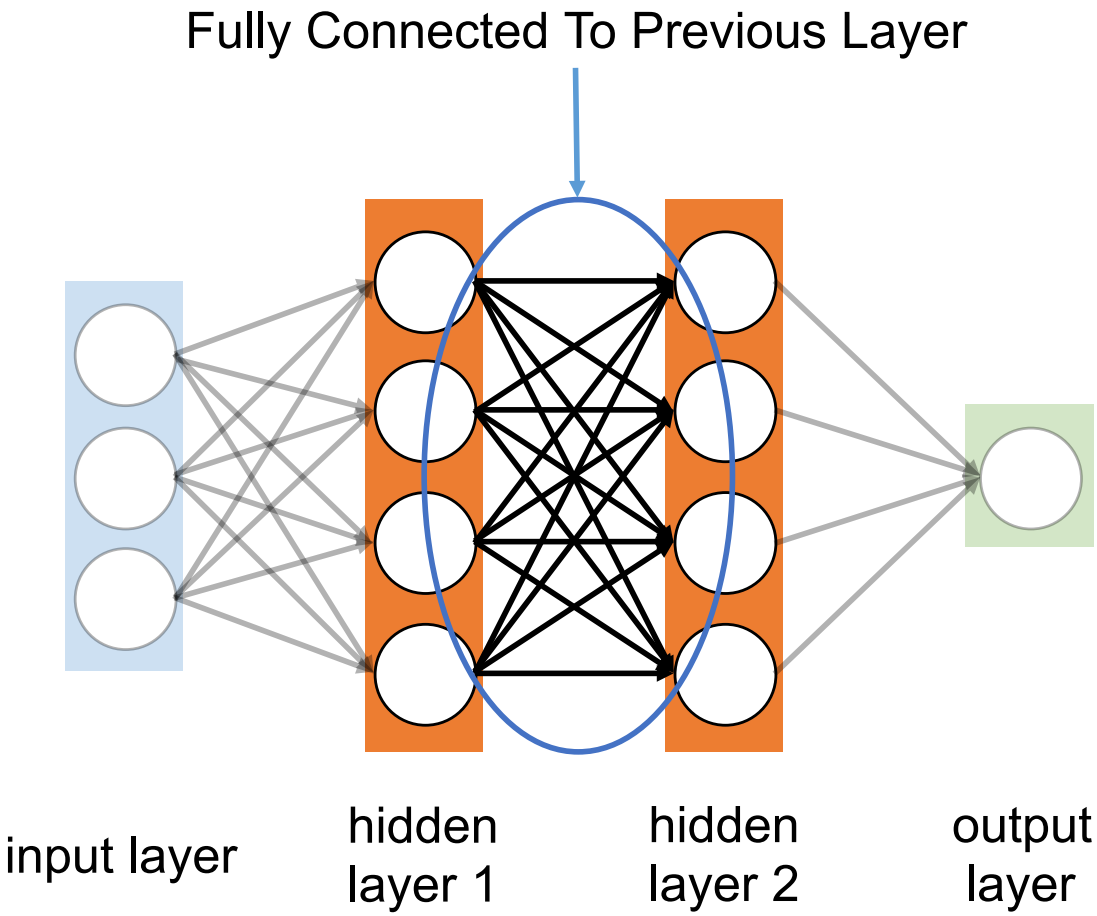
Feed Forward Neural Network

Feedforward Neural Networks Cont.



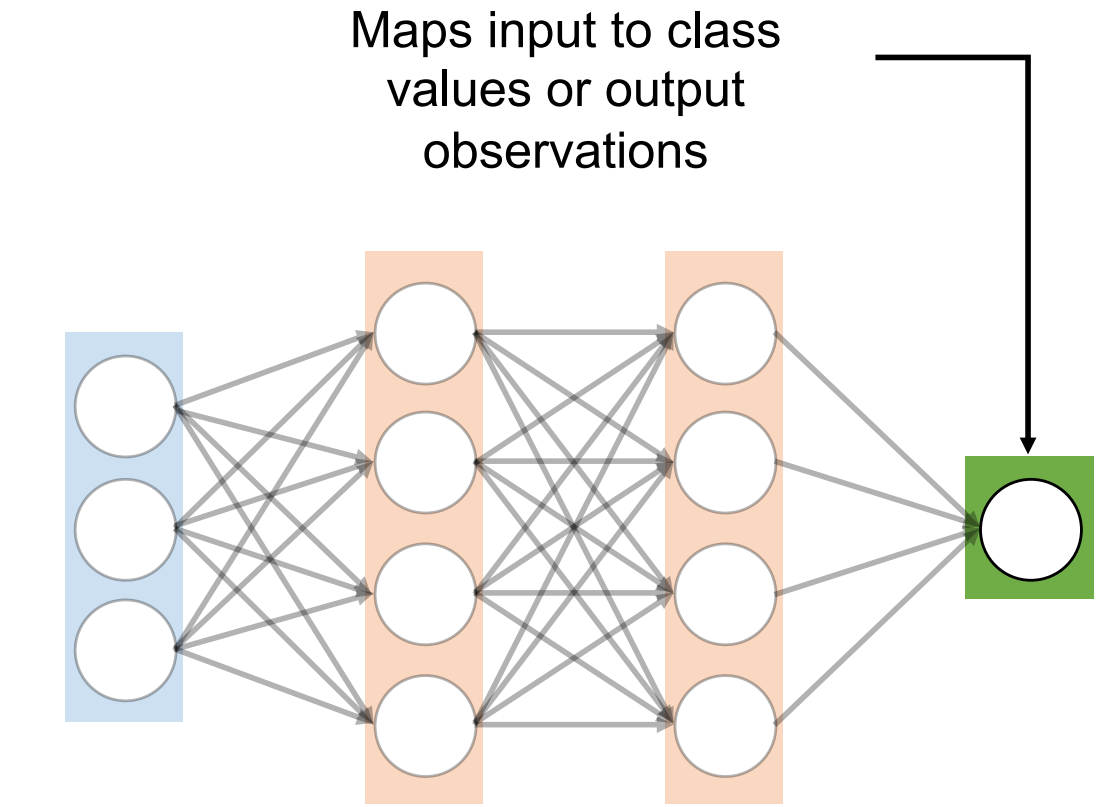
Feed Forward Neural Network

Feedforward Neural Networks Cont.



Feed Forward Neural Network

Feedforward Neural Networks Cont.



Feed Forward Neural Network

Motivation

Feedforward Neural Networks:

Motivation

Feedforward Neural Networks:

- Don't scale well to full images

Motivation

Feedforward Neural Networks:

- Don't scale well to full images
- CIFAR-10: $32 \times 32 \times 3$ (Height x Width x Color Channels)

Motivation

Feedforward Neural Networks:

- Don't scale well to full images
- CIFAR-10: $32 \times 32 \times 3$ (Height x Width x Color Channels)
  - 3072 weights in the first fully connected layer

Motivation

Feedforward Neural Networks:

- Don't scale well to full images
- CIFAR-10: $32 \times 32 \times 3$ (Height x Width x Color Channels)
  - 3072 weights in the first fully connected layer
- Larger image: $200 \times 200 \times 3$

Motivation

Feedforward Neural Networks:

- Don't scale well to full images
- CIFAR-10: $32 \times 32 \times 3$ (Height x Width x Color Channels)
  - 3072 weights in the first fully connected layer
- Larger image: $200 \times 200 \times 3$
  - $120,000$ weights in the first fully connected layer

Motivation

Feedforward Neural Networks:

- Don't scale well to full images
- CIFAR-10: $32 \times 32 \times 3$ (Height x Width x Color Channels)
  - 3072 weights in the first fully connected layer
- Larger image: $200 \times 200 \times 3$
  - $120,000$ weights in the first fully connected layer
- leads to overfitting

Motivation

Feedforward Neural Networks:

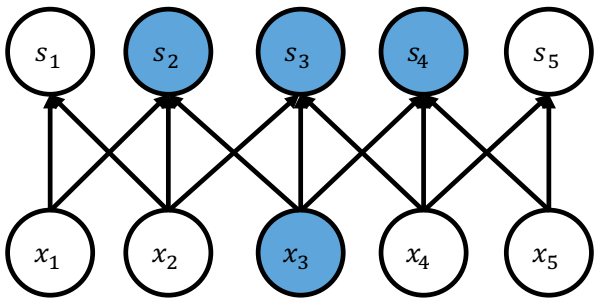
- Don't scale well to full images
- CIFAR-10: $32 \times 32 \times 3$ (Height x Width x Color Channels)
  - 3072 weights in the first fully connected layer
- Larger image: $200 \times 200 \times 3$
  - $120,000$ weights in the first fully connected layer
- leads to overfitting
- Three principles that help improve machine learning systems:
  - Sparse interactions
  - Parameter sharing
  - Equivarient representations

Sparse Interactions

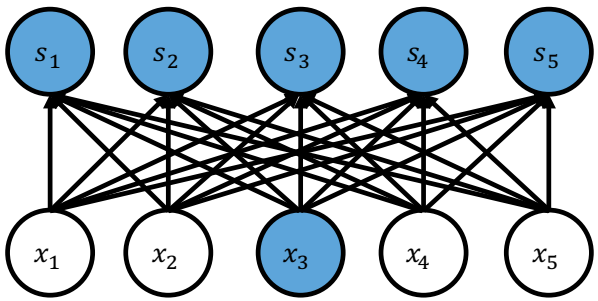
Sparsity

- Sparsity is achieved by using a kernel smaller than the input
- Example:
  - Image has thousands or millions of pixels
  - Can detect small meaningful features with kernels that use tens to hundreds of pixels
  - Requires fewer parameters to store
  - Matrix multiplication for $m \times n$ parameters requires $\mathcal{O}(m \times n)$ operations
  - Let connections to k, then sparsity requires $\mathcal{O}(m \times k)$ operations

Convolution



Fully Connected

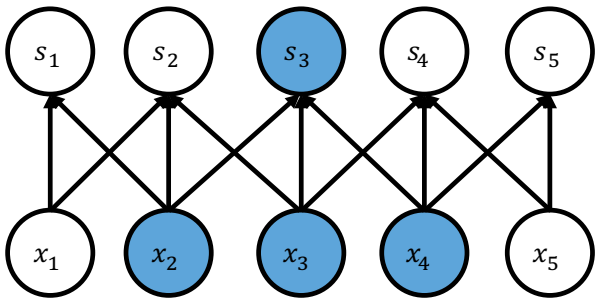


Sparse Interactions

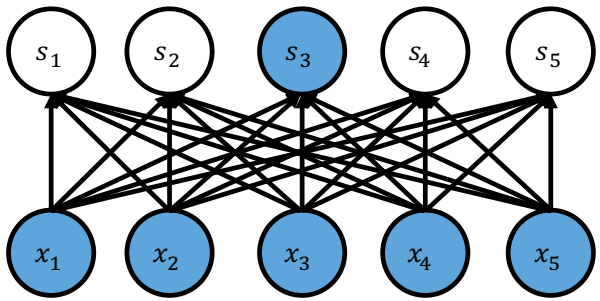
Sparsity

- Sparsity is achieved by using a kernel smaller than the input
- Example:
  - Image has thousands or millions of pixels
  - Can detect small meaningful features with kernels that use tens to hundreds of pixels
  - Requires fewer parameters to store
  - Matrix multiplication for $m \times n$ parameters requires $\mathcal{O}(m \times n)$ operations
  - Let connections to k, then sparsity requires $\mathcal{O}(m \times k)$ operations

Convolution



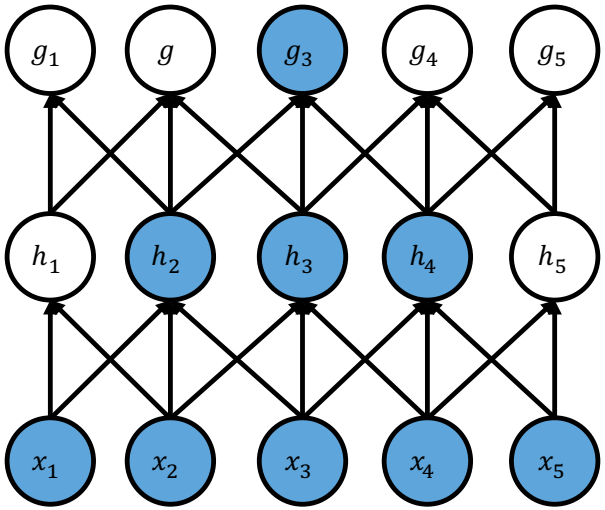
Fully Connected



Sparse Interactions

Deep Layers

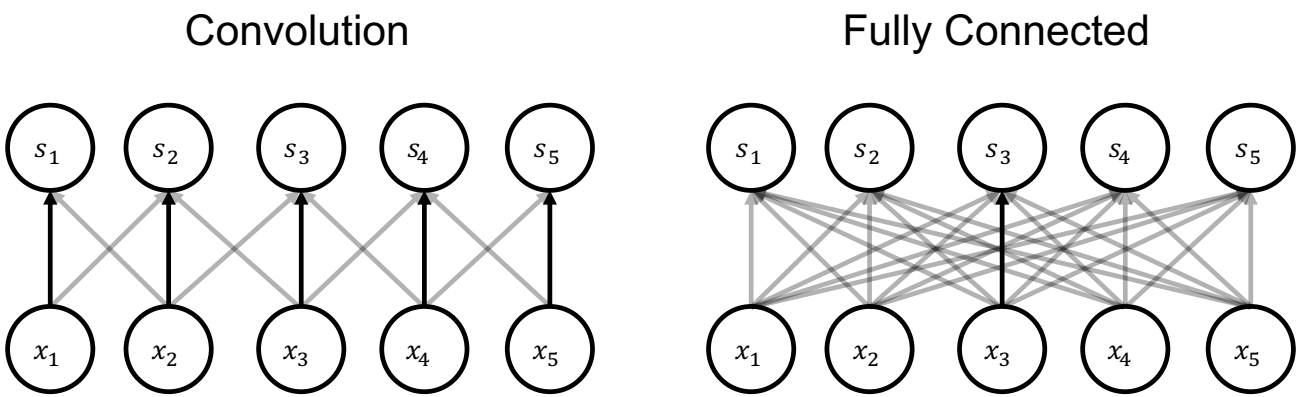
- Indirectly interact with a larger portion of the network
- Allows the network to efficiently describe complicated interactions between many variables



Parameter Sharing

Parameter Sharing

- Use same parameter for more than one function within the model
- Feedforward neural network, each element of the weight matrix is used exactly once
- Convolutional neural network, learn one set of parameters
  - Each member of the kernel is used at every position of the input
  - More efficient than matrix multiplication in terms of memory requirement and statistical efficiency



Equivariance

Equivariance

- If the input changes then the output changes in the same way
- A function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$

Equivariance

Equivariance

- If the input changes then the output changes in the same way
- A function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$

Convolution Example

- Let I be a function giving image brightness at integer coordinaes
- Let g be a function mapping one image function to another image function such that $I' = g(I)$
- $I'(x, y) = I(x - 1, y)$
- If we apply the transformation to I and then convolution it is the same as applying convolution then the transformation g to the output
- Convolution is not equivariant to transforms which affect the scale or rotation of the image

Spatial Arrangement

The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{4}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride

Spatial Arrangement

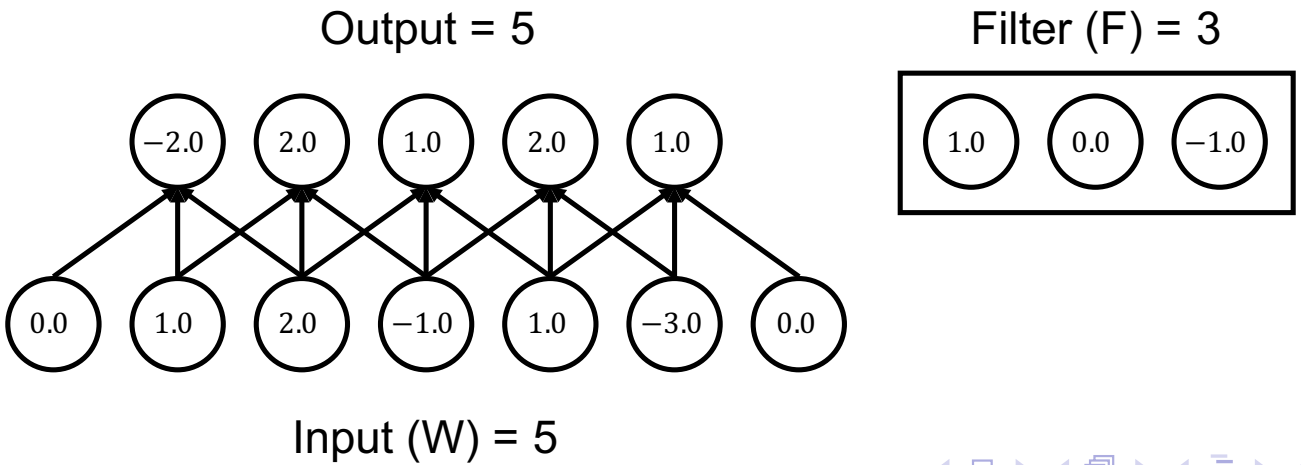
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{4}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

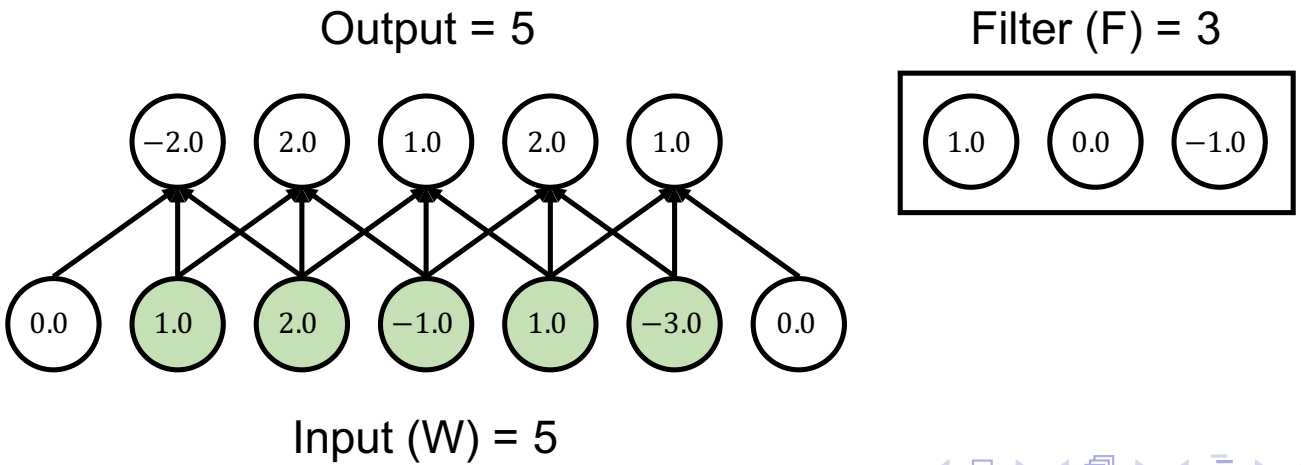
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{5}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

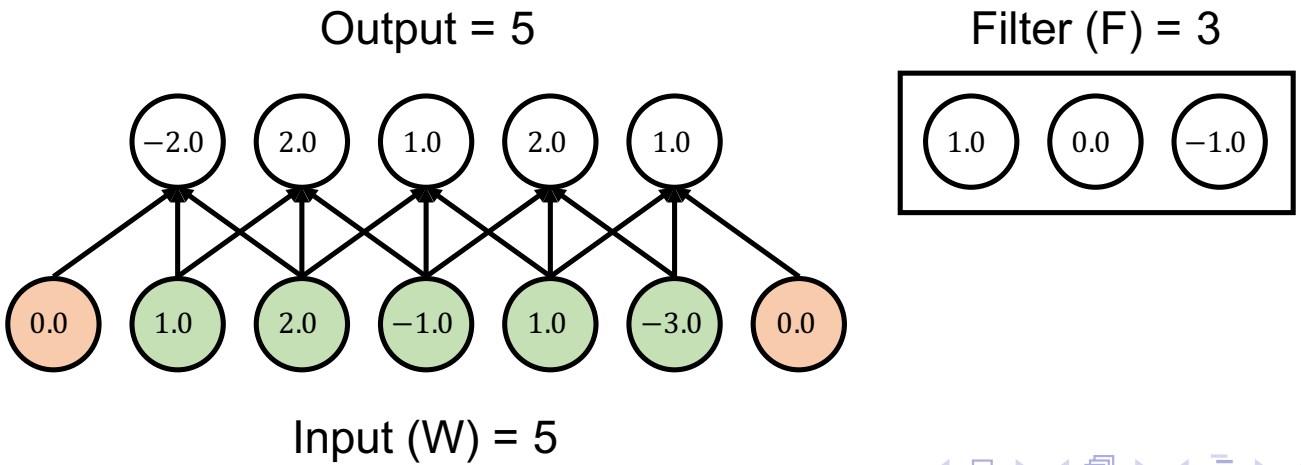
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{6}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

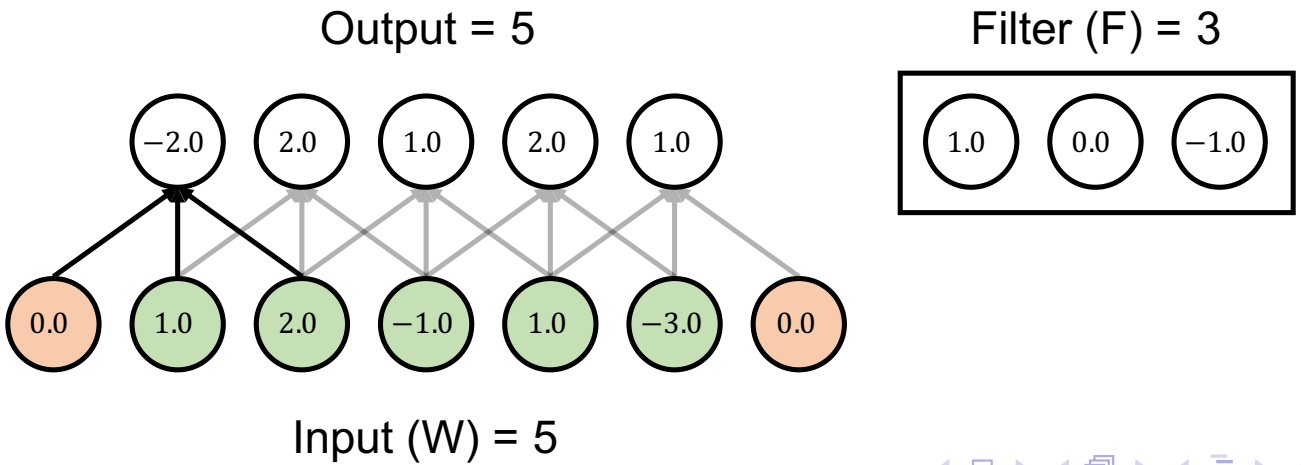
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{7}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

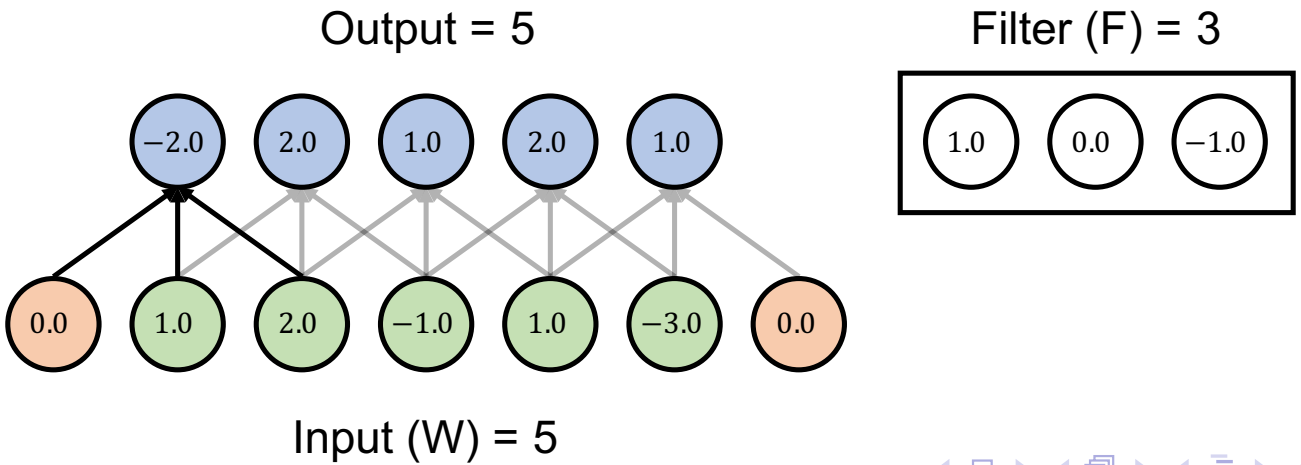
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{8}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

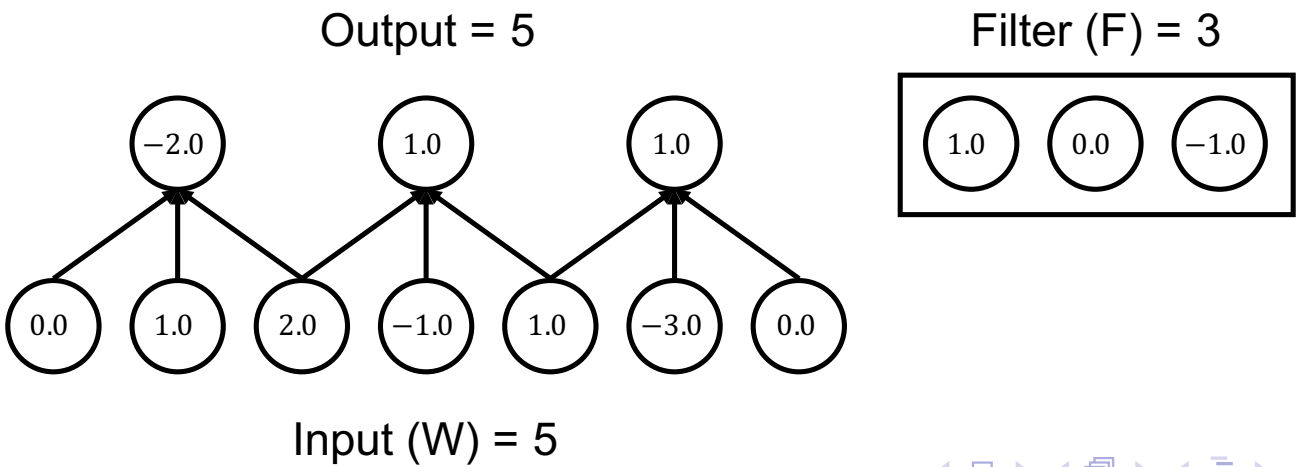
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{9}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

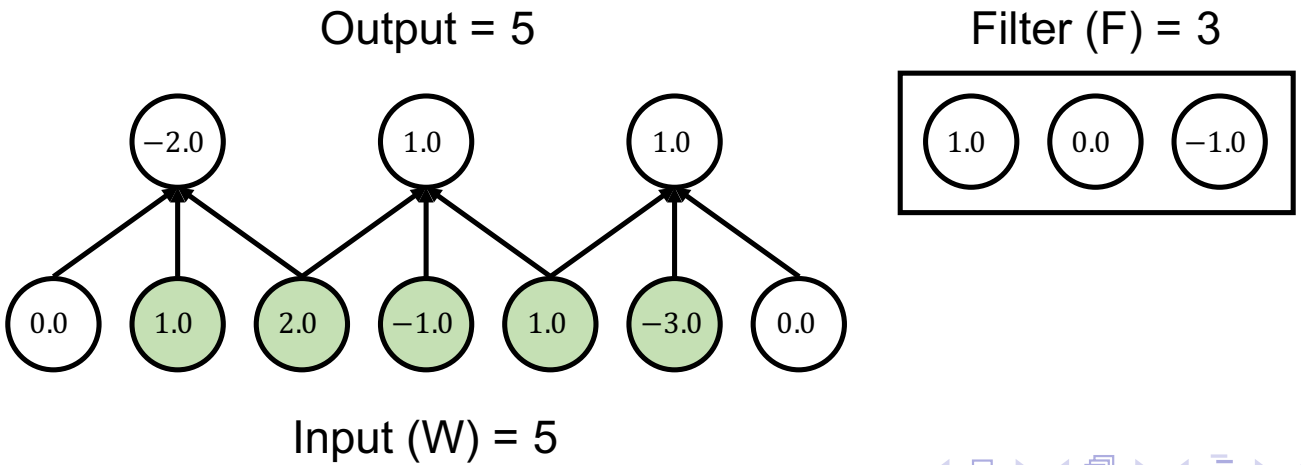
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{10}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

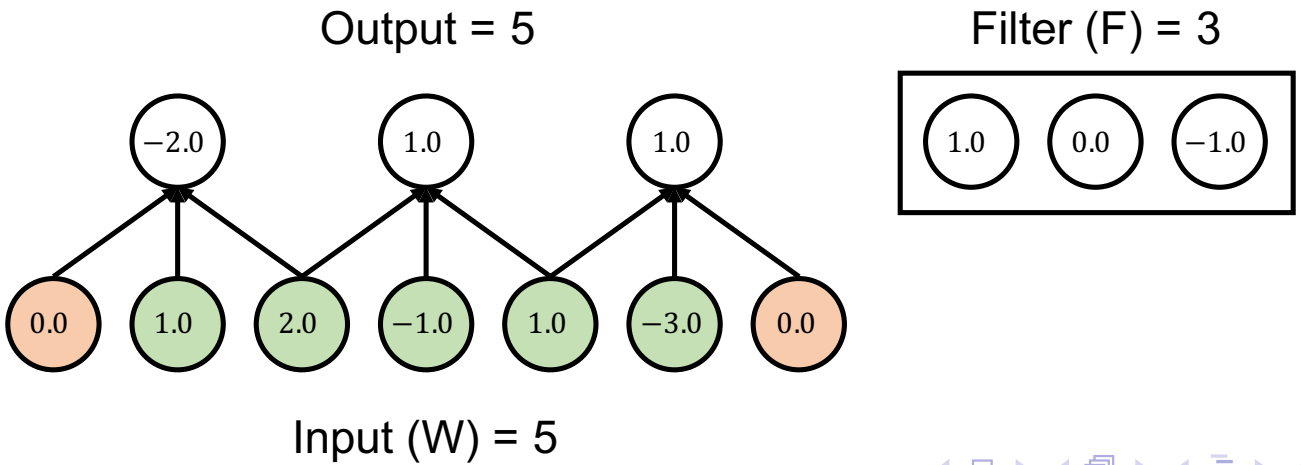
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{11}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

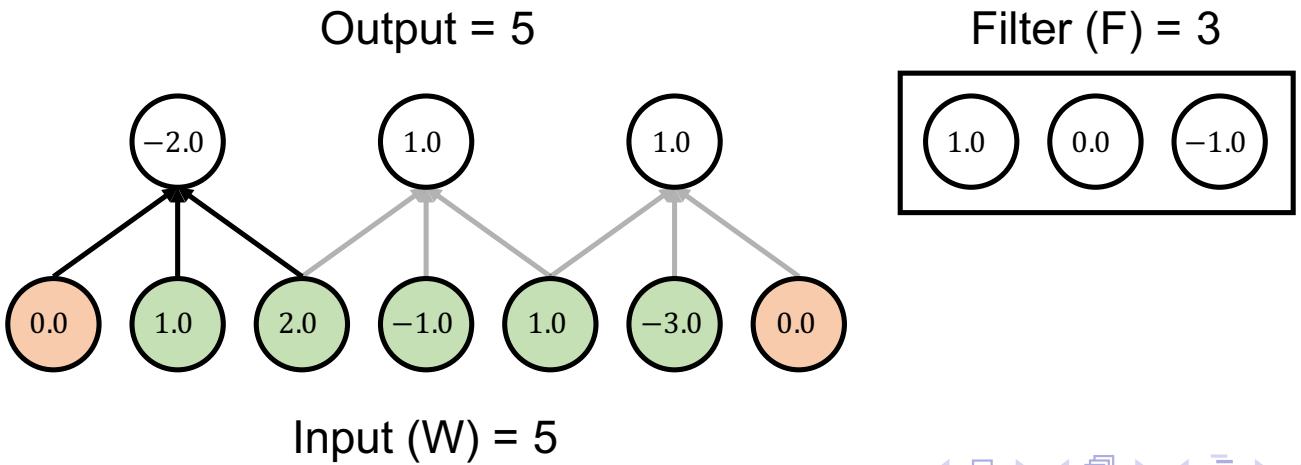
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{12}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Spatial Arrangement

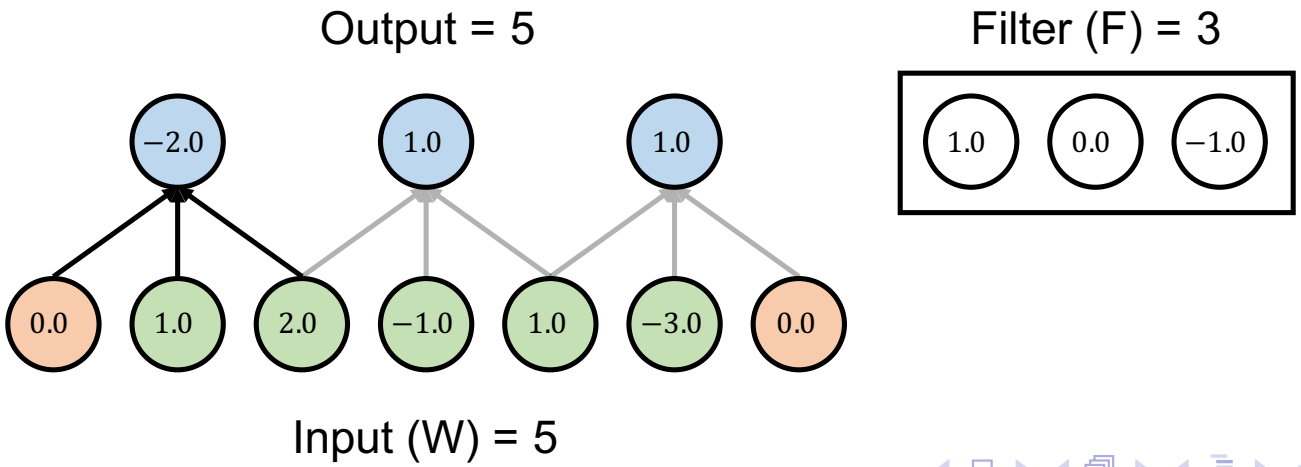
The output is controlled by three hyperparameters

- Depth: number of filters to be used
- Stride: number of pixels/samples to slide filter
- Zero-padding: number of zeros placed around boarder

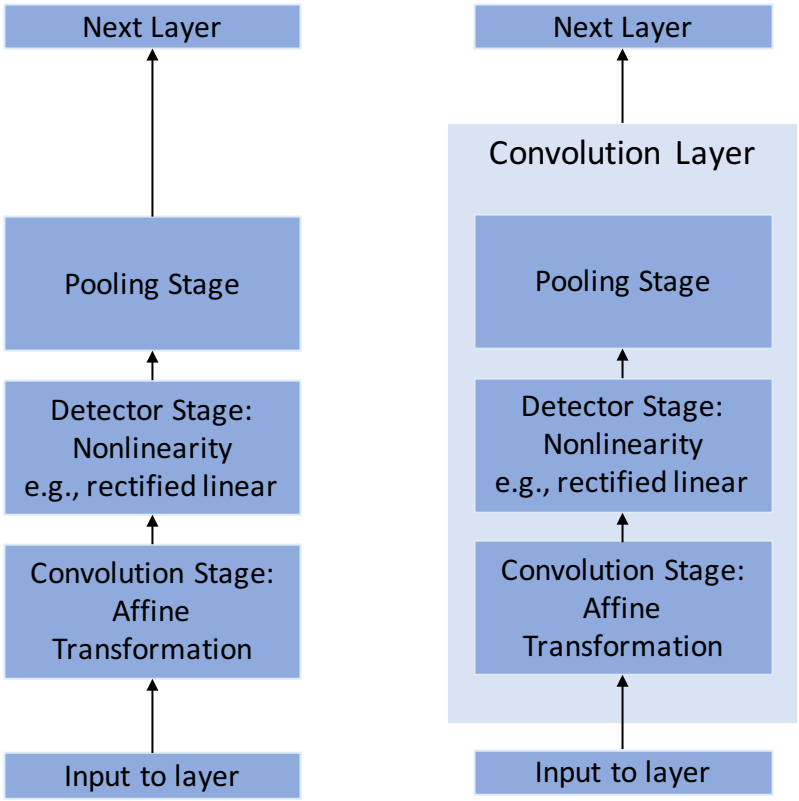
Output can be calculated as:

$$1 + (W - F + 2P)/S \tag{13}$$

where, W is the input dimension, F is the size of the filter, P is the size of the zero padding, S is the size of the stride



Network Architecture



Pooling

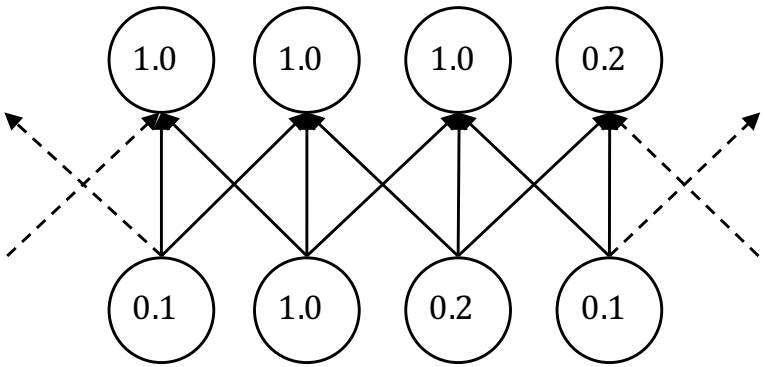
Pooling

- Replaces output at a certain location with summary statistics
- Types of pooling
  - Max, Average, L2 Norm, Weighted Average
- Max Pooling: reports the maximum output within a rectangular neighborhood
- Helps representation become invariant to small translations in the input

Pooling

Pooling

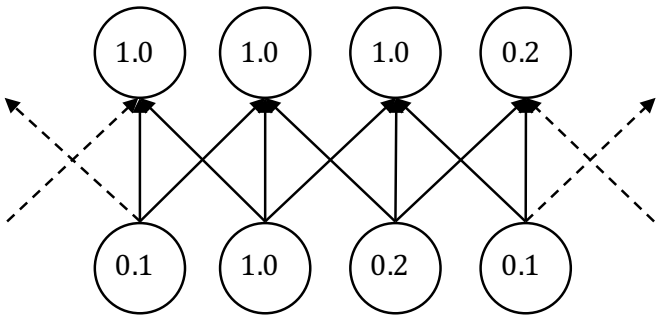
- Replaces output at a certain location with summary statistics
- Types of pooling
  - Max, Average, L2 Norm, Weighted Average
- Max Pooling: reports the maximum output within a rectangular neighborhood
- Helps representation become invariant to small translations in the input

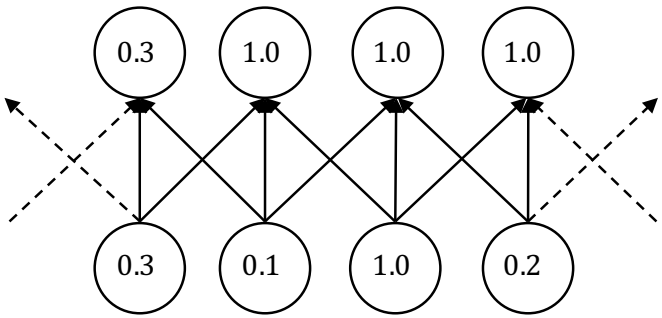


Pooling

Pooling

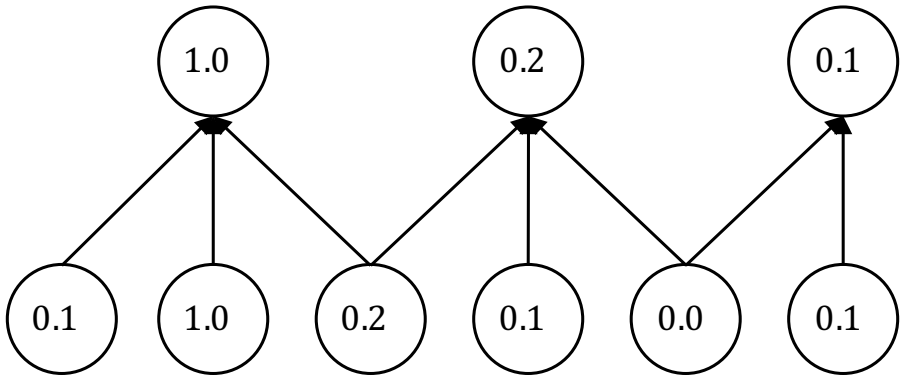
- Replaces output at a certain location with summary statistics
- Types of pooling
  - Max, Average, L2 Norm, Weighted Average
- Max Pooling: reports the maximum output within a rectangular neighborhood
- Helps representation become invariant to small translations in the input





Pooling with Downsampling

- Uses fewer pooling units than detector units
- Reports summary of statistics for regions spaced k pixels apart
- Next layer has approximately k fewer inputs



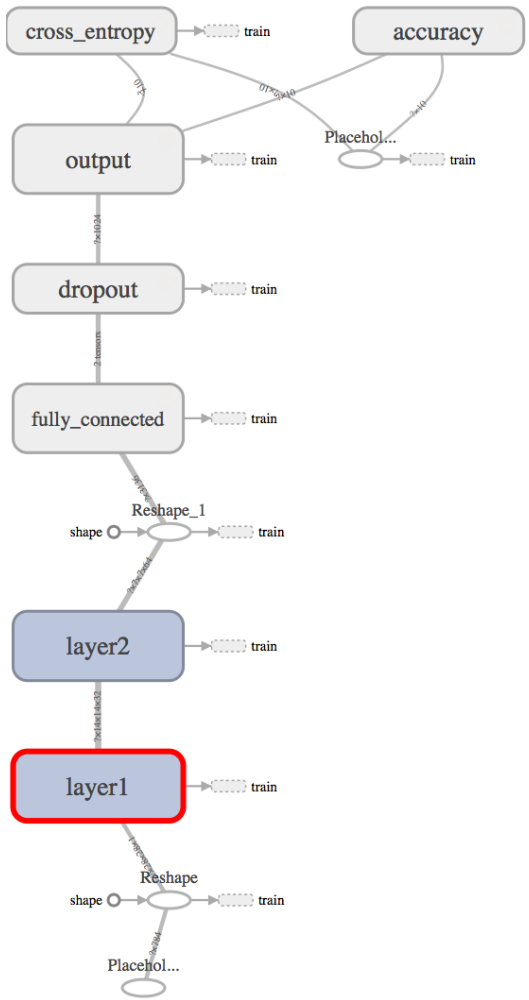
MNIST: Network

First hidden convolutional layer

- Number of filters: 32
- Filter size: 5×5
- Activation: Rectified Linear Unit (ReLU)
- Pooling: Max Pooling with downsampling of 2

Second hidden convolutional layer

- Number of filters: 64
- Filter size: 5×5
- Activation: Rectified Linear Unit (ReLU)
- Pooling: Max Pooling with downsampling of 2



MNIST: Network

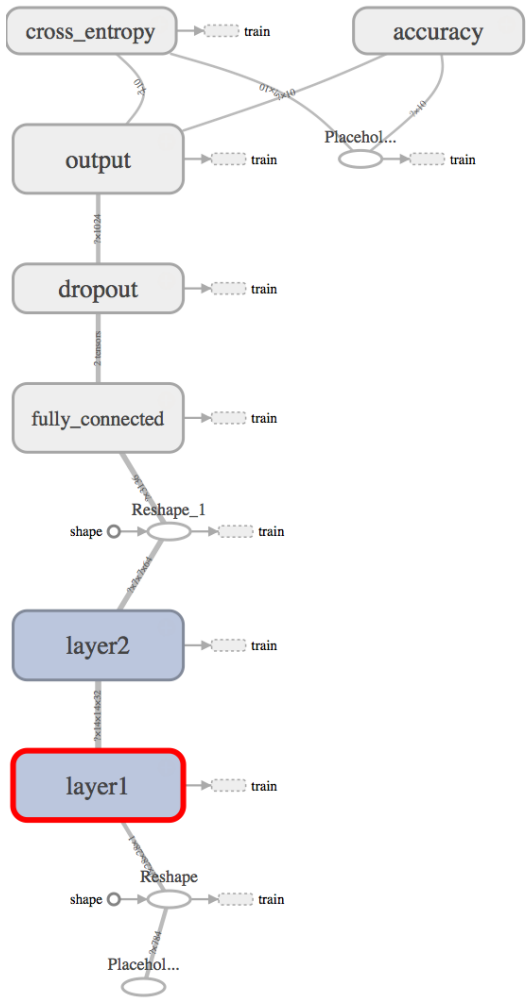
Fully Connected Layer

- Input: 3136
- Output Size: 1024,
- Activation: Rectified Linear Unit (ReLU)

Dropout, cost function, optimization

- Keep Probability: 80%
- Cost Function: Cross entropy
- Optimization: ADAM
- Learning Rate: $1e-4$
- Epochs: 20001

Accuracy = 99.2%



MNIST

MNIST: Classification of handwritten digits

Results [error %]	Reference
0.21	Regularization of Neural Networks using DropConnect
0.23	Multi-column Deep Neural Networks for Image Classification
0.23	Augmented Pattern Classification with Neural Networks
0.24	Batch-normalized Maxout Network in Network
0.29	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated and Tree

CIFAR-10

CIFAR-10: Classify 32x32 color images, 10 classes

Results [accuracy %]	Reference
96.53	Fractional Max Pooling
95.59	Striving for Simplicity: The All Convolutional Net
94.16	All you need is a good init
94.00	Lessons Learned from Manually Classifying CIFAR-10
93.95	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated and Tree

SVHN

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST(e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

SVHN

Results [error %]	Reference
1.69	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated and Tree
1.76	Competitive Multi-scale Convolution
1.77	Recurrent Convolutional Neural Network Object Recognition
1.81	Batch-normalized Maxout Network in Network
1.92	Deeply-Supervised Nets