CS5489
Lecture 9.2: Neural Networks and Deep Learning: Part II

Kede Ma

City University of Hong Kong (Dongguan)

Slide template by courtesy of Benjamin M. Marlin

Part of slides are borrowed from the Stanford University CS231n course

Outline

Gradient Descent with the Chain-Rule

- Suppose we have a 2-layer network
  - ℓ is the cost function
  - g_1, g_2 are the output functions of the two layers
    - $g_j(\mathbf{x}) = f(\mathbf{W}_j^T \mathbf{x})$, and $\mathbf{W}_1$, $\mathbf{W}_2$ are the weight matrices
  - Prediction for input $\mathbf{x}$: $\hat{y} = g_2(g_1(\mathbf{x}))$
  - Cost for input $\mathbf{x}$: $\ell(\mathbf{x}) = \ell(g_2(g_1(\mathbf{x})))$
- Apply the chain rule to get the gradients of weights
  - $\frac{d\ell(\mathbf{x})}{d\mathbf{W}_2} = \frac{d\ell}{dg_2}\frac{dg_2}{d\mathbf{W}_2}$, $\frac{d\ell(\mathbf{x})}{d\mathbf{W}_1} = \frac{d\ell}{dg_2}\frac{dg_2}{dg_1}\frac{dg_1}{d\mathbf{W}_1}$
- Define a set of recursive relationships
  1. Calculate the output of each node from first to last layer
  2. Calculate the gradient of each node from last to first layer
- NOTE: the gradients multiply in each layer!
  - If two gradients are small (< 1), their product will be even smaller. This is the "vanishing gradient" problem

Backpropagation through Computational Graph

Backpropagation: a simple example

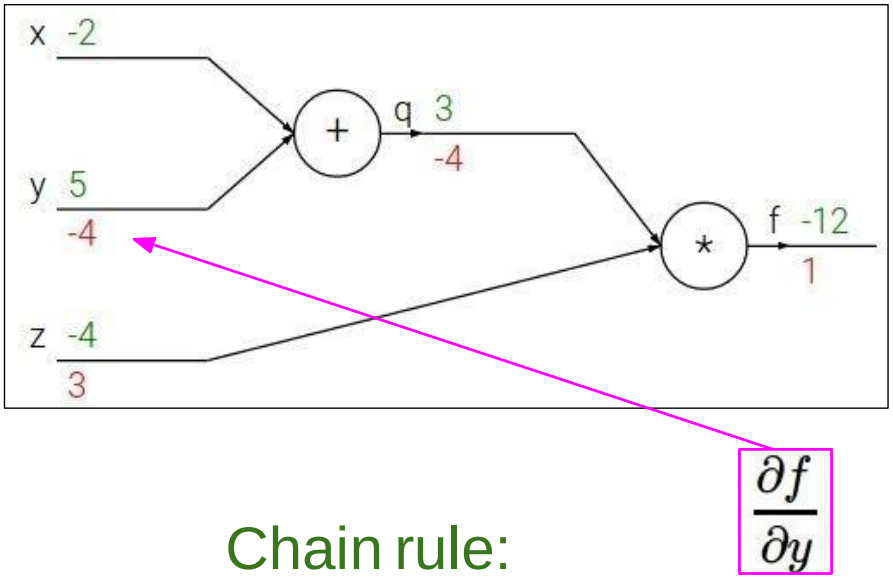
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



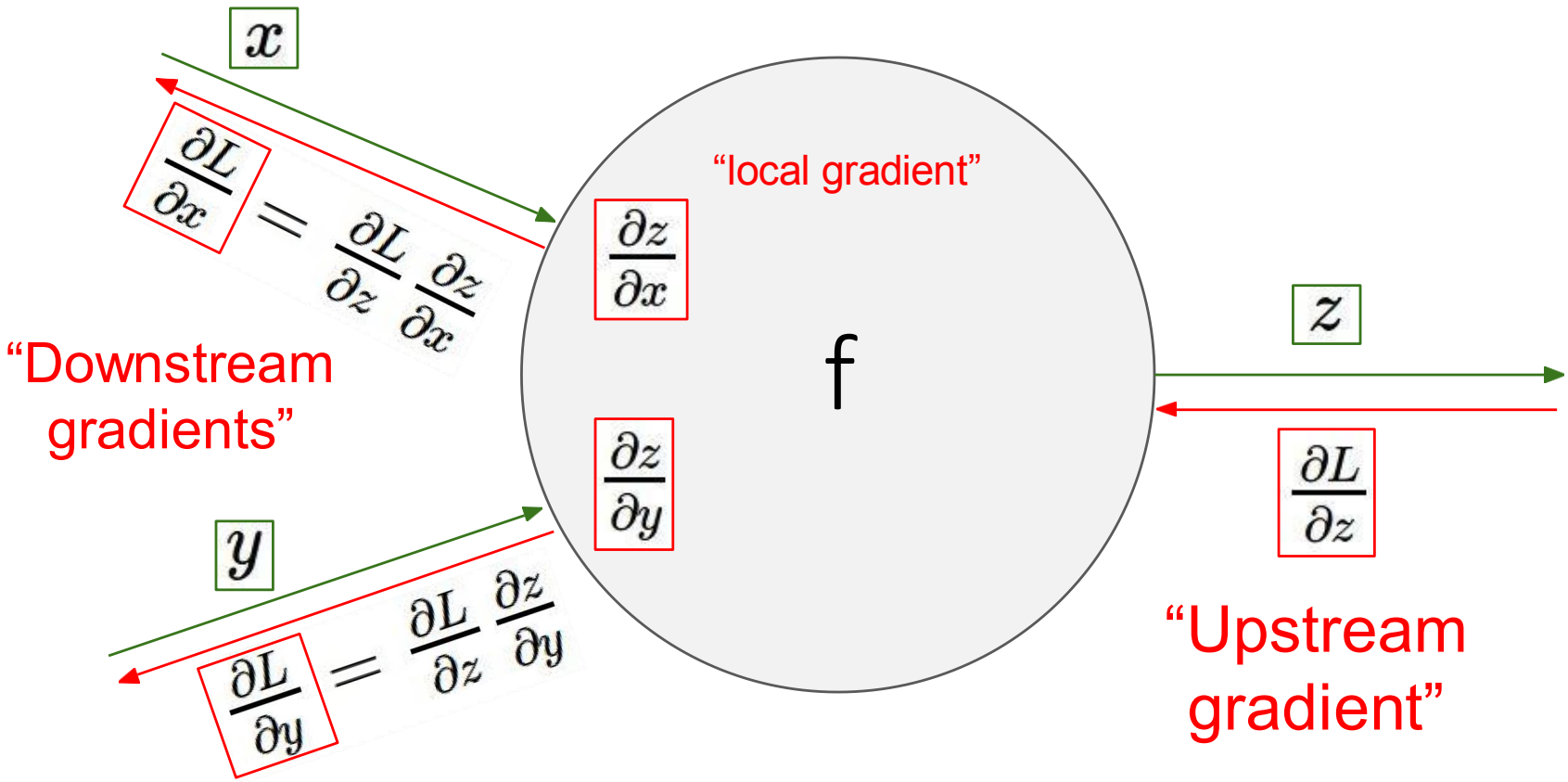
Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Upstream gradient

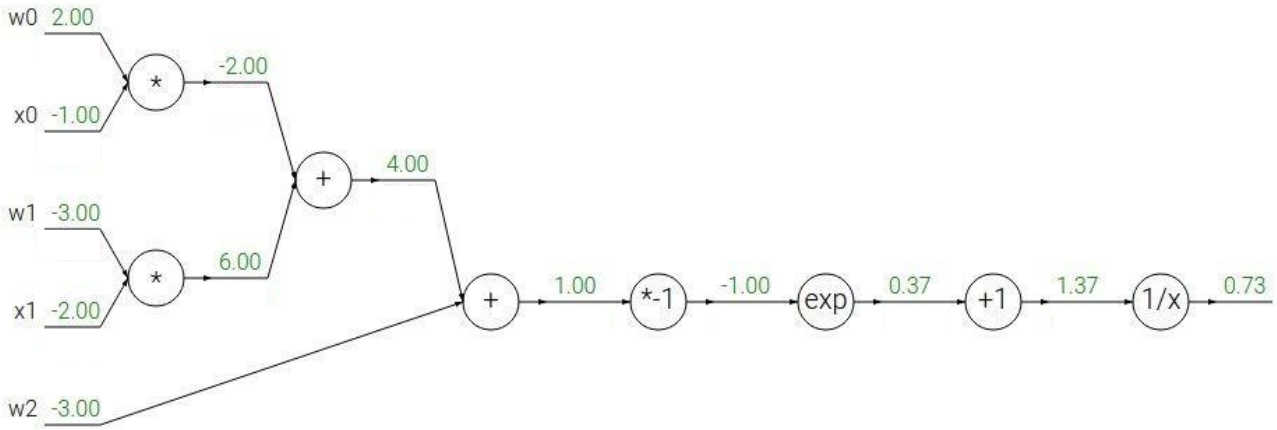
Local gradient

Backpropagation



Example: Backpropagation through Computational Graph

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

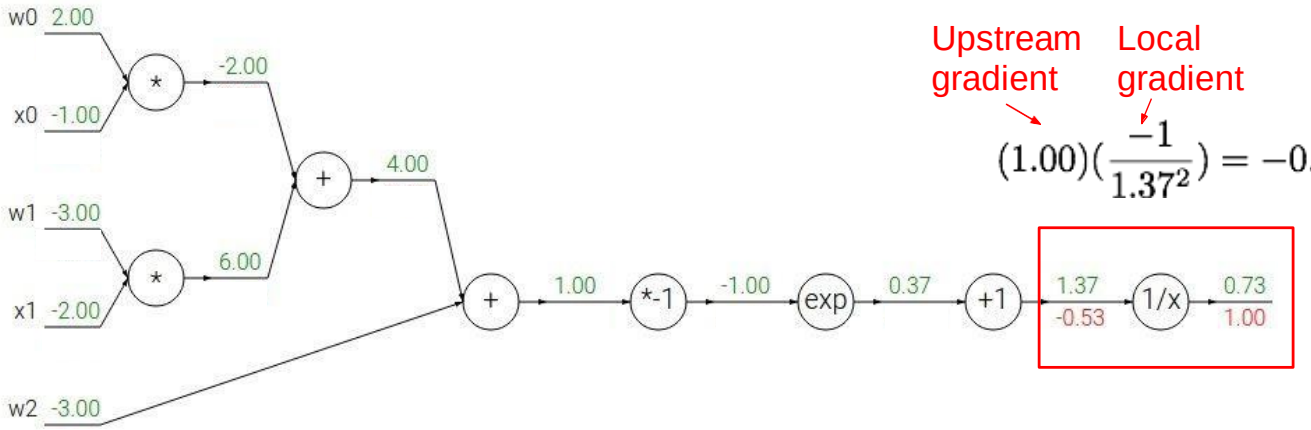
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Backpropagation through Computational Graph

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

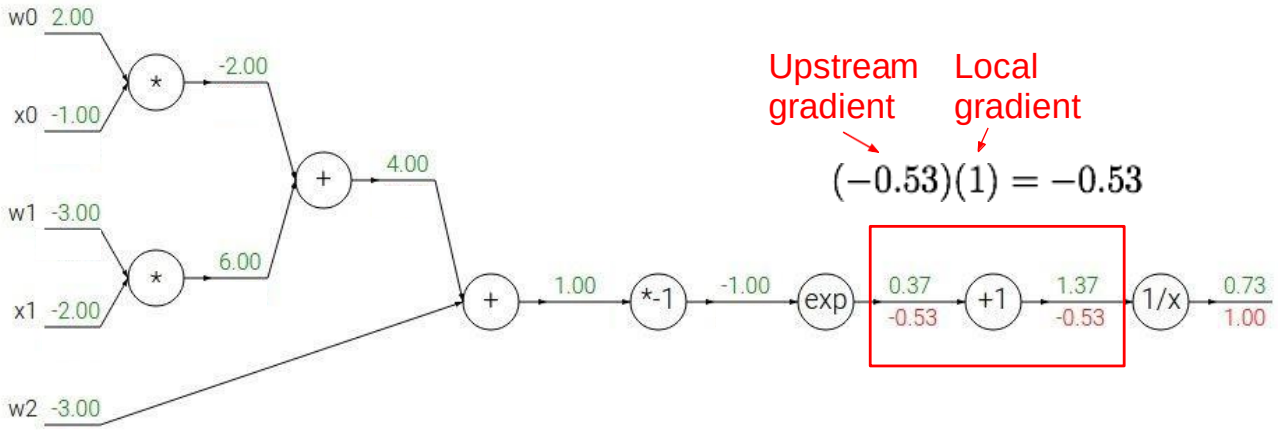
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Backpropagation through Computational Graph

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

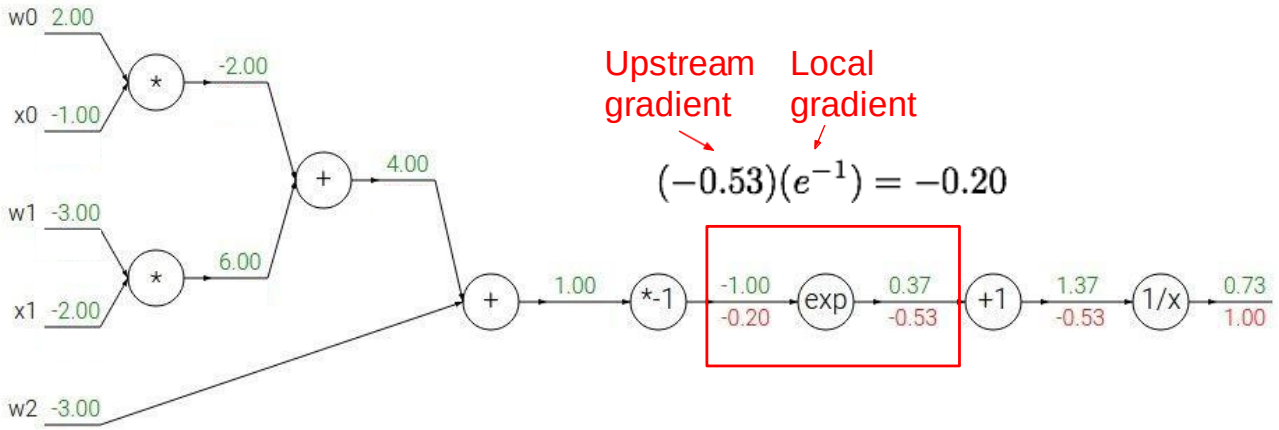
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Backpropagation through Computational Graph

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

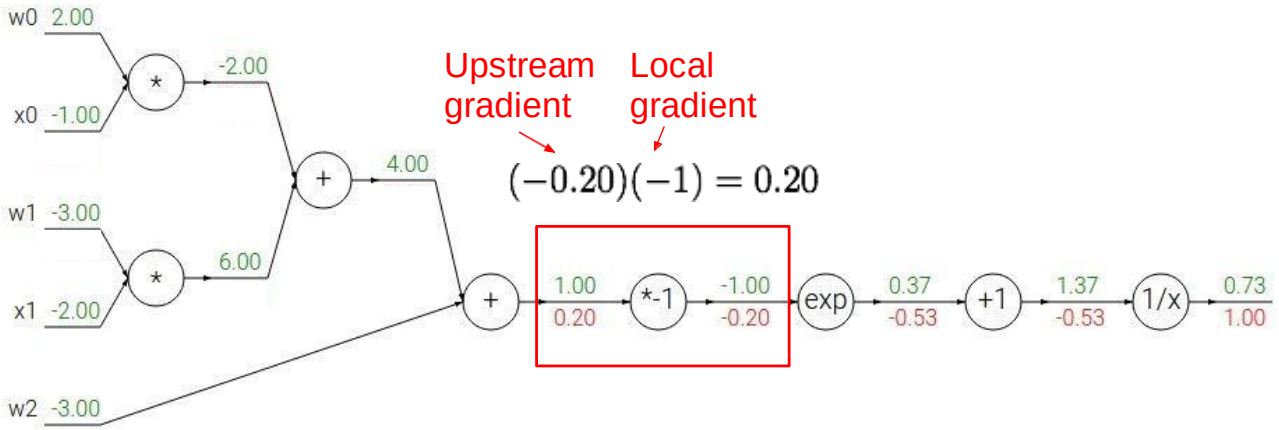
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Backpropagation through Computational Graph

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

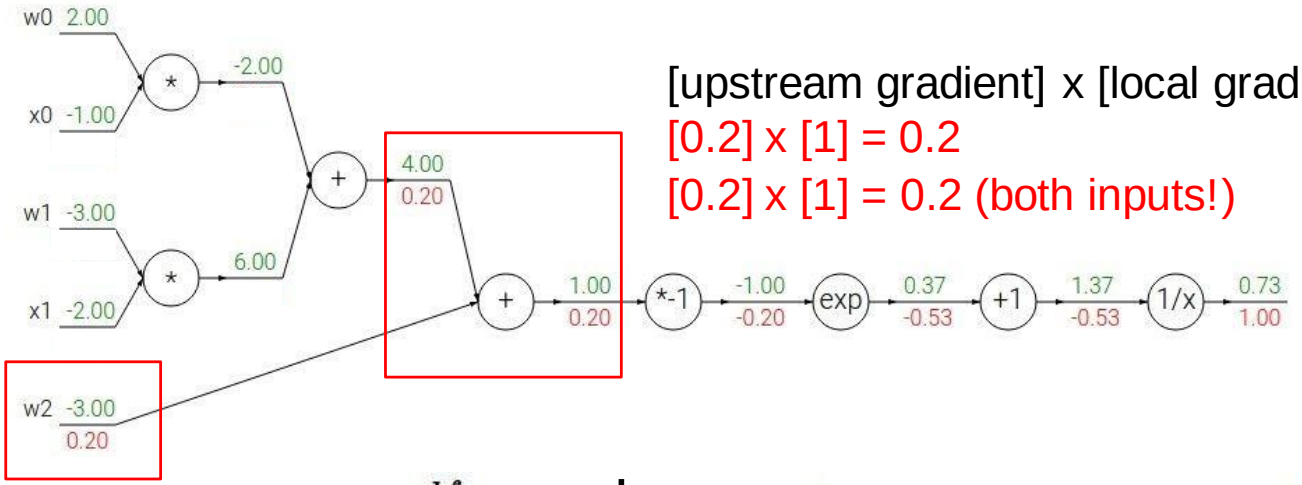
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Backpropagation through Computational Graph

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

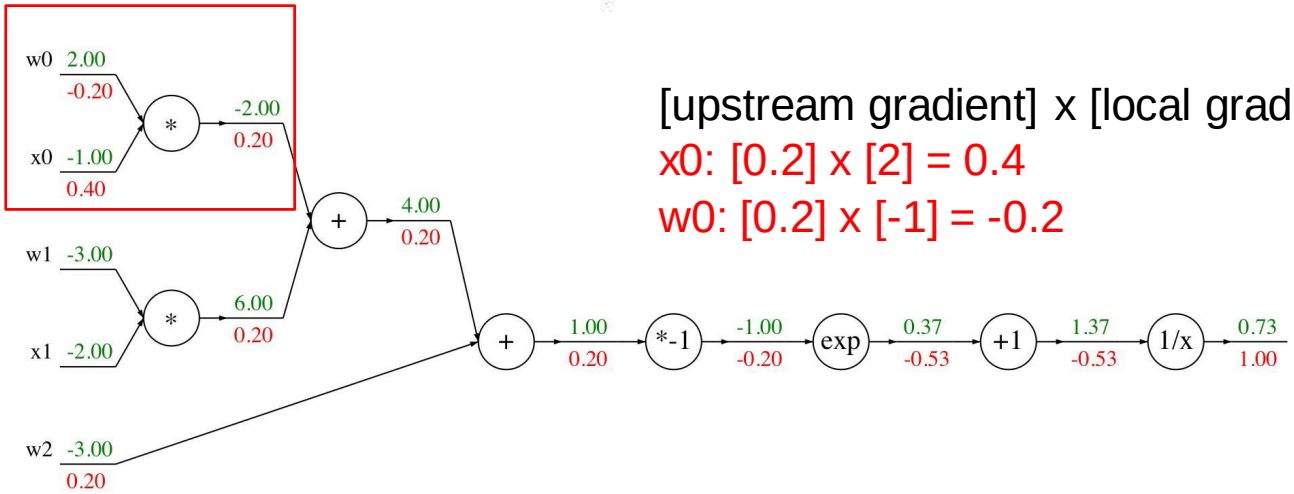
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Backpropagation through Computational Graph

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[upstream gradient] x [local gradient]

x0: [0.2] x [2] = 0.4

w0: [0.2] x [-1] = -0.2

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

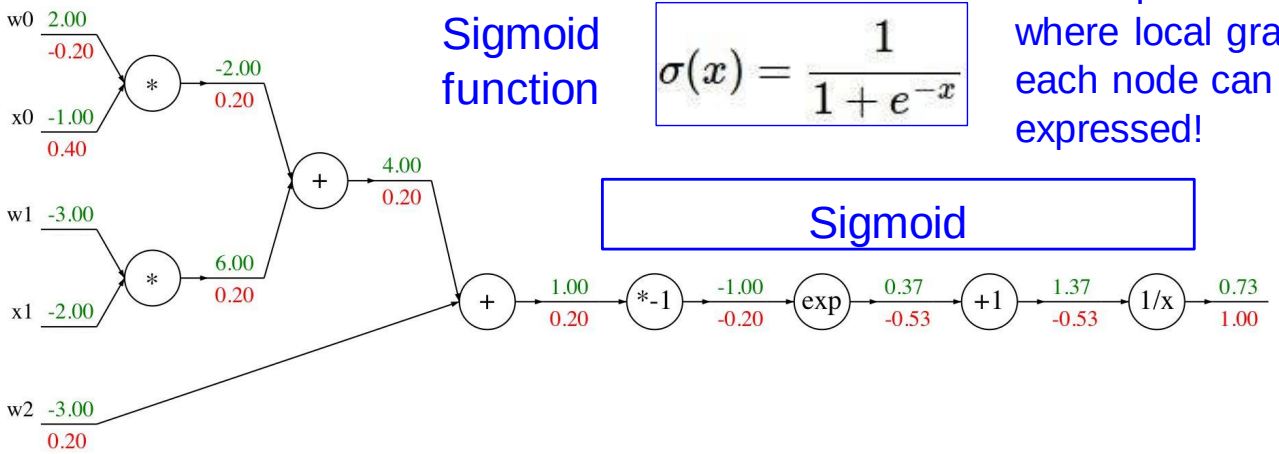
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Backpropagation through Computational Graph

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



Sigmoid function

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid

Sigmoid local gradient:

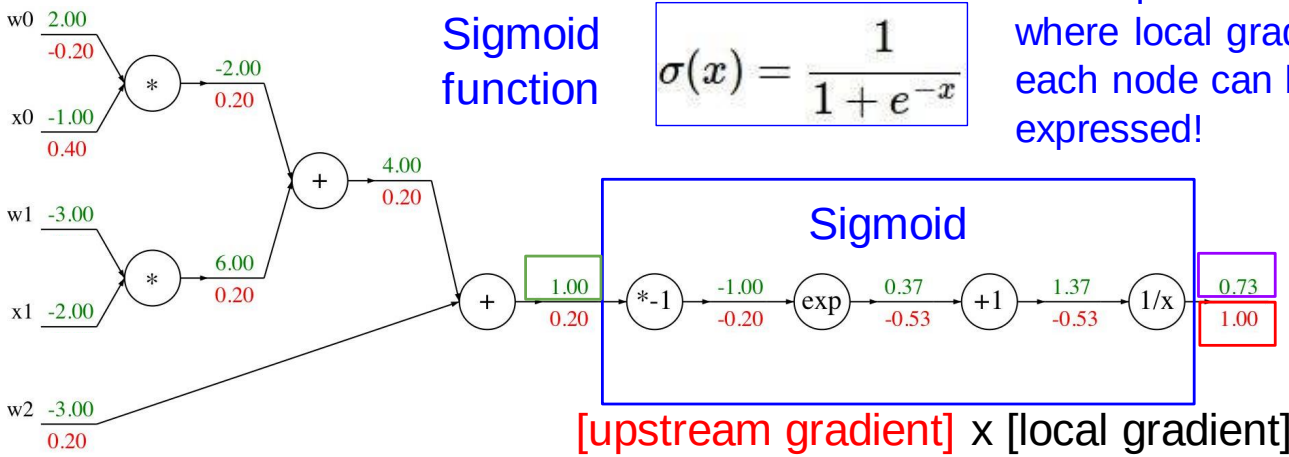
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Backpropagation through Computational Graph

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!



Sigmoid function

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

[upstream gradient] x [local gradient]

[1.00] x [(1 - 0.73) (0.73)] = 0.2

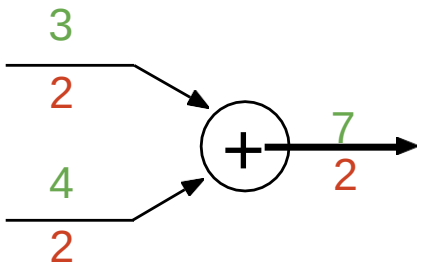
Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

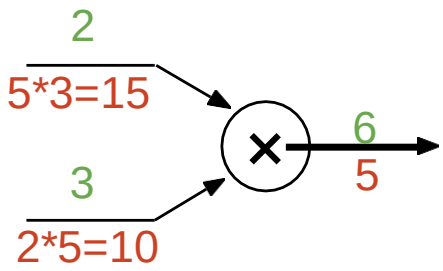
Backpropagation through Computational Graph

Patterns in gradient flow

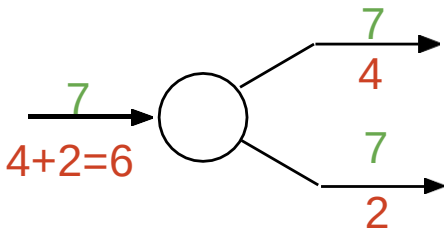
add gate: gradient distributor



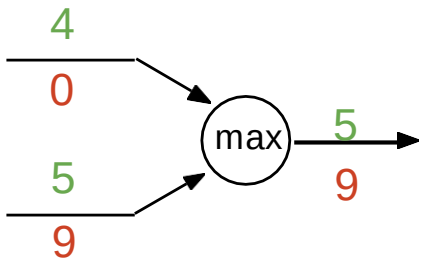
mul gate: "swap multiplier"



copy gate: gradient adder



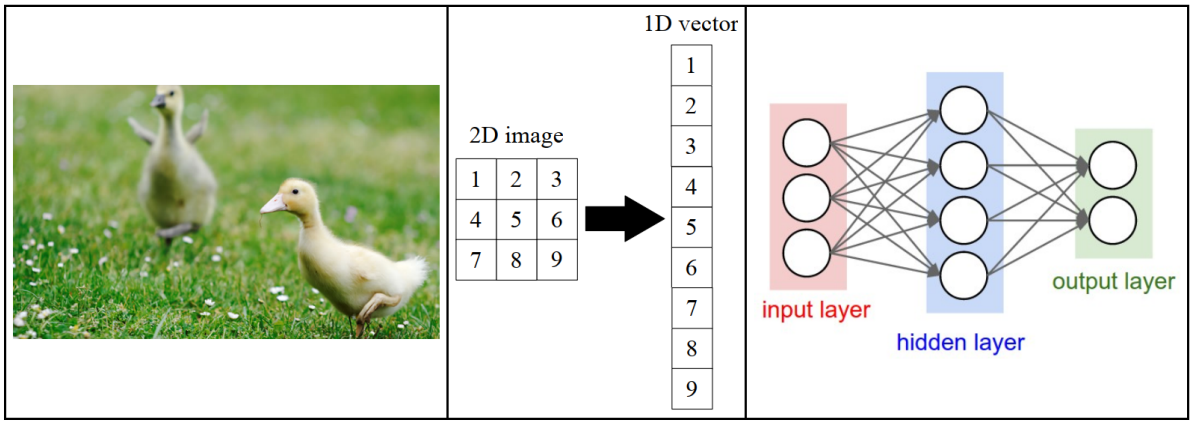
max gate: gradient router



Outline

Image Inputs and Neural Networks

- In MLP, each node accepts all nodes in the previous layer
  - For image input, we transform the image into a vector, which is the input into the MLP

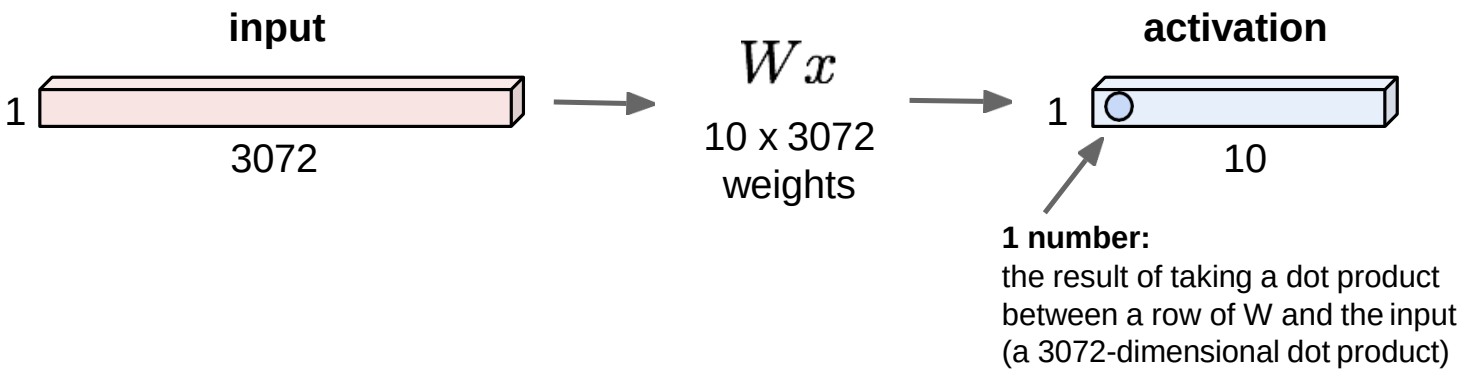


- **Problem**: This ignores spatial relationship between pixels in the image
  - Images contain local structures
    - Groups of neighboring pixels correspond to visual structures (edges, corners, texture)
    - Pixels far from each other are typically not correlated

From Fully Connected to Convolutional Layer

Fully Connected Layer

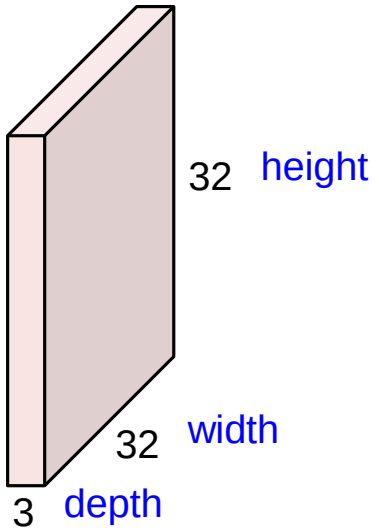
32x32x3 image -> stretch to 3072 x 1



From Fully Connected to Convolutional Layer

Convolution Layer

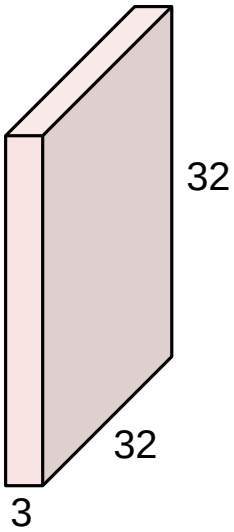
32x32x3 image -> preserve spatial structure



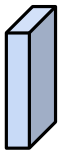
From Fully Connected to Convolutional Layer

Convolution Layer

32x32x3 image



5x5x3 filter



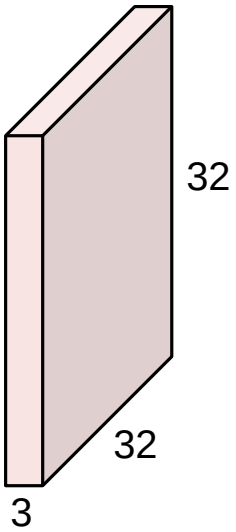
Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

From Fully Connected to Convolutional Layer

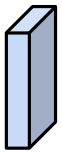
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



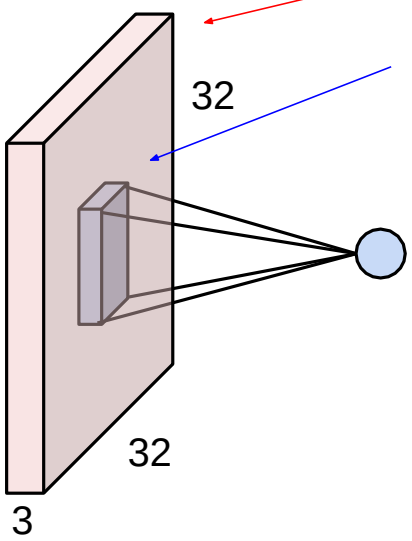
5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

From Fully Connected to Convolutional Layer

Convolution Layer



32x32x3 image

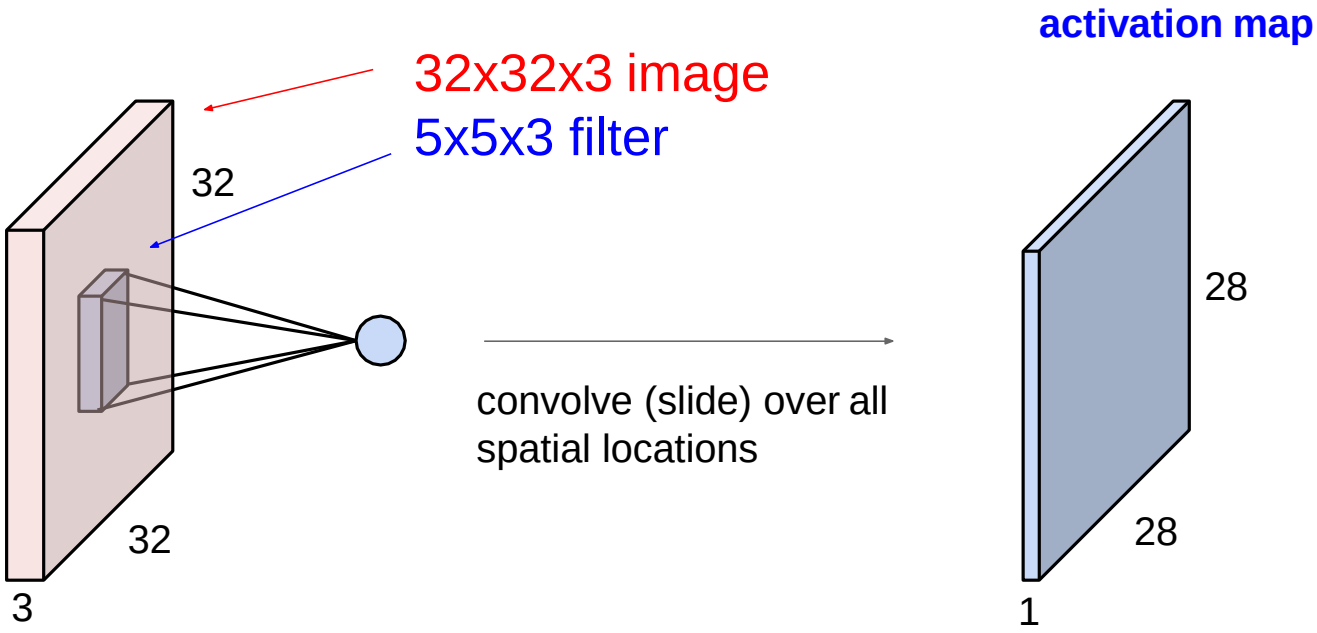
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

From Fully Connected to Convolutional Layer

Convolution Layer



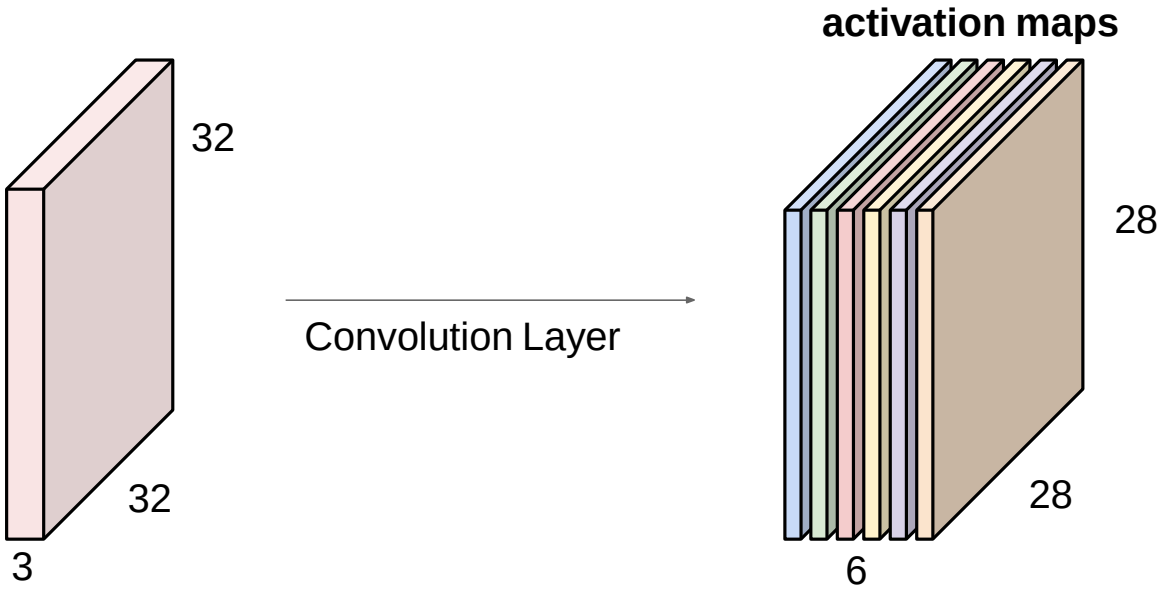
From Fully Connected to Convolutional Layer

Convolution Layer

consider a second, green filter

From Fully Connected to Convolutional Layer

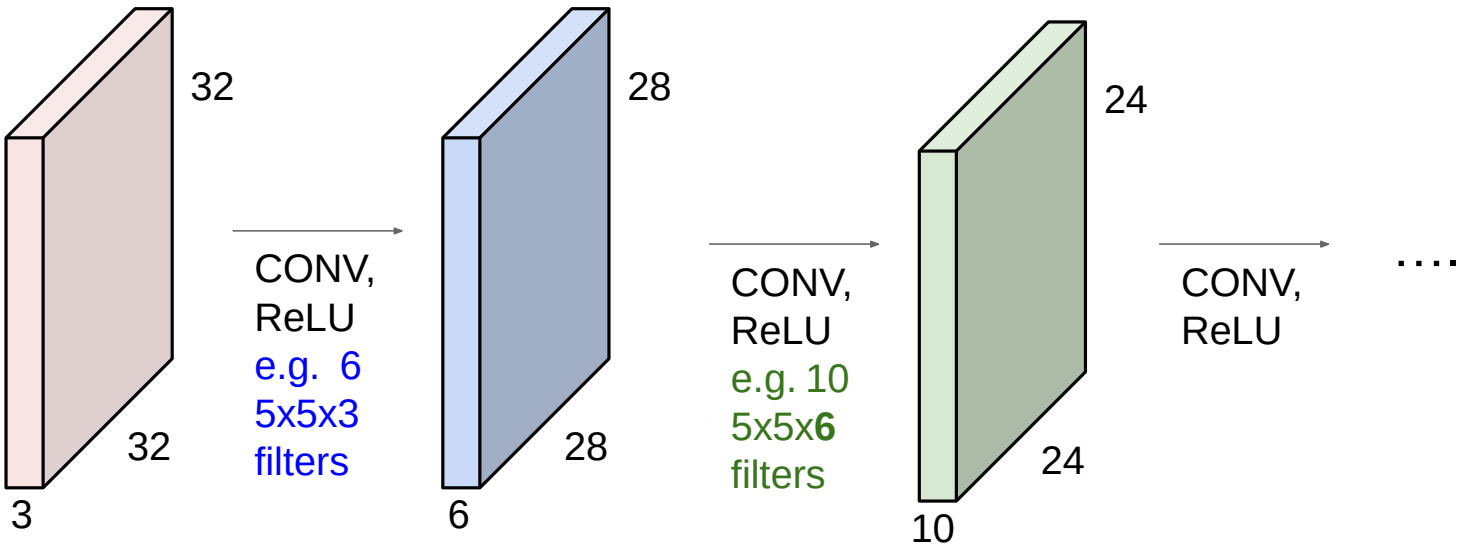
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

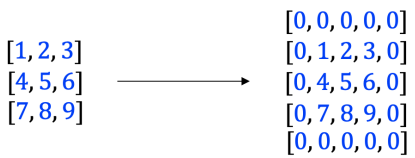
Convolutional Neural Networks

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Padding

- Valid padding
  - No padding is added to the input image, and the output image is smaller than the input image
- Same padding
  - Padding is added to the input image such that the size of the output image is the same as the input image
- Full padding
  - Padding is added to the input image such that the size of the output image is greater than the input image
- Constant padding (including zero padding)
  - Add a border of constant-value pixels around the edges of the original image



Padding Cont.

- Replicate padding
  - The pixels of the padding are copied from the border values

$$\begin{bmatrix}1,2,3\\4,5,6\\7,8,9\end{bmatrix} \longrightarrow \begin{bmatrix}1,1,2,3,3\\1,1,2,3,3\\4,4,5,6,6\\7,7,8,9,9\\7,7,8,9,9\end{bmatrix}$$

- Reflection padding
  - The pixels at the edges of the image are mirrored to create a boundary of reflected pixels

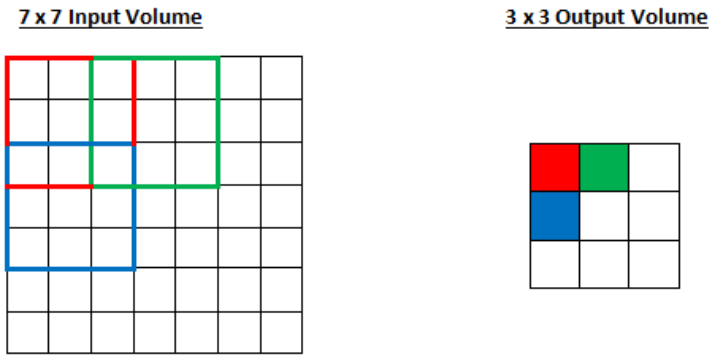
$$\begin{bmatrix}1,2,3\\4,5,6\\7,8,9\end{bmatrix} \longrightarrow \begin{bmatrix}5,4,5,6,5\\2,1,2,3,2\\5,4,5,6,5\\8,7,8,9,8\\5,4,5,6,5\end{bmatrix}$$

- Circular padding
  - Copy the pixels at the edges of the image and append them to the opposite side of the image

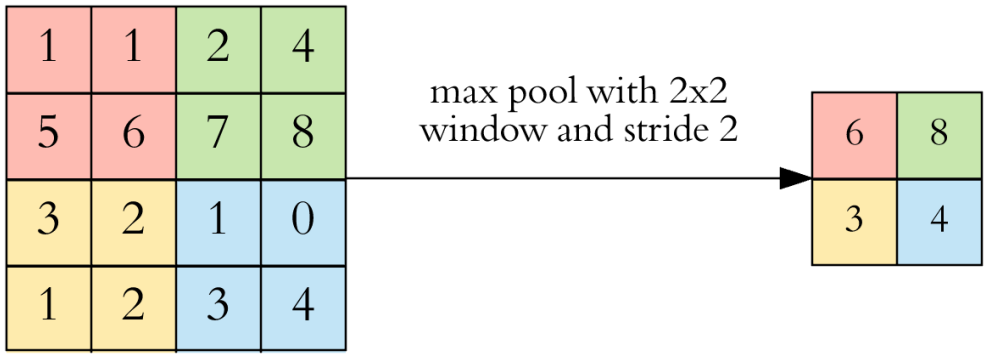
$$\begin{bmatrix}1,2,3\\4,5,6\\7,8,9\end{bmatrix} \longrightarrow \begin{bmatrix}9,7,8,9,7\\3,1,2,3,1\\6,4,5,6,4\\9,7,8,9,7\\3,1,2,3,1\end{bmatrix}$$

Spatial Sub-Sampling

- Reduce the feature map size by subsampling feature maps
  - *Stride* for convolution filter - step size when moving the windows across the image

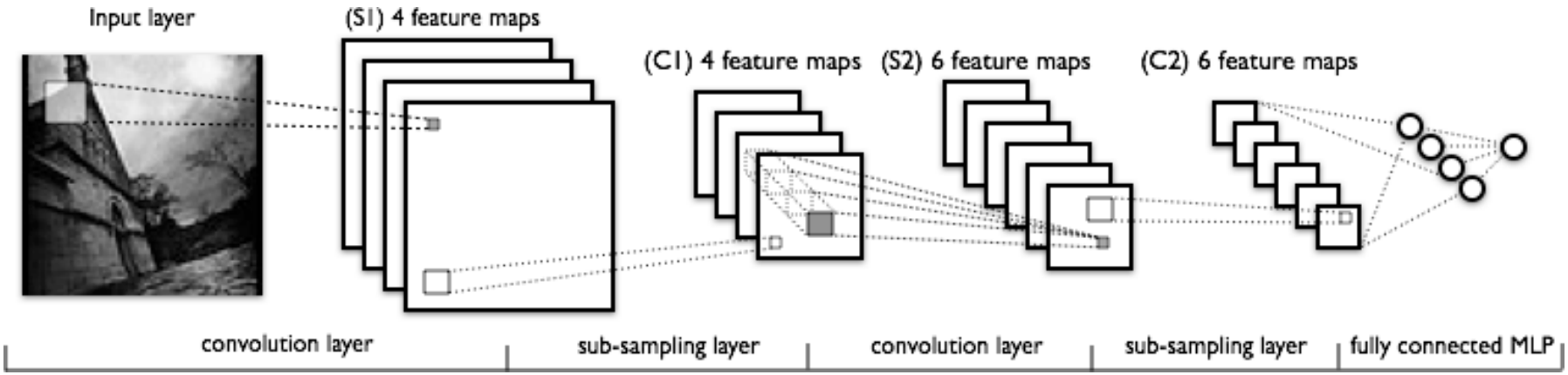


  - Use the maximum over the pooling window



Adding back MLP

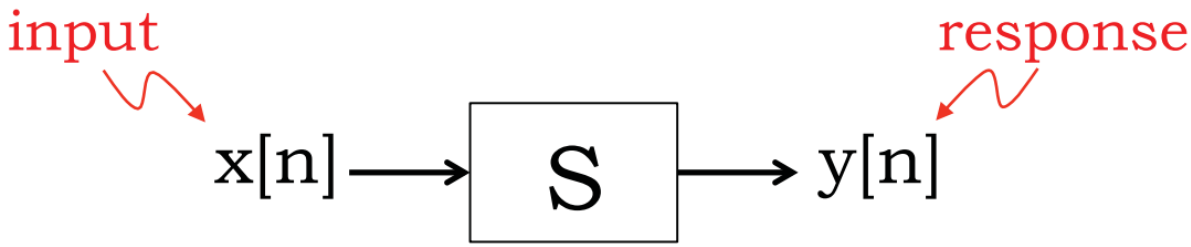
- After several convolutional layers, input the feature map into an MLP to get the final classification



Outline

Where Does Convolution Come From?

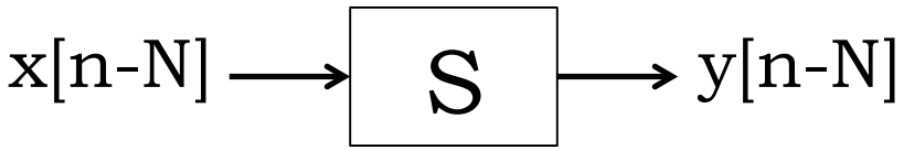
- Convolution arises from the field of signal processing, which describes the output (in terms of the input) of an important class of operations (or systems) known as *linear time-invariant* (LTI)



- A discrete-time signal such as $x[n]$ or $y[n]$ is described by an infinite sequence of values, *i.e.*, the time index n takes values in $-\infty$ to ∞. The above picture is a snapshot at a particular time n
- The sequence of output values $y[\cdot]$ is the response of system S to the input sequence $x[\cdot]$

Time Invariant Systems

- Let $y[n]$ be the response of S to input $x[n]$
- If for all possible sequences $x[n]$ and integers N



then system S is said to be time invariant (TI). A time shift in the input sequence to S results in an identical time shift of the output sequence

Linear Systems

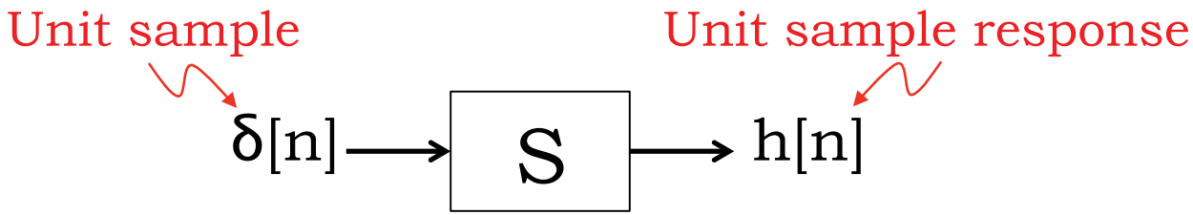
- Let $y_1[n]$ be the response of S to an arbitrary input $x_1[n]$ and $y_2[n]$ be the response to an arbitrary $x_2[n]$
- If, for arbitrary scalar coefficients a and b, we have

$$ax_1[n]+bx_2[n] \longrightarrow \boxed{\text{S}} \longrightarrow ay_1[n]+by_2[n]$$

then system S is said to be linear. If the input is the weighted sum of several signals, the response is the superposition (*i.e.*, the same weighted sum) of the response to those signals

- One key consequence: If the input is identically 0 for a linear system, the output must also be identically 0

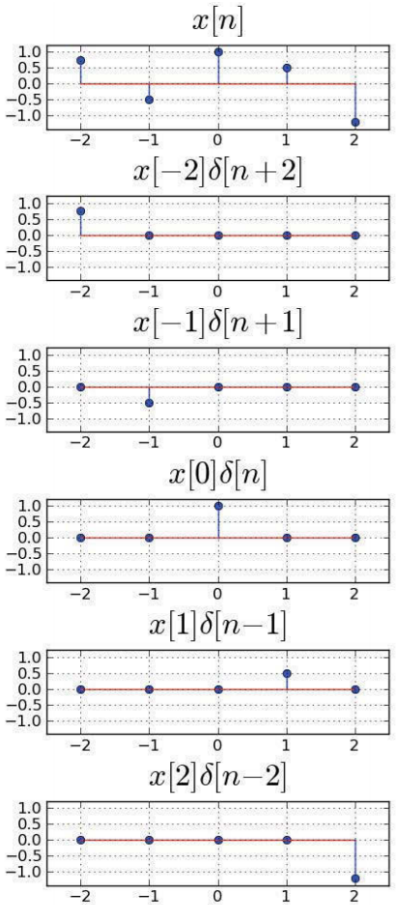
Unit Sample Responses



$$\delta[n] = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad \delta[n-N] = \begin{cases} 1, & \text{if } n = N \\ 0, & \text{otherwise} \end{cases}$$

- The unit sample response of a system S is the response of the system to the unit sample input. We will always denote the unit sample response as $h[n]$

Unit Sample Decomposition



- A discrete-time signal can be decomposed into a sum of time-shifted, scaled unit samples
- Example: in the figure, $x[n]$ is the sum of $x[-2]\delta[n+2] + x[-1]\delta[n+1] + \ldots + x[2]\delta[n-2]$
- In general,

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

For any particular index, only one term of this sum is non-zero

Modeling LTI Systems

- If system S is both linear and time-invariant (LTI), then we can use the unit sample response to predict the response to any input waveform $x[n]$

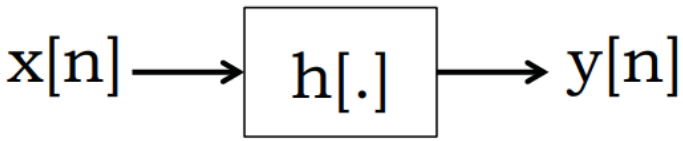
Sum of shifted, scaled unit samples

Sum of shifted, scaled responses

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \longrightarrow \boxed{S} \longrightarrow y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

CONVOLUTION SUM

- Indeed, the unit sample response $h[n]$ completely characterizes the LTI system S, so you often see



Discrete Convolution

- Discrete convolution typically contains the following steps:
  1. List the index 'k' covering a sufficient range
  2. List the input $x[k]$
  3. Obtain the **reversed** sequence $h[-k]$, and align the rightmost element of $h[n-k]$ to the leftmost element of $x[k]$
  4. Cross-multiply and sum (*i.e.*, dot product) the nonzero overlap terms to produce $y[n]$
  5. Slide $h[n-k]$ to the right by one position
  6. Repeat Steps 4 and 5; stop if all the output values are zero or if required

Example

Example: Find the convolution of the two sequences x[n] and h[n] given by,

x[k] = [3 1 2] h[k] = [3 2 1]

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

Hint: The value of k starts from **(– length of h + 1)** and continues till **(length of h + length of x – 1)**

Here ***k*** starts from **-3 + 1 = -2** and continues till **3 + 3 – 1 = 5**

Example

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

$y[0] = 3 \times 3 = 9$

Example

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

y[0] = 3 x 3 = 9

y[1] = 3 x 2 + 3 x 1 = 9

Example

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

y[0] = 3 x 3 = 9

y[1] = 3 x 2 + 3 x 1 = 9

y[2] = 3 x 1 + 1 x 2 + 2 x 3= 11

Example

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

y[0] = 3 x 3 = 9

y[1] = 3 x 2 + 3 x 1 = 9

y[2] = 3 x 1 + 1 x 2 + 2 x 3= 11

y[3] = 1 x 1 + 2 x 2 = 5

Example

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

y[0] = 3 x 3 = 9

y[1] = 3 x 2 + 3 x 1 = 9

y[2] = 3 x 1 + 1 x 2 + 2 x 3= 11

y[3] = 1 x 1 + 2 x 2 = 5

y[4] = 2 x 1 = 2

Example

k:	-2	-1	0	1	2	3	4	5
x[k]:			3	1	2			
h[-k]:	1	2	3					
h[1-k]:		1	2	3				
h[2-k]:			1	2	3			
h[3-k]:				1	2	3		
h[4-k]:					1	2	3	
h[5-k]:						1	2	3

y[0] = 3 x 3 = 9

y[1] = 3 x 2 + 3 x 1 = 9

y[2] = 3 x 1 + 1 x 2 + 2 x 3= 11

y[3] = 1 x 1 + 2 x 2 = 5

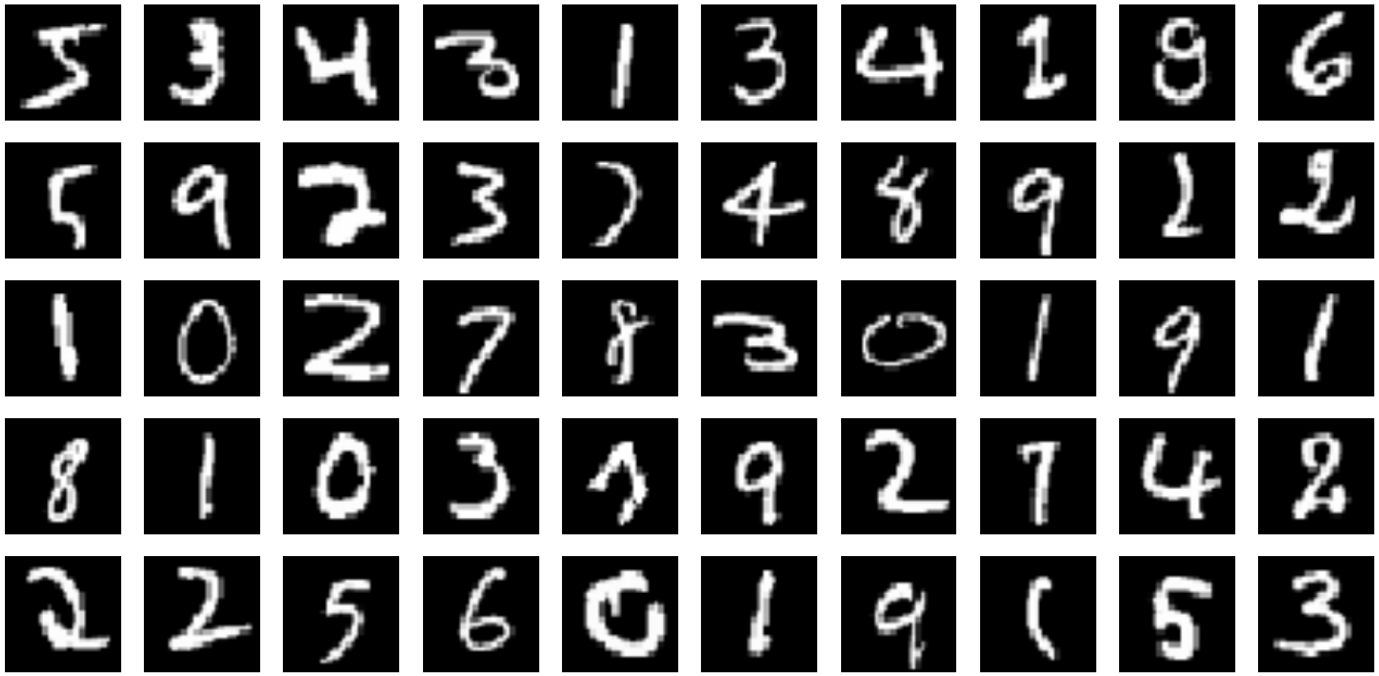
y[4] = 2 x 1 = 2

y[5] = 0 (no overlap)

Outline

Example on MNIST Dataset

- Images are 28×28, digits 0-9
  - $60,000$ for training
  - $10,000$ for testing

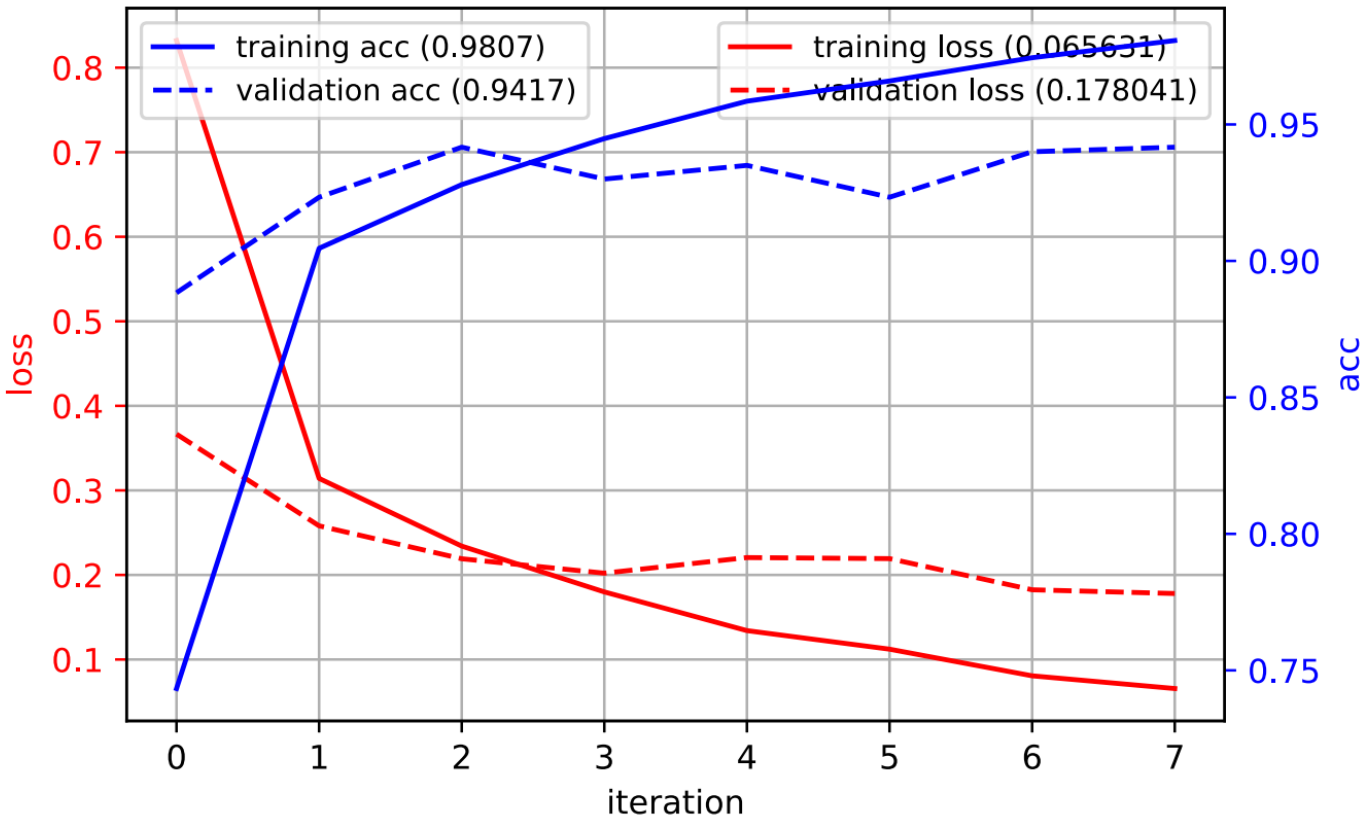


Example on MNIST Dataset

- Pre-processing
  - Scale to $[0, 1]$
  - 4-D data: (sample, channel, height, width)
    - Channel = 1 (grayscale)
  - Create traning/validation sets
- Shallow CNN architecture as a baseline ($98,820$ parameters)
  - 1 convolutional layer
    - $1 \times 5 \times 5$, 10 feature maps
    - stride, 2 (step size between sliding windows)
    - No pooling here since the input image is small (28×28)
  - 1 fully-connected layer, 50 nodes
    - Input: $10 \times 14 \times 14 = 1,960$ –> Output: 50
  - Softmax with 10 nodes as the final layer

Example on MNIST Dataset

- Test accuracy of the shallow CNN: 0.9449

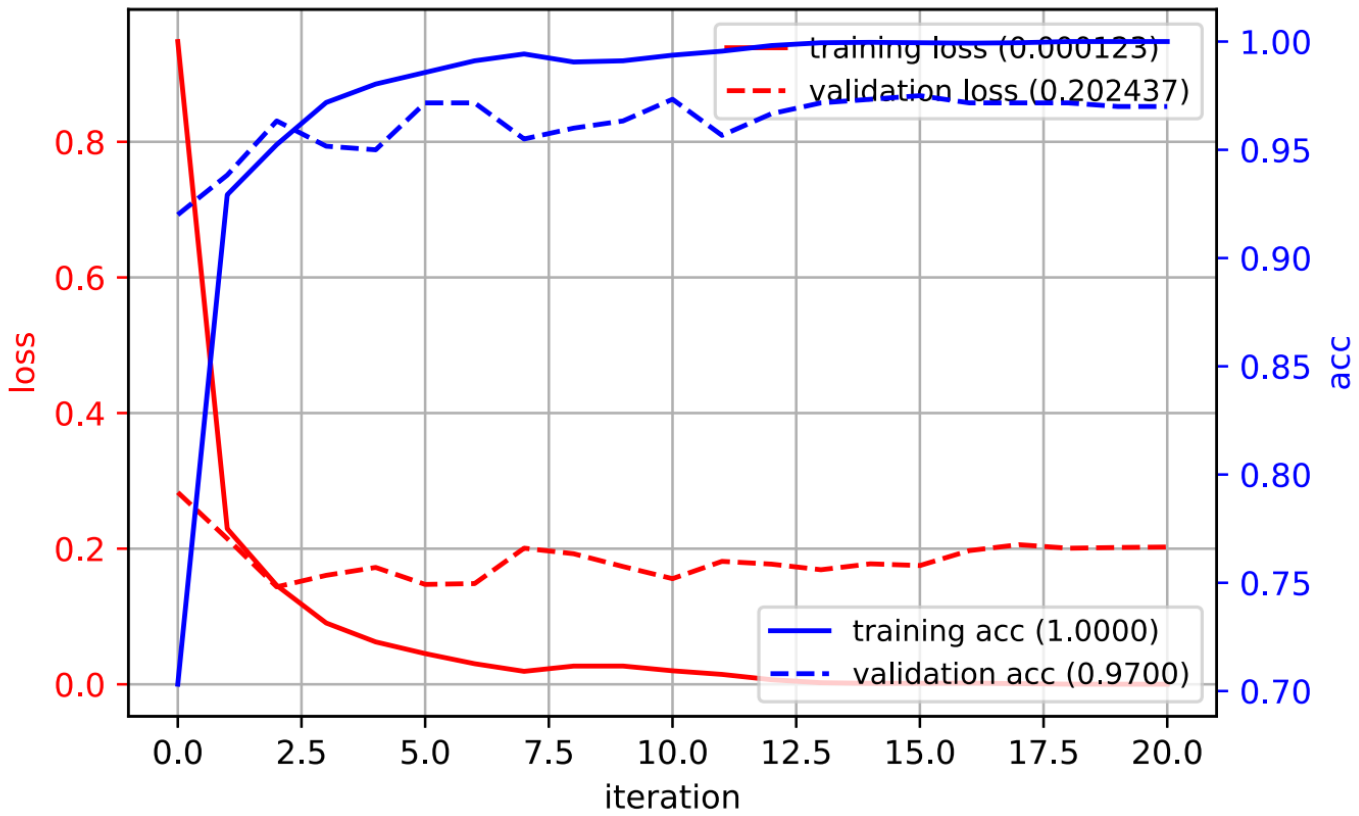


Example on MNIST Dataset

- Deep CNN architecture (229, 340 parameters)
  - 3 convolutional layers
    - $1 \times 5 \times 5$, stride 2, 10 feature maps ($10 \times 14 \times 14$)
    - $10 \times 3 \times 3$, stride 2, 40 feature maps ($40 \times 7 \times 7$)
    - $40 \times 3 \times 3$, stride 1, 80 feature maps ($80 \times 7 \times 7$)
  - 1 fully-connected layer (MLP), 50 nodes
    - Input: $80 \times 7 \times 7 = 3920$ –> Output: 50
  - Softmax with 10 nodes as the final layer

Example on MNIST Dataset

- Test accuracy of the deep CNN: 0.9687



Summary

- Different types of neural networks
  - Perceptron - single node (similar to logistic regression)
  - MLP - collection of perceptrons in layers
    - Also called fully connected network
  - CNN - convolution filters for extracting local image features
    - Originates from LTI systems in signal processing
- **Training**
  - Optimization using *variants* of stochastic gradient descent
- **Advantages**
  - Large capacity to learn from large amounts of data
- **Disadvantages**
  - Lots of parameters - easy to overfit data
    - Need to *regularize* parameters
    - Need to monitor the training process
  - Sensitive to initialization, learning rate, training algorithm