Introduction to Machine Learning

Convolutional Networks

Andres Mendez-Vazquez

August 22, 2020

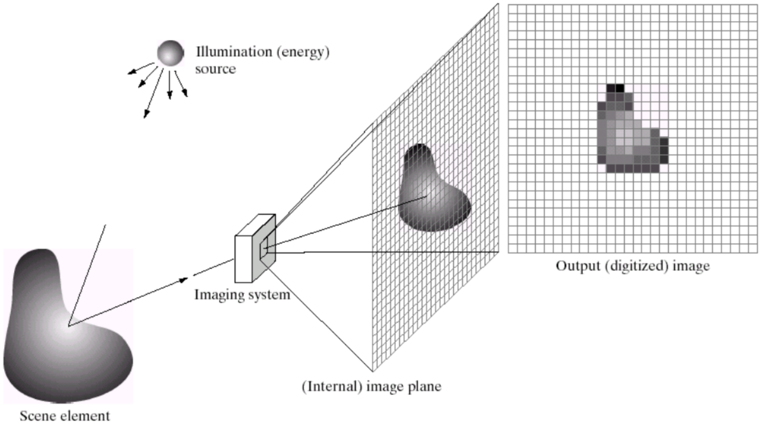
Outline

Cinvestav

Outline

Cinvestav

Digital Images as pixels in a digitized matrix [1]



Further [1]

Pixel values typically represent

- Gray levels, colors, heights, opacities etc

Something Notable

- Remember digitization implies that a digital image is an approximation of a real scene

Further [1]

Pixel values typically represent

- Gray levels, colors, heights, opacities etc

Something Notable

- Remember digitization implies that a digital image is an approximation of a real scene

Images

Common image formats include

- On sample/pixel per point (B&W or Grayscale)
- Three samples/pixel per point (Red, Green, and Blue)
- Four samples/pixel per point (Red, Green, Blue, and "Alpha")

Therefore, we have the following process

Low Level Process

Input	Processes	Output
Image	Noise Removal	Improved Image
	Image Sharpening	

Example

Edge Detection

Example

Edge Detection

Then

Mid Level Process

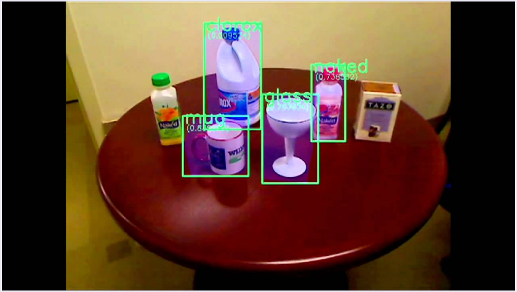
Input	Processes	Output
Image	Object Recognition Segmentation	Attributes

Example

Object Recognition

Example

Object Recognition



Therefore

It would be nice to automatize all these processes

- We would solve a lot of headaches when setting up such process

Why not to use the data sets

- By using a Neural Networks that replicates the process.

Therefore

It would be nice to automatize all these processes

- We would solve a lot of headaches when setting up such process

Why not to use the data sets

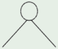
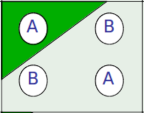
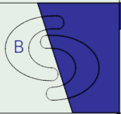
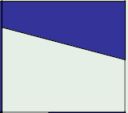
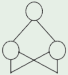
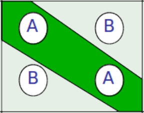
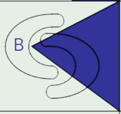
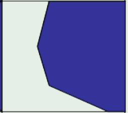
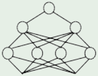
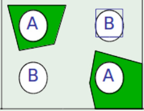
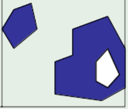
- By using a Neural Networks that replicates the process.

Outline

Cinvestav

Multilayer Neural Network Classification

We have the following classification [2]

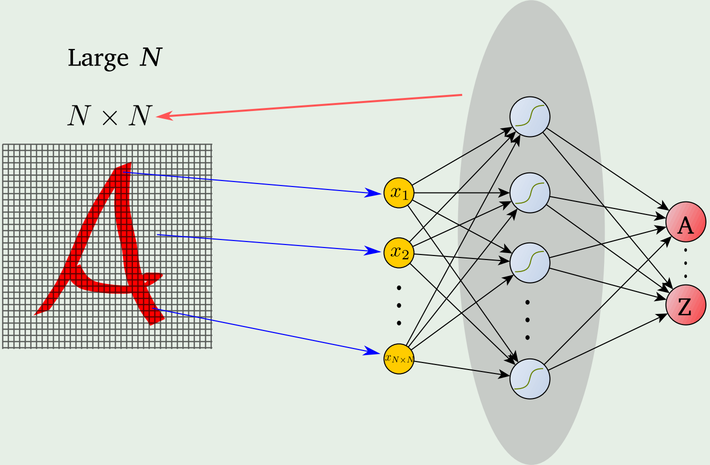
Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyper plane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Outline

Cinvestav

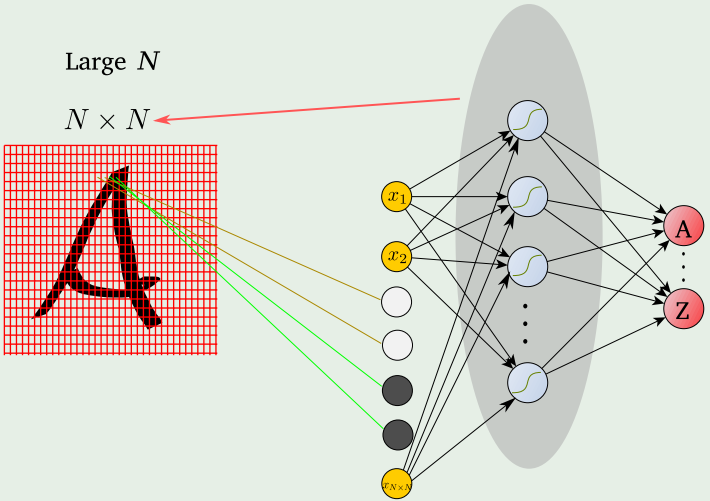
Drawbacks of previous neural networks

The number of trainable parameters becomes extremely large



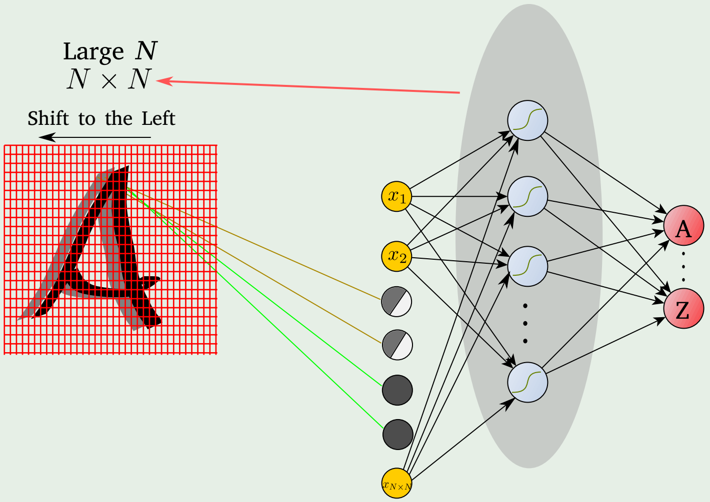
Drawbacks of previous neural networks

In addition, little or no invariance to shifting, scaling, and other forms of distortion



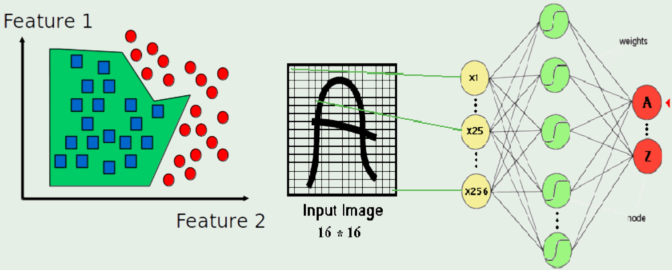
Drawbacks of previous neural networks

In addition, little or no invariance to shifting, scaling, and other forms of distortion



Drawbacks of previous neural networks

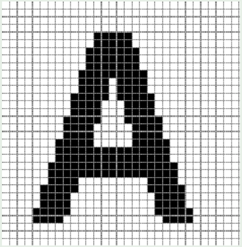
The topology of the input data is completely ignored



For Example

We have

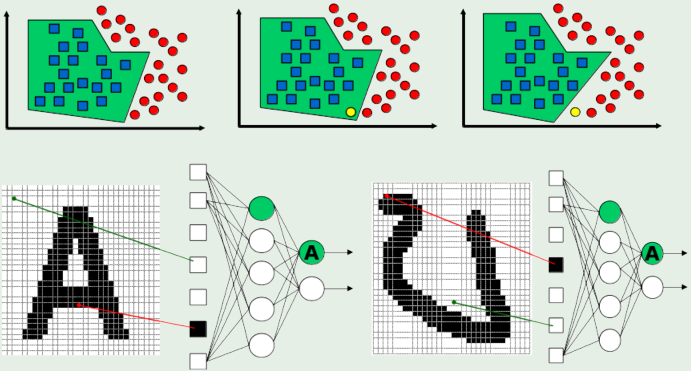
- Black and white patterns: $2^{32\times 32} = 2^{1024}$
- Gray scale patterns: $256^{32\times 32} = 256^{1024}$



32 * 32 input image

For Example

If we have an element that the network has never seen



Possible Solution

We can minimize this drawbacks by getting

Fully connected network of sufficient size can produce outputs that are invariant with respect to such variations.

Problem!!!

- Training time
- Network size
- Free parameters

Possible Solution

We can minimize this drawbacks by getting

Fully connected network of sufficient size can produce outputs that are invariant with respect to such variations.

Problem!!!

- Training time
- Network size
- Free parameters

Outline

Cinvestav

Hubel/Wiesel Architecture

Something Notable [3]

D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)

They commented

The visual cortex consists of a hierarchy of simple, complex, and hyper-complex cells

Hubel/Wiesel Architecture

Something Notable [3]

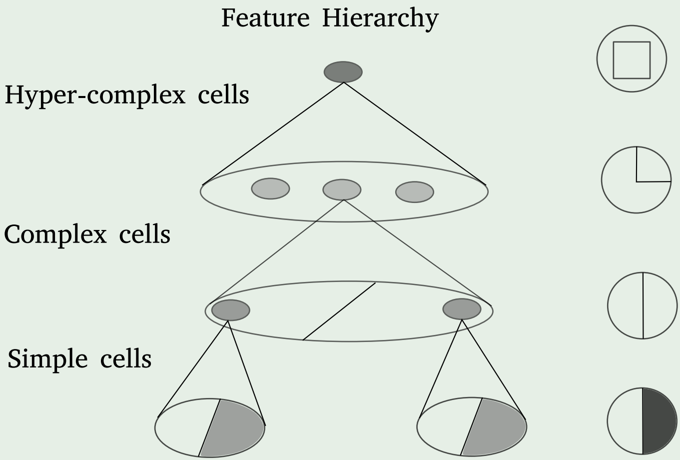
D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)

They commented

The visual cortex consists of a hierarchy of simple, complex, and hyper-complex cells

Something Like

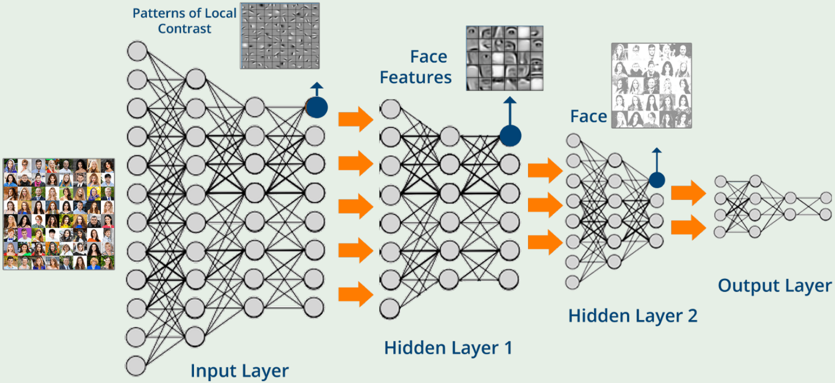
We have



History

Convolutional Neural Networks (CNN) were invented by [4]

In 1989, Yann LeCun and Yoshua Bengio introduced the concept of Convolutional Neural networks.



About CNN's

Something Notable

CNN's Were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.

In addition

They designed a network structure that implicitly extracts relevant features.

Properties

Convolutional Neural Networks are a special kind of multilayer neural networks.

About CNN's

Something Notable

CNN's Were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.

In addition

They designed a network structure that implicitly extracts relevant features.

Properties

Convolutional Neural Networks are a special kind of multilayer neural networks.

About CNN's

Something Notable

CNN's Were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.

In addition

They designed a network structure that implicitly extracts relevant features.

Properties

Convolutional Neural Networks are a special kind of multilayer neural networks.

About CNN's

In addition

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

About CNN's

In addition

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

Cinvestav

About CNN's

In addition

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

About CNN's

In addition

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability.

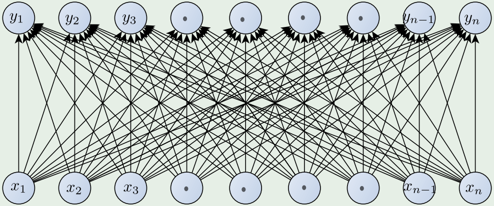
Outline

Cinvestav

Local Connectivity

We have the following idea [5]

- Instead of using a full connectivity...



Input Image

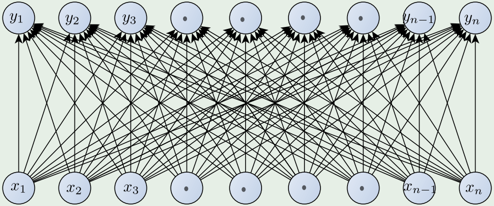
We would have something like this

$$y_i = f\left(\sum_{i=1}^{n} w_i x_i\right) \quad (1)$$

Local Connectivity

We have the following idea [5]

- Instead of using a full connectivity...



We would have something like this

$$y_i = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{1}$$

Local Connectivity

We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:
  - 1 if gray scale
  - 3 in the RGB case

Local Connectivity

We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:
  - 1 if gray scale
  - 3 in the RGB case

Local Connectivity

We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:
  - 1 if gray scale
  - 3 in the RGB case

Local Connectivity

We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:
  - 1 if gray scale
  - 3 in the RGB case

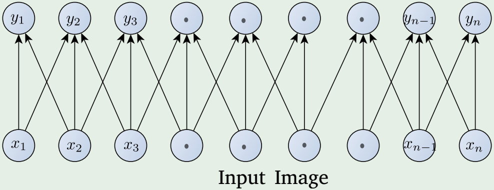
Local Connectivity

We decide only to connect the neurons in a local way

- Each hidden unit is connected only to a subregion (patch) of the input image.
- It is connected to all channels:
  - 1 if gray scale
  - 3 in the RGB case

Example

For gray scale, we get something like this

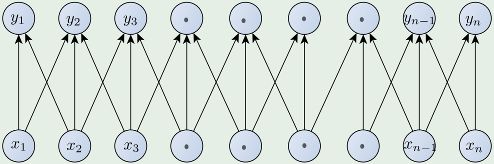


Then, our formula changes

$$y_i = f\left(\sum_{i \in L_p} w_i x_i\right) \tag{2}$$

Example

For gray scale, we get something like this



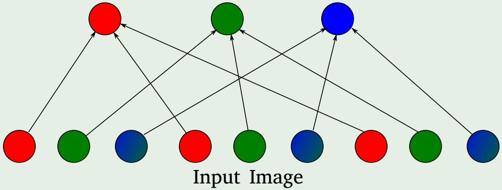
Input Image

Then, our formula changes

$$y_i = f\left(\sum_{i \in L_p} w_i x_i\right) \tag{2}$$

Example

In the case of the 3 channels

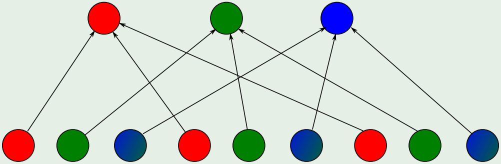


Thus

$$y_i = f\left(\sum_{i \in L_{p,c}} w_i x_i^c\right) \quad (3)$$

Example

In the case of the 3 channels



Thus

$$y_i = f\left(\sum_{i \in L_p, c} w_i x_i^c\right) \quad (3)$$

Solving the following problems...

First

- Fully connected hidden layer would have an unmanageable number of parameters

Second

- Computing the linear activation of the hidden units would have been quite expensive

Solving the following problems...

First

- Fully connected hidden layer would have an unmanageable number of parameters

Second

- Computing the linear activation of the hidden units would have been quite expensive

How this looks in the image...

We have

Receptive Field

Outline

Parameter Sharing

Second Idea

Share matrix of parameters across certain units.

These units are organized into

- The same feature "map"
  - Where the units share same parameters (For example, the same mask)

Parameter Sharing

Second Idea

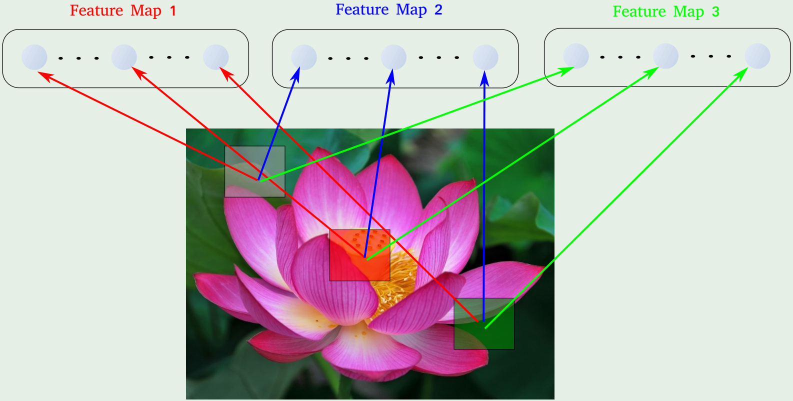
Share matrix of parameters across certain units.

These units are organized into

- The same feature "map"
  - Where the units share same parameters (For example, the same mask)

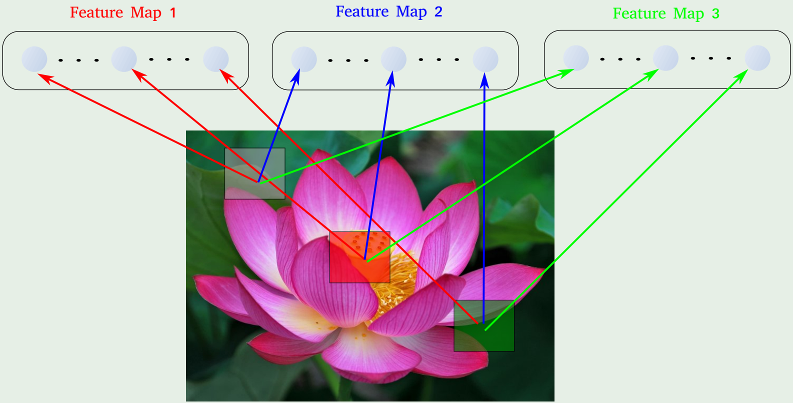
Example

We have something like this



Example

We have something like this



Now, in our notation

We have a collection of matrices representing this connectivity

- W_{ij} is the connection matrix the ith input channel with the jth feature map.
- In each cell of these matrices is the weight to be multiplied with the local input to the local neuron.

Now, in our notation

We have a collection of matrices representing this connectivity

- W_{ij} is the connection matrix the ith input channel with the jth feature map.
- In each cell of these matrices is the weight to be multiplied with the local input to the local neuron.

An now why the name of convolution

Yes!!! The definition is coming now.

Outline

Cinvestav

Digital Images

In computer vision [1, 6]

We usually operate on digital (discrete) images:

- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

Digital Images

In computer vision [1, 6]

We usually operate on digital (discrete) images:

- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

The image can now be represented as a matrix of integer values, $f : [a, b] \times [c, d] \rightarrow I$

$$\begin{array}{cc} & j \longrightarrow \\ i \downarrow & \begin{bmatrix} 79 & 5 & 6 & 90 & 12 & 34 & 2 & 1 \\ 8 & 90 & 12 & 34 & 26 & 78 & 34 & 5 \\ 8 & 1 & 3 & 90 & 12 & 34 & 11 & 61 \\ 77 & 90 & 12 & 34 & 200 & 2 & 9 & 45 \\ 1 & 3 & 90 & 12 & 20 & 1 & 6 & 23 \end{bmatrix} \end{array}$$

Digital Images

In computer vision [1, 6]

We usually operate on digital (discrete) images:

- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

The image can now be represented as a matrix of integer values, $f : [a, b] \times [c, d] \rightarrow I$

$$
\begin{array}{cc}
 & j \longrightarrow \\
i \downarrow & \left[\begin{array}{cccccccc}
79 & 5 & 6 & 90 & 12 & 34 & 2 & 1 \\
8 & 90 & 12 & 34 & 26 & 78 & 34 & 5 \\
8 & 1 & 3 & 90 & 12 & 34 & 11 & 61 \\
77 & 90 & 12 & 34 & 200 & 2 & 9 & 45 \\
1 & 3 & 90 & 12 & 20 & 1 & 6 & 23
\end{array}\right]
\end{array}
$$

Digital Images

In computer vision [1, 6]

We usually operate on digital (discrete) images:

- Sample the 2D space on a regular grid.
- Quantize each sample (round to nearest integer).

The image can now be represented as a matrix of integer values, $f : [a, b] \times [c, d] \rightarrow I$

$$
\begin{array}{cc}
 & j \longrightarrow \\
i \downarrow & \begin{bmatrix}
79 & 5 & 6 & 90 & 12 & 34 & 2 & 1 \\
8 & 90 & 12 & 34 & 26 & 78 & 34 & 5 \\
8 & 1 & 3 & 90 & 12 & 34 & 11 & 61 \\
77 & 90 & 12 & 34 & 200 & 2 & 9 & 45 \\
1 & 3 & 90 & 12 & 20 & 1 & 6 & 23
\end{bmatrix}
\end{array}
$$

We can see the coordinate of f as follows

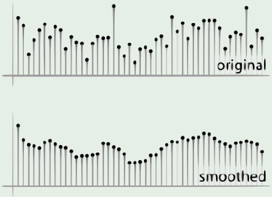
We have the following

$$f = \begin{pmatrix} f_{-n,-n} & f_{-n,-n+1} & \cdots & f_{-n,(n-1)} & f_{-n,n} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \vdots & \cdots & f_{0,0} & \cdots & \vdots \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ f_{n\times -n} & f_{n\times -n+1} & \cdots & f_{n\times(n-1)} & f_{n,n} \end{pmatrix} \tag{4}$$

Cinvestav

Many times we want to eliminate noise in a image

By using for example a moving average



This last moving average can be seen as

$$(f * g)(i) = \sum_{j=-n}^{n} f(j)\, g(i-j) = \frac{1}{N} \sum_{j=m}^{-m} f(j) \quad (5)$$

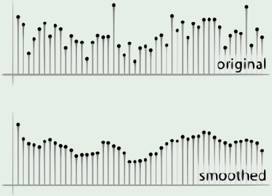
With $f(j)$ representing the value of the pixel at position i,

$$g(h) = \begin{cases} \frac{1}{N} & \text{if } h \in \{-m, -m+1, ..., 1, 0, 1, ..., m-1, m\} \\ 0 & \text{else} \end{cases}$$

with $0 < m < n$.

Many times we want to eliminate noise in a image

By using for example a moving average



This last moving average can be seen as

$$(f * g)(i) = \sum_{j=-n}^{n} f(j)\, g(i-j) = \frac{1}{N} \sum_{j=m}^{-m} f(j) \tag{5}$$

With $f(j)$ representing the value of the pixel at position i,

$$g(h) = \begin{cases} \frac{1}{N} & \text{if } h \in \{-m, -m+1, ..., 1, 0, 1, ..., m-1, m\} \\ 0 & \text{else} \end{cases}$$

with $0 < m < n$.

This can be generalized into the 2D images

Left f and Right $f * g$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0								

This can be generalized into the 2D images

Left f and Right $f * g$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10							

This can be generalized into the 2D images

Left f and Right $f * g$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20						

This can be generalized into the 2D images

Left f and Right $f * g$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Moving average in 2D

Basically in 2D

We have that we can define different types of filter using the idea of weighted average

$$(f * g)(i,j) = \sum_{k=n}^{-n} \sum_{l=-n}^{n} f(k,l) \times g(i-k, j-l) \tag{6}$$

What is this weight matrix also called a kernel of 3×3 moving average

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{"The Box Filter"} \tag{7}$$

Moving average in 2D

Basically in 2D

We have that we can define different types of filter using the idea of weighted average

$$(f * g)(i,j) = \sum_{k=n}^{-n} \sum_{l=-n}^{n} f(k,l) \times g(i-k, j-l) \tag{6}$$

What is this weight matrix also called a kernel of 3×3 moving average

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{"The Box Filter"} \tag{7}$$

Outline

Cinvestav

Convolution

Definition

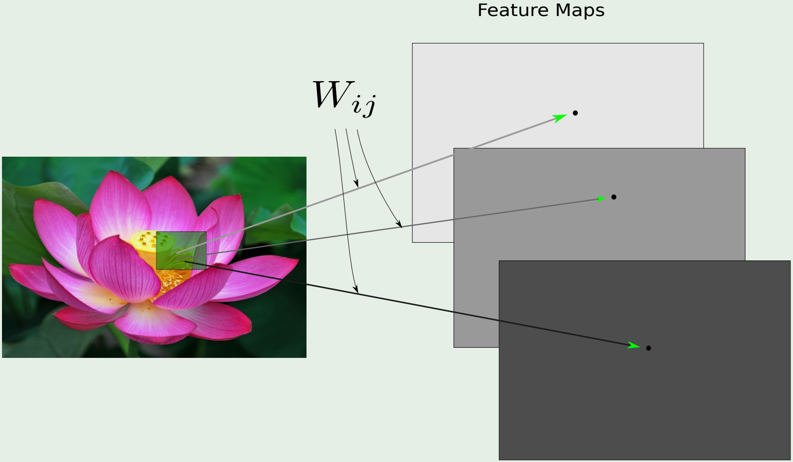
Let $f : [a, b] \times [c, d] \to I$ be the image and $g : [e, f] \times [h, i] \to V$ be the kernel. The output of convolving f with g, denoted $f * g$ is

$$(f * g)[x, y] = \sum_{k=-n}^{n} \sum_{l=-n}^{n} f(k, l)\, g(x - k, y - l) \tag{8}$$

- The Flipped Kernel

Back on the Convolutional Architecture

We have then something like this



Thus

Each Feature Map forms a 2D grid of features

That can be computed with a discrete convolution (*) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with rows and columns with its rows and columns flipped.

Thus

Each Feature Map forms a 2D grid of features

That can be computed with a discrete convolution (*) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with rows and columns with its rows and columns flipped.

In addition

- x_i is the ith channel of input.
- k_{ij} is the convolution kernel.
- y_j is the hidden layer output.

Thus

Each Feature Map forms a 2D grid of features

That can be computed with a discrete convolution (*) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with rows and columns with its rows and columns flipped.

In addition

- x_i is the ith channel of input.
- k_{ij} is the convolution kernel.
- y_j is the hidden layer output.

Thus the total output

$$y_j = \sum_i k_{ij} * x_i \tag{9}$$

Thus

Each Feature Map forms a 2D grid of features

That can be computed with a discrete convolution (*) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with rows and columns with its rows and columns flipped.

In addition

- x_i is the ith channel of input.
- k_{ij} is the convolution kernel.
- y_j is the hidden layer output.

Thus the total output

$$y_j = \sum_i k_{ij} * x_i \quad (9)$$

Thus

Each Feature Map forms a 2D grid of features

That can be computed with a discrete convolution (*) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with rows and columns with its rows and columns flipped.

In addition

- x_i is the ith channel of input.
- k_{ij} is the convolution kernel.
- y_j is the hidden layer output.

Thus the total output

$$y_j = \sum_i k_{ij} * x_i \tag{9}$$

Furthermore

Let layer l be a Convolutional Layer

Then, the input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer.

Each input layer has a size of $m_2^{(l-1)} \times m_3^{(l-1)}$

In the case where $l = 1$, the input is a single image I consisting of one or more channels.

Thus

The output of layer l consists of $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Cinvestav

Furthermore

Let layer l be a Convolutional Layer

Then, the input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer.

Each input layer has a size of $m_2^{(l-1)} \times m_3^{(l-1)}$

In the case where $l = 1$, the input is a single image I consisting of one or more channels.

Thus

The output of layer l consists of $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Cinvestav

Furthermore

Let layer l be a Convolutional Layer

Then, the input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer.

Each input layer has a size of $m_2^{(l-1)} \times m_3^{(l-1)}$

In the case where $l = 1$, the input is a single image I consisting of one or more channels.

Thus

The output of layer l consists of $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Cinvestav

Remark

We have that

- A Convolutional Neural Network (CNN) directly accepts raw images as input.

Thus, their importance when training discrete filters

- Instead of assuming a certain comprehension of Computer Vision, one could think this is as a Silver Bullet.

However, you still

- You still need to be aware of :
  - The need of great quantities of data.
  - And there is not an understanding why this work.

Cinvestav

Remark

We have that

- A Convolutional Neural Network (CNN) directly accepts raw images as input.

Thus, their importance when training discrete filters

- Instead of assuming a certain comprehension of Computer Vision, one could think this is as a Silver Bullet.

However, you still

- You still need to be aware of :
  - The need of great quantities of data.
  - And there is not an understanding why this work.

Cinvestav

Remark

We have that

- A Convolutional Neural Network (CNN) directly accepts raw images as input.

Thus, their importance when training discrete filters

- Instead of assuming a certain comprehension of Computer Vision, one could think this is as a Silver Bullet.

However, you still

- You still need to be aware of :
  - The need of great quantities of data.
  - And there is not an understanding why this work.

Another Remark

We have the following

- $Y_j^{(l)}$ is a matrix representing the l layer and j^{th} feature map.

Therefore

- We can see the convolutional as a fusion of information from different feature maps.

$$\sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)}$$

Cinvestav

Another Remark

We have the following

- $Y_j^{(l)}$ is a matrix representing the l layer and j^{th} feature map.

Therefore

- We can see the convolutional as a fusion of information from different feature maps.

$$\sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)}$$

Cinvestav

Thus

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)} \tag{10}$$

Thus

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)} \tag{10}$$

Where

- $Y_i^{(l)}$ is the i^{th} feature map in layer l.
- $B_i^{(l)}$ is the bias matrix for output j.
- $K_{ij}^{(l)}$ is the filter of size $\left[2h_1^{(l)}+1\right] \times \left[2h_2^{(l)}+1\right]$.

Thus

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)} \quad (10)$$

Where

- $Y_i^{(l)}$ is the i^{th} feature map in layer l.
- $B_i^{(l)}$ is the bias matrix for output j.
- $K_{ij}^{(l)}$ is the filter of size $\left[2h_1^{(l)}+1\right] \times \left[2h_2^{(l)}+1\right]$.

Thus

The input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$

Thus

Given a specific layer l, we have that i^{th} feature map in such layer equal to

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)} \tag{10}$$

Where

- $Y_i^{(l)}$ is the i^{th} feature map in layer l.
- $B_i^{(l)}$ is the bias matrix for output j.
- $K_{ij}^{(l)}$ is the filter of size $\left[2h_1^{(l)} + 1\right] \times \left[2h_2^{(l)} + 1\right]$.

Thus

The input of layer l comprises $m_1^{(l-1)}$ feature maps from the previous layer, each of size $m_2^{(l-1)} \times m_3^{(l-1)}$

Therefore

Thew output of layer l

- It consists $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$

Something Notable

- $m_2^{(l)}$ and $m_3^{(l)}$ are influenced by border effects.
- Therefore, the output feature maps when the convolutional sum is defined properly have size

$$m_2^{(l)} = m_2^{(l-1)} - 2h_1^{(l)}$$

$$m_3^{(l)} = m_3^{(l-1)} - 2h_2^{(l)}$$

Therefore

Thew output of layer l

- It consists $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$

Something Notable

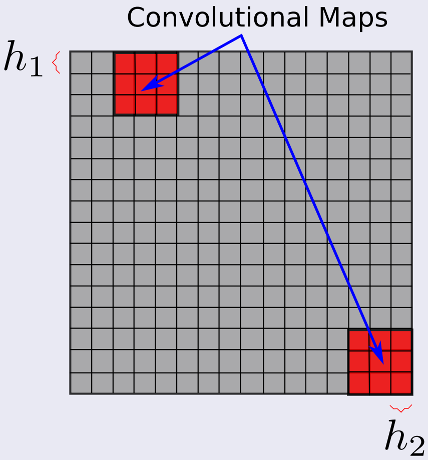
- $m_2^{(l)}$ and $m_3^{(l)}$ are influenced by border effects.
- Therefore, the output feature maps when the convolutional sum is defined properly have size

$$
\begin{aligned}
m_2^{(l)} &= m_2^{(l-1)} - 2h_1^{(l)} \\
m_3^{(l)} &= m_3^{(l-1)} - 2h_2^{(l)}
\end{aligned}
$$

Cinvestav

Why?

Example



Special Case

When $l = 1$

The input is a single image I consisting of one or more channels.

Thus

We have

Each feature map $Y_i^{(l)}$ in layer l consists of $m_1^{(l)} \cdot m_2^{(l)}$ units arranged in a two dimensional array.

Thus, the unit at position (r, s) computes

$$
\begin{aligned}
\left(Y_i^{(l)}\right)_{r,s} &= \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \left(K_{ij}^{(l)} * Y_j^{(l-1)}\right)_{r,s} \\
&= \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{ij}^{(l)}\right)_{k,t} \left(Y_j^{(l-1)}\right)_{r+k,s+t}
\end{aligned}
$$

Thus

We have

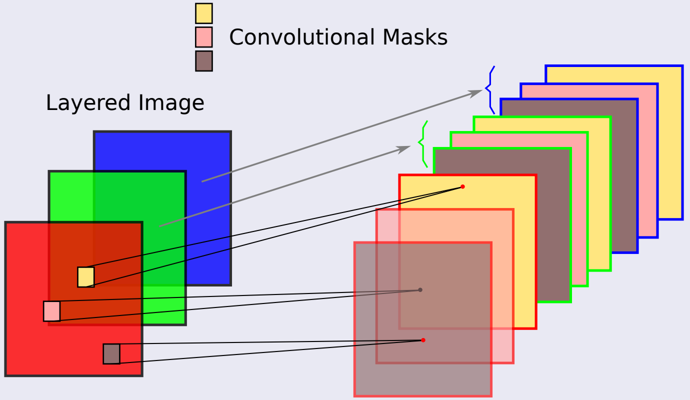
Each feature map $Y_i^{(l)}$ in layer l consists of $m_1^{(l)} \cdot m_2^{(l)}$ units arranged in a two dimensional array.

Thus, the unit at position (r, s) computes

$$\begin{aligned} \left(Y_i^{(l)}\right)_{r,s} &= \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \left(K_{ij}^{(l)} * Y_j^{(l-1)}\right)_{r,s} \\ &= \left(B_i^{(l)}\right)_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{ij}^{(l)}\right)_{k,t} \left(Y_j^{(l-1)}\right)_{r+k,s+t} \end{aligned}$$

Example

A Convolutional Layer against a RGB Image using three masks/filters



Outline

Cinvestav

As in Multilayer Perceptron

We use a non-linearity

- However, there is a drawback when using Back-Propagation under a sigmoid function

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

Because if we imagine a Convolutional Network as a series of layer functions f_i

$$y(A) = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1(A)$$

With f_t is the last layer.

Therefore, we finish with a sequence of derivatives

$$\frac{\partial y(A)}{\partial w_{1i}} = \frac{\partial f_t(f_{t-1})}{\partial f_{t-1}} \cdot \frac{\partial f_{t-1}(f_{t-2})}{\partial f_{t-2}} \cdot \cdots \cdot \frac{\partial f_2(f_1)}{\partial f_2} \cdot \frac{\partial f_1(A)}{\partial w_{1i}}$$

As in Multilayer Perceptron

We use a non-linearity

- However, there is a drawback when using Back-Propagation under a sigmoid function

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

Because if we imagine a Convolutional Network as a series of layer functions f_i

$$y(A) = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1(A)$$

With f_t is the last layer.

Therefore, we finish with a sequence of derivatives

$$\frac{\partial y(A)}{\partial w_{1i}} = \frac{\partial f_t(f_{t-1})}{\partial f_{t-1}} \cdot \frac{\partial f_{t-1}(f_{t-2})}{\partial f_{t-2}} \cdot \cdots \cdot \frac{\partial f_2(f_1)}{\partial f_2} \cdot \frac{\partial f_1(A)}{\partial w_{1i}}$$

As in Multilayer Perceptron

We use a non-linearity

- However, there is a drawback when using Back-Propagation under a sigmoid function

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

Because if we imagine a Convolutional Network as a series of layer functions f_i

$$y(A) = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1(A)$$

With f_t is the last layer.

Therefore, we finish with a sequence of derivatives

$$\frac{\partial y(A)}{\partial w_{1i}} = \frac{\partial f_t(f_{t-1})}{\partial f_{t-1}} \cdot \frac{\partial f_{t-1}(f_{t-2})}{\partial f_{t-2}} \cdot \cdots \cdot \frac{\partial f_2(f_1)}{\partial f_2} \cdot \frac{\partial f_1(A)}{\partial w_{1i}}$$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

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

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{\left(1 + e^{-x}\right)^2} + \frac{2\left(e^{-x}\right)^2}{\left(1 + e^{-x}\right)^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

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

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{\left(1+e^{-x}\right)^2} + \frac{2\left(e^{-x}\right)^2}{\left(1+e^{-x}\right)^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

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

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{(1+e^{-x})^2} + \frac{2(e^{-x})^2}{(1+e^{-x})^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Convolutional Network

- We could finish with

$$\lim_{k \to \infty} \left(\frac{ds(x)}{dx} \right)^k = \lim_{k \to \infty} (0.25)^k \to 0$$

A vanishing derivative

- Making quite difficult to do train a deeper network using this activation function

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Convolutional Network

- We could finish with

$$\lim_{k \to \infty} \left(\frac{ds(x)}{dx} \right)^k = \lim_{k \to \infty} (0.25)^k \to 0$$

A vanishing derivative

- Making quite difficult to do train a deeper network using this activation function

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Convolutional Network

- We could finish with

$$\lim_{k\to\infty}\left(\frac{ds(x)}{dx}\right)^k = \lim_{k\to\infty}(0.25)^k \to 0$$

A vanishing derivative

- Making quite difficult to do train a deeper network using this activation function

Thus

The need to introduce a new function

$$f(x) = x^{+} = \max(0, x)$$

It is called ReLu or Rectifier

With a smooth approximation (Softplus function)

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

Thus

The need to introduce a new function

$$f(x) = x^{+} = \max(0, x)$$

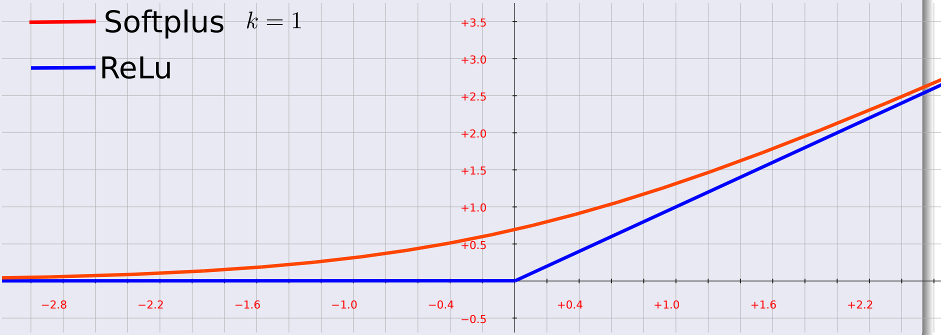
It is called ReLu or Rectifier

With a smooth approximation (Softplus function)

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

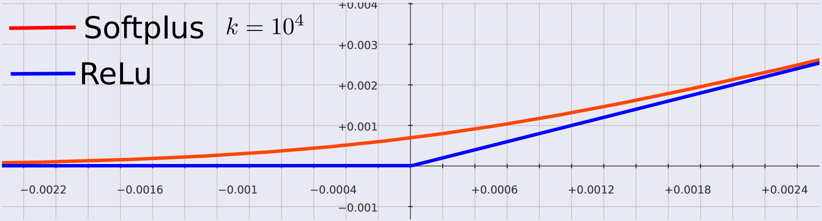
Therefore, we have

When $k = 1$



Increase k

When $k = 10^4$



Non-Linearity Layer

If layer l is a non-linearity layer

Its input is given by $m_1^{(l)}$ feature maps.

What about the output

Its output comprises again $m_1^{(l)} = m_1^{(l-1)}$ feature maps

Each of them of size

$$m_2^{(l-1)} \times m_3^{(l-1)} \tag{11}$$

With $m_2^{(l)} = m_2^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$.

Non-Linearity Layer

If layer l is a non-linearity layer

Its input is given by $m_1^{(l)}$ feature maps.

What about the output

Its output comprises again $m_1^{(l)} = m_1^{(l-1)}$ feature maps

Each of them of size

$$m_2^{(l-1)} \times m_3^{(l-1)} \tag{11}$$

With $m_2^{(l)} = m_2^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$.

Non-Linearity Layer

If layer l is a non-linearity layer

Its input is given by $m_1^{(l)}$ feature maps.

What about the output

Its output comprises again $m_1^{(l)} = m_1^{(l-1)}$ feature maps

Each of them of size

$$m_2^{(l-1)} \times m_3^{(l-1)} \tag{11}$$

With $m_2^{(l)} = m_2^{(l-1)}$ and $m_3^{(l)} = m_3^{(l-1)}$.

Thus

With the final output

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \quad (12)$$

Where

f is the activation function used in layer l and operates point wise.

You can also add a gain

$$Y_i^{(l)} = g_i f\left(Y_i^{(l-1)}\right) \quad (13)$$

Thus

With the final output

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \tag{12}$$

Where

f is the activation function used in layer l and operates point wise.

You can also add a gain

$$Y_i^{(l)} = g_i f\left(Y_i^{(l-1)}\right) \tag{13}$$

Thus

With the final output

$$Y_i^{(l)} = f\left(Y_i^{(l-1)}\right) \tag{12}$$

Where

f is the activation function used in layer l and operates point wise.

You can also add a gain

$$Y_i^{(l)} = g_i f\left(Y_i^{(l-1)}\right) \tag{13}$$

Cinvestav

Outline

Cinvestav

Rectification Layer, R_{abs}

Now a rectification layer

Then its input comprises $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Then, the absolute value for each component of the feature maps is computed

$$Y_i^{(l)} = \left| Y_i^{(l)} \right| \tag{14}$$

Where the absolute value

It is computed point wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

Rectification Layer, R_{abs}

Now a rectification layer

Then its input comprises $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Then, the absolute value for each component of the feature maps is computed

$$Y_i^{(l)} = \left| Y_i^{(l)} \right| \quad (14)$$

Where the absolute value

It is computed point wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

Rectification Layer, R_{abs}

Now a rectification layer

Then its input comprises $m_1^{(l)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$.

Then, the absolute value for each component of the feature maps is computed

$$Y_i^{(l)} = \left| Y_i^{(l)} \right| \tag{14}$$

Where the absolute value

It is computed point wise such that the output consists of $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

Cinvestav

Thus

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

We have that

Experiments show that rectification plays a central role in achieving good performance.

Thus

$$f\left(x\right)=\frac{\ln\left(1+e^{kx}\right)}{k}$$

We have that

Experiments show that rectification plays a central role in achieving good performance.

You can find this in

K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In Computer Vision, International Conference on, pages 2146–2153, 2009.

- But also it can be seen as an independent layer

Thus

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

We have that

Experiments show that rectification plays a central role in achieving good performance.

You can find this in

K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In Computer Vision, International Conference on, pages 2146–2153, 2009.

Remark

- Rectification could be included in the non-linearity layer.
- But also it can be seen as an independent layer.

Thus

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

We have that

Experiments show that rectification plays a central role in achieving good performance.

You can find this in

K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In Computer Vision, International Conference on, pages 2146–2153, 2009.

Remark

- Rectification could be included in the non-linearity layer.
- But also it can be seen as an independent layer.

Given that we are using Backpropagation

We need a soft approximation to $f(x) = |x|$

For this, we have

$$\frac{\partial f}{\partial x} = \mathsf{sgn}(x)$$

- When $x \neq 0$. Why?

We can use the following approximation

$$\mathsf{sgn}(x) = 2\left(\frac{\exp\{kx\}}{1+\exp\{kx\}}\right) - 1$$

Therefore, we have by integration and working the C

$$f(x) = \frac{2}{k}\ln(1+\exp\{kx\}) - x - \frac{2}{k}\ln(2)$$

Given that we are using Backpropagation

We need a soft approximation to $f(x) = |x|$

For this, we have

$$\frac{\partial f}{\partial x} = \mathsf{sgn}(x)$$

- When $x \neq 0$. Why?

We can use the following approximation

$$\mathsf{sgn}(x) = 2\left(\frac{\exp\{kx\}}{1+\exp\{kx\}}\right) - 1$$

Therefore, we have by integration and working the C

$$f(x) = \frac{2}{k}\ln(1+\exp\{kx\}) - x - \frac{2}{k}\ln(2)$$

Given that we are using Backpropagation

We need a soft approximation to $f(x) = |x|$

For this, we have

$$\frac{\partial f}{\partial x} = \mathsf{sgn}(x)$$

- When $x \neq 0$. Why?

We can use the following approximation

$$\mathsf{sgn}(x) = 2\left(\frac{\exp\{kx\}}{1+\exp\{kx\}}\right) - 1$$

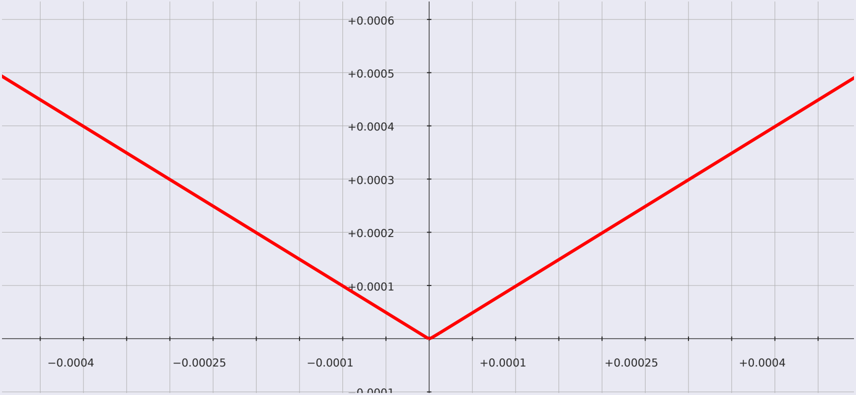
Therefore, we have by integration and working the C

$$f(x) = \frac{2}{k}\ln\left(1+\exp\{kx\}\right) - x - \frac{2}{k}\ln(2)$$

We get the following situation

Something Notable

$$f(x) = \frac{2}{k}\ln\left(1+\exp\{kx\}\right) - x - \frac{2}{k}\ln(2)$$



Cinvestav

Outline

Cinvestav

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

Cinvestav

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Normalizing

Contrast normalization layer

The task of a local contrast normalization layer:

- To enforce local competitiveness between adjacent units within a feature map.
- To enforce competitiveness units at the same spatial location.

We have two types of operations

- Subtractive Normalization.
- Brightness Normalization.

Subtractive Normalization

Given $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$

The output of layer l comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

With the operation

$$Y_i^{(l)} = Y_i^{(l-1)} - \sum_{j=1}^{m_1^{(l-1)}} K_{G(\sigma)} * Y_j^{(l-1)} \tag{15}$$

With

$$\left(K_{G(\sigma)}\right)_{r,s} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{r^2+s^2}{2\sigma^2}\right\} \tag{16}$$

Subtractive Normalization

Given $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$

The output of layer l comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

With the operation

$$Y_i^{(l)} = Y_i^{(l-1)} - \sum_{j=1}^{m_1^{(l-1)}} K_{G(\sigma)} * Y_j^{(l-1)} \tag{15}$$

With

$$\left(K_{G(\sigma)}\right)_{r,s} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{r^2+s^2}{2\sigma^2}\right\} \tag{16}$$

Subtractive Normalization

Given $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$

The output of layer l comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps unchanged in size.

With the operation

$$Y_i^{(l)} = Y_i^{(l-1)} - \sum_{j=1}^{m_1^{(l-1)}} K_{G(\sigma)} * Y_j^{(l-1)} \tag{15}$$

With

$$\left(K_{G(\sigma)}\right)_{r,s} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{r^2+s^2}{2\sigma^2}\right\} \tag{16}$$

Brightness Normalization

An alternative is to normalize the brightness in combination with the rectified linear units

$$\left(Y_i^{(l)}\right)_{r,s} = \frac{\left(Y_i^{(l-1)}\right)_{r,s}}{\left(\kappa + \lambda \sum_{j=1}^{m_1^{(l-1)}} \left(Y_j^{(l-1)}\right)_{r,s}^2\right)^{\mu}} \quad (17)$$

Where

- κ, μ and λ are hyperparameters which can be set using a

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

validation set.

Brightness Normalization

An alternative is to normalize the brightness in combination with the rectified linear units

$$\left(Y_i^{(l)}\right)_{r,s} = \frac{\left(Y_i^{(l-1)}\right)_{r,s}}{\left(\kappa + \lambda \sum_{j=1}^{m_1^{(l-1)}} \left(Y_j^{(l-1)}\right)_{r,s}^2\right)^{\mu}} \tag{17}$$

Where

- κ, μ and λ are hyperparameters which can be set using a

$$f(x) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

validation set.

Outline

Cinvestav

Subsampling Layer

Motivation

The motivation of subsampling the feature maps obtained by previous layers is robustness to noise and distortions.

How?

- Normally, in traditional Convolutional Networks subsampling this is done by applying skipping factors!!!
- However, it is possible to combine subsampling with pooling and do it in a separate laye

Subsampling Layer

Motivation

The motivation of subsampling the feature maps obtained by previous layers is robustness to noise and distortions.

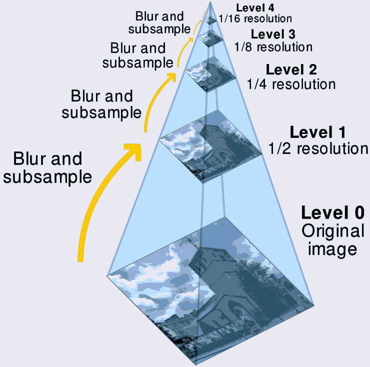
How?

- Normally, in traditional Convolutional Networks subsampling this is done by applying skipping factors!!!
- However, it is possible to combine subsampling with pooling and do it in a separate laye

Sub-sampling

The subsampling layer

- It seems to be acting as the well know sub-sampling pyramid



How is subsampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Cinvestav

How is subsampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

How is subsampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

How is subsampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

How is subsampling implemented?

We know that Image Pyramids

They were designed to find:

1. filter-based representations to decompose images into information at multiple scales,
2. To extract features/structures of interest,
3. To attenuate noise.

Example of usage of this filters

- The SURF and SIFT filters

Projection Vectors

Let $I \in \mathbb{R}^N$ an image

And a projection transformation such that

$$\boldsymbol{a} = PI$$

Where

$$\boldsymbol{a} = \begin{bmatrix} \boldsymbol{a}_0 & \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_{M-1} \end{bmatrix} \in \mathbb{R}^M$$

- The transformation coefficients...

Additionally, we have the projection vectors in P

$$P = \begin{bmatrix} \boldsymbol{p}_0 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_{M-1} \end{bmatrix}$$

Projection Vectors

Let $I \in \mathbb{R}^N$ an image

And a projection transformation such that

$$\boldsymbol{a} = PI$$

Where

$$\boldsymbol{a} = \begin{bmatrix} \boldsymbol{a}_0 & \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_{M-1} \end{bmatrix} \in \mathbb{R}^M$$

- The transformation coefficients...

Additionally, we have the projection vectors in P

$$P = \begin{bmatrix} \boldsymbol{p}_0 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_{M-1} \end{bmatrix}$$

Cinvestav

Projection Vectors

Let $I \in \mathbb{R}^N$ an image

And a projection transformation such that

$$\boldsymbol{a} = PI$$

Where

$$\boldsymbol{a} = \begin{bmatrix} \boldsymbol{a}_0 & \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_{M-1} \end{bmatrix} \in \mathbb{R}^M$$

- The transformation coefficients...

Additionally, we have the projection vectors in P

$$P = \begin{bmatrix} \boldsymbol{p}_0 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_{M-1} \end{bmatrix}$$

Thus, we have the following cases

When $M = N$

- Thus, the projection P is to be critically sampled (Relation with the rank of P)

When $N < M$

- Over-sampled

When $M < N$

- Under-sampled

Thus, we have the following cases

When $M = N$

- Thus, the projection P is to be critically sampled (Relation with the rank of P)

When $N < M$

- Over-sampled

When $M < N$

- Under-sampled

Thus, we have the following cases

When $M = N$

- Thus, the projection P is to be critically sampled (Relation with the rank of P)

When $N < M$

- Over-sampled

When $M < N$

- Under-sampled

Therefore

We have that we can build a series of subsampled images

$$\left\{ \begin{matrix} I_0 & I_1 & \cdots & I_T \end{matrix} \right\}$$

Usually constructed with a separable 1D kernel h

$$I_{k+1} = PI_k = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}}_{\text{down-sampling}} \underbrace{\begin{pmatrix} \ddots & & & & & \\ - & h & - & & & \\ & - & h & - & & \\ & & - & h & - & \\ & & & & & \ddots \end{pmatrix}}_{\text{conv toplitz matrix}} I_k$$

Therefore

We have that we can build a series of subsampled images

$$\left\{ \; I_0 \quad I_1 \quad \cdots \quad I_T \; \right\}$$

Usually constructed with a separable 1D kernel h

$$I_{k+1} = PI_k = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}}_{\text{down-sampling}} \underbrace{\begin{pmatrix} \ddots & & & & & \\ - & h & - & & & \\ & - & h & - & & \\ & & - & h & - & \\ & & & & \ddots & \end{pmatrix}}_{\text{conv toplitz matrix}} I_k$$

There are also other ways of doing this

subsampling can be done using so called skipping factors

$$s_1^{(l)} \text{ and } s_2^{(l)}$$

The basic idea is to skip a fixed number of pixels

Therefore the size of the output feature map is given by

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)} + 1} \text{ and } m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)} + 1}$$

There are also other ways of doing this

subsampling can be done using so called skipping factors

$$s_1^{(l)} \text{ and } s_2^{(l)}$$

The basic idea is to skip a fixed number of pixels

Therefore the size of the output feature map is given by

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)} + 1} \text{ and } m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)} + 1}$$

What is Pooling?

Pooling

Spatial pooling is way to compute image representation based on encoded local features.

Pooling

Let l be a pooling layer

Its output comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps of reduced size.

Pooling Operation

It operates by placing windows at non-overlapping positions in each feature map and keeping one value per window such that the feature maps are subsampled.

Cinvestav

Pooling

Let l be a pooling layer

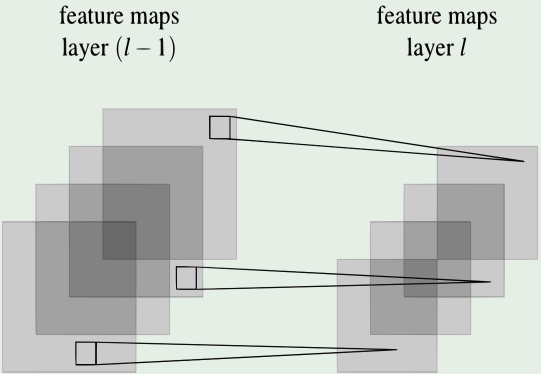
Its output comprises $m_1^{(l)} = m_1^{(l-1)}$ feature maps of reduced size.

Pooling Operation

It operates by placing windows at non-overlapping positions in each feature map and keeping one value per window such that the feature maps are subsampled.

Example

If layer l is a pooling and subsampling layer and given $m_1^{(l-1)} = 4$ feature maps



Thus

In the previous example

All feature maps are pooled and subsampled individually.

Each unit

In one of the $m_1^{(l)} = 4$ output feature maps represents the average or the maximum within a fixed window of the corresponding feature map in layer $(l-1)$.

Thus

In the previous example

All feature maps are pooled and subsampled individually.

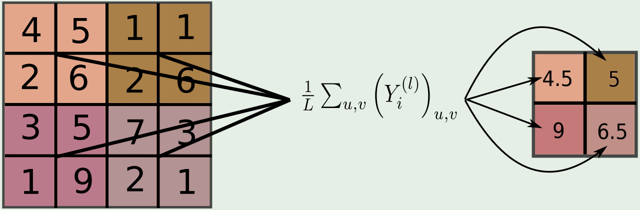
Each unit

In one of the $m_1^{(l)} = 4$ output feature maps represents the average or the maximum within a fixed window of the corresponding feature map in layer $(l-1)$.

We distinguish two types of pooling

Average pooling

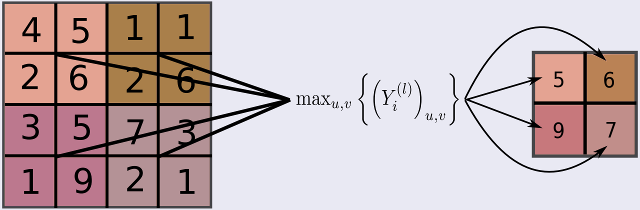
When using a boxcar filter, the operation is called average pooling and the layer denoted by P_A.



We distinguish two types of pooling

Max pooling

For max pooling, the maximum value of each window is taken. The layer is denoted by P_M.



Outline

Cinvestav

Fully Connected Layer

If a layer l is a fully connected layer

If layer $(l-1)$ is a fully connected layer, use the equation to compute the output of i^{th} unit at layer l:

$$z_i^{(l)} = \sum_{k=0}^{m^{(l)}} w_{i,k}^{(l)} y_k^{(l)} \text{ thus } y_i^{(l)} = f\left(z_i^{(l)}\right)$$

Otherwise

Layer l expects $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ as input.

Fully Connected Layer

If a layer l is a fully connected layer

If layer $(l-1)$ is a fully connected layer, use the equation to compute the output of i^{th} unit at layer l:

$$z_i^{(l)} = \sum_{k=0}^{m^{(l)}} w_{i,k}^{(l)} y_k^{(l)} \text{ thus } y_i^{(l)} = f\left(z_i^{(l)}\right)$$

Otherwise

Layer l expects $m_1^{(l-1)}$ feature maps of size $m_2^{(l-1)} \times m_3^{(l-1)}$ as input.

Then

Thus, the i^{th} unit in layer l computes

$$y_i^{(l)} = f\left(z_i^{(l)}\right)$$

$$z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} w_{i,j,r,s}^{(l)} \left(Y_j^{(l-1)}\right)_{r,s}$$

Cinvestav

Here

Where $w_{i,j,r,s}^{(l)}$

- It denotes the weight connecting the unit at position (r, s) in the j^{th} feature map of layer $(l-1)$ and the i^{th} unit in layer l.

Something Notable

- In practice, Convolutional Layers are used to learn a feature hierarchy and one or more fully connected layers are used for classification purposes based on the computed features.

Here

Where $w_{i,j,r,s}^{(l)}$

- It denotes the weight connecting the unit at position (r, s) in the j^{th} feature map of layer $(l-1)$ and the i^{th} unit in layer l.

Something Notable

- In practice, Convolutional Layers are used to learn a feature hierarchy and one or more fully connected layers are used for classification purposes based on the computed features.

Basically

We can use a loss function at the output of such layer

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N}\sum_{k=1}^{K}\left(y_{nk}^{(l)} - t_{nk}\right)^2 \text{ (Sum of Squared Error)}$$

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\log\left(y_{nk}^{(l)}\right) \text{ (Cross-Entropy Error)}$$

Assuming W the tensor used to represent all the possible weights

- We can use the Backpropagation idea as long we can apply the corresponding derivatives.

Basically

We can use a loss function at the output of such layer

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N}\sum_{k=1}^{K}\left(y_{nk}^{(l)} - t_{nk}\right)^2 \text{ (Sum of Squared Error)}$$

$$L(\boldsymbol{W}) = \sum_{n=1}^{N} E_n(\boldsymbol{W}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\log\left(y_{nk}^{(l)}\right) \text{ (Cross-Entropy Error)}$$

Assuming $\boldsymbol{W}$ the tensor used to represent all the possible weights

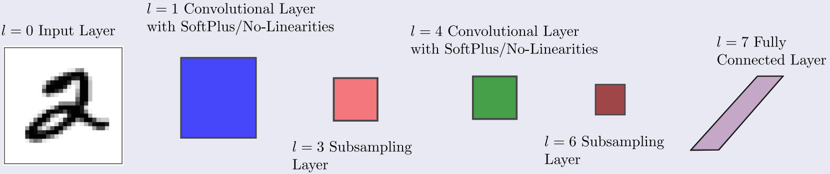
- We can use the Backpropagation idea as long we can apply the corresponding derivatives.

Outline

Cinvestav

We have the following Architecture

Simplified Architecture by Jean LeCun "Backpropagation applied to handwritten zip code recognition"



Therefore, we have

Layer $l = 1$

- This Layer is using a Softplus f with 1 channels $j = 1$ Black and White

$$f\left[\left(Y_1^{(1)}\right)_{r,s}\right] = f\left[\left(B_1^{(l)}\right)_{r,s} + \sum_{k=-h_1^{(1)}}^{h_1^{(1)}} \sum_{t=-h_2^{(1)}}^{h_2^{(1)}} \left(K_{ij}^{(1)}\right)_{k,t} \left(Y_1^{(0)}\right)_{r+k,s+t}\right]$$

Now

We have the $l = 2$ subsampling for each coordinate

$$Y_1^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} f\left[\left(Y_1^{(1)}\right)\right] \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}^T$$

Cinvestav

Then, you repeat the previous

Thus we obtain a reduced convoluted version $Y_1^{(6)}$ of the $Y_1^{(4)}$ convolution and subsampling

- Thus, we use those as inputs for the fully connected layer of input.

Now assuming a single $k=1$ neuron

$$y_1^{(7)} = f\left(z_1^{(7)}\right)$$

$$z_1^{(7)} = \sum_{r=1}^{m_2^{(6)}} \sum_{s=1}^{m_3^{(6)}} w_{r,s}^{(7)} \left(Y_1^{(6)}\right)_{r,s}$$

Then, you repeat the previous

Thus we obtain a reduced convoluted version $Y_1^{(6)}$ of the $Y_1^{(4)}$ convolution and subsampling

- Thus, we use those as inputs for the fully connected layer of input.

Now assuming a single $k = 1$ neuron

$$y_1^{(7)} = f\left(z_1^{(7)}\right)$$

$$z_1^{(7)} = \sum_{r=1}^{m_2^{(6)}} \sum_{s=1}^{m_3^{(6)}} w_{r,s}^{(7)} \left(Y_1^{(6)}\right)_{r,s}$$

Cinvestav

We have

That our final cost function is equal to

$$L(\boldsymbol{t}) = \frac{1}{2}\left(y_1^{(7)} - t_1^{(7)}\right)^2$$

Outline

After collecting all input/output

Therefore

- We have using sum of squared errors (loss function):

$$\min_{\boldsymbol{W}} H(\boldsymbol{W}) = \frac{1}{2}\left(y_1^{(7)} - t_1^{(7)}\right)^2$$

Therefore, we can obtain

$$\frac{\partial H(\boldsymbol{W})}{\partial w_{1,r,s}^{(7)}} = \frac{1}{2} \times \frac{\partial\left(y_1^{(7)} - t_1^{(7)}\right)^2}{\partial w_{1,r,s}^{(7)}}$$

After collecting all input/output

Therefore

- We have using sum of squared errors (loss function):

$$\min_{\boldsymbol{W}} H(\boldsymbol{W}) = \frac{1}{2}\left(y_1^{(7)} - t_1^{(7)}\right)^2$$

Therefore, we can obtain

$$\frac{\partial H(\boldsymbol{W})}{\partial w_{1,r,s}^{(7)}} = \frac{1}{2} \times \frac{\partial \left(y_1^{(7)} - t_1^{(7)}\right)^2}{\partial w_{1,r,s}^{(7)}}$$

Therefore

We get in the first part of the equation

$$\frac{\partial \left(t_1 - y_1^{(7)}\right)^2}{\partial w_{1,r,s}^{(7)}} = \left(y_1^{(7)} - t_1^{(7)}\right) \frac{\partial y_1^{(7)}}{\partial w_{1,r,s}^{(7)}}$$

With

$$y_1^{(7)} = f\left(z_1^{(7)}\right) = \frac{\ln\left(1 + e^{kz_k^{(7)}}\right)}{k}$$

Therefore

We get in the first part of the equation

$$\frac{\partial\left(t_1 - y_1^{(7)}\right)^2}{\partial w_{1,r,s}^{(7)}} = \left(y_1^{(7)} - t_1^{(7)}\right)\frac{\partial y_1^{(7)}}{\partial w_{1,r,s}^{(7)}}$$

With

$$y_1^{(7)} = f\left(z_1^{(7)}\right) = \frac{\ln\left(1 + e^{kz_k^{(7)}}\right)}{k}$$

Therefore

We have

$$\frac{\partial y_1^{(7)}}{\partial w_{1,r,s}^{(7)}} = \frac{\partial f\left(z_1^{(7)}\right)}{\partial z_1^{(7)}} \times \frac{\partial z_1^{(7)}}{\partial w_{1,r,s}^{(7)}}$$

Therefore

$$\frac{\partial f\left(z_1^{(7)}\right)}{\partial z_1^{(7)}} = \frac{e^{kz_1^{(7)}}}{\left(1+e^{kz_1^{(7)}}\right)}$$

Finally

$$\frac{\partial z_1^{(7)}}{\partial w_{1,r,s}^{(7)}} = \left(Y_1^{(6)}\right)_{r,s}$$

Therefore

We have

$$\frac{\partial y_1^{(7)}}{\partial w_{1,r,s}^{(7)}} = \frac{\partial f\left(z_1^{(7)}\right)}{\partial z_1^{(7)}} \times \frac{\partial z_1^{(7)}}{\partial w_{1,r,s}^{(7)}}$$

Therefore

$$\frac{\partial f\left(z_1^{(7)}\right)}{\partial z_1^{(7)}} = \frac{e^{kz_1^{(7)}}}{\left(1+e^{kz_1^{(7)}}\right)}$$

Finally

$$\frac{\partial z_1^{(7)}}{\partial w_{1,r,s}^{(7)}} = \left(Y_1^{(6)}\right)_{r,s}$$

Therefore

We have

$$\frac{\partial y_1^{(7)}}{\partial w_{1,r,s}^{(7)}} = \frac{\partial f\left(z_1^{(7)}\right)}{\partial z_1^{(7)}} \times \frac{\partial z_1^{(7)}}{\partial w_{1,r,s}^{(7)}}$$

Therefore

$$\frac{\partial f\left(z_1^{(7)}\right)}{\partial z_1^{(7)}} = \frac{e^{kz_1^{(7)}}}{\left(1+e^{kz_1^{(7)}}\right)}$$

Finally

$$\frac{\partial z_1^{(7)}}{\partial w_{1,r,s}^{(7)}} = \left(Y_1^{(6)}\right)_{r,s}$$

Now

Given the pooling

$$Y_1^{(6)} = Sf\left[\left(Y_1^{(4)}\right)\right]S^T$$

We have that

$$\left(Y_1^{(4)}\right)_{r,s} = \left(B_1^{(4)}\right)_{r,s} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{l=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{11}^{(4)}\right)_{k,l} \left(Y^{(3)}\right)_{r+k,s+l}$$

Now

Given the pooling

$$Y_1^{(6)} = Sf\left[\left(Y_1^{(4)}\right)\right]S^T$$

We have that

$$\left(Y_1^{(4)}\right)_{r,s} = \left(B_1^{(4)}\right)_{r,s} + \sum_{k=-h_1^{(l)}}^{h_1^{(l)}} \sum_{t=-h_2^{(l)}}^{h_2^{(l)}} \left(K_{11}^{(4)}\right)_{k,t} \left(Y^{(3)}\right)_{r+k,s+t}$$

Therefore

We have then

$$\frac{\partial H\left(\boldsymbol{W}\right)}{\partial\left(K_{11}^{(4)}\right)_{k,t}}=\frac{1}{2}\times\frac{\partial\left(y_{1}^{(7)}-t_{1}\right)^{2}}{\partial\left(K_{11}^{(4)}\right)_{k,t}}$$

We have the following chain of derivations

$$\frac{\partial H\left(\boldsymbol{W}\right)}{\partial\left(K_{11}^{(4)}\right)_{k,t}}=\left(y_{i}^{(l)}-t_{i}\right)\frac{\partial f\left(z_{i}^{(7)}\right)}{\partial z_{i}^{(7)}}\times\frac{\partial z_{i}^{(7)}}{\partial\left(Y_{1}^{(6)}\right)_{r,s}}\times\frac{\partial\left(Y_{1}^{(6)}\right)_{r,s}}{\partial\left(K_{11}^{(4)}\right)_{k,t}}$$

Therefore

We have then

$$\frac{\partial H\left(\boldsymbol{W}\right)}{\partial\left(K_{11}^{(4)}\right)_{k,t}}=\frac{1}{2}\times\frac{\partial\left(y_{1}^{(7)}-t_{1}\right)^{2}}{\partial\left(K_{11}^{(4)}\right)_{k,t}}$$

We have the following chain of derivations

$$\frac{\partial H\left(\boldsymbol{W}\right)}{\partial\left(K_{11}^{(4)}\right)_{k,t}}=\left(y_{i}^{(l)}-t_{i}\right)\frac{\partial f\left(z_{i}^{(7)}\right)}{\partial z_{i}^{(7)}}\times\frac{\partial z_{i}^{(7)}}{\partial\left(Y_{1}^{(6)}\right)_{r,s}}\times\frac{\partial\left(Y_{1}^{(6)}\right)_{r,s}}{\partial\left(K_{11}^{(4)}\right)_{k,t}}$$

Therefore

We have

$$\frac{\partial z_i^{(7)}}{\partial \left(Y_1^{(6)}\right)_{r,s}} = w_{r,s}^{(7)}$$

The final convolution is assuming that

$$\frac{\partial \left(Y_1^{(6)}\right)_{r,s}}{\partial \left(K_{11}^{(4)}\right)_{k,t}} = \frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial \left(K_{11}^{(4)}\right)_{k,t}}$$

Therefore

We have

$$\frac{\partial z_i^{(7)}}{\partial \left(Y_1^{(6)}\right)_{r,s}} = w_{r,s}^{(7)}$$

The final convolution is assuming that

$$\frac{\partial \left(Y_1^{(6)}\right)_{r,s}}{\partial \left(K_{11}^{(4)}\right)_{k,t}} = \frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial \left(K_{11}^{(4)}\right)_{k,t}}$$

Therefore

We have

$$\frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial\left(K_{11}^{(4)}\right)_{k,t}} = \frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}} \times \frac{\partial\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}}{\partial\left(K_{11}^{(4)}\right)_{k,t}}$$

Then

$$\frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}} = f'\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]$$

Therefore

We have

$$\frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial\left(K_{11}^{(4)}\right)_{k,t}}=\frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}}\times\frac{\partial\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}}{\partial\left(K_{11}^{(4)}\right)_{k,t}}$$

Then

$$\frac{\partial f\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]}{\partial\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}}=f'\left[\left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}\right]$$

Cinvestav

Finally, we have

The equation

$$\frac{\partial \left(Y_1^{(4)}\right)_{2(r-1),2(s-1)}}{\partial \left(K_{11}^{(4)}\right)_{k,t}} = \left(Y^{(3)}\right)_{2(r-1)+k,2(s-1)+t}$$

Cinvestav

The Other Equations

I will leave you to devise them

- They are a repetitive procedure.

R. Szeliski, *Computer Vision: Algorithms and Applications*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 2010.

S. Haykin, *Neural Networks and Learning Machines*. No. v. 10 in Neural networks and learning machines, Prentice Hall, 2009.

D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

W. Zhang, K. Itoh, J. Tanida, and Y. Ichioka, "Parallel distributed processing model with local space-invariant interconnections and its optical architecture," *Appl. Opt.*, vol. 29, pp. 4790–4797, Nov 1990.

J. J. Weng, N. Ahuja, and T. S. Huang, "Learning recognition and segmentation of 3-d objects from 2-d images," in *1993 (4th) International Conference on Computer Vision*, pp. 121–128, IEEE, 1993.

Cinvestav