CSC321 Lecture 11
Convolutional networks

Roger Grosse and Nitish Srivastava

February 15, 2015

Overview

The last two weeks were about modeling sequences, with an emphasis on language.

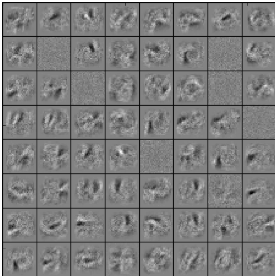
Now we'll turn to vision, which presents a different set of challenges.

Overview

The last two weeks were about modeling sequences, with an emphasis on language.

Now we'll turn to vision, which presents a different set of challenges.

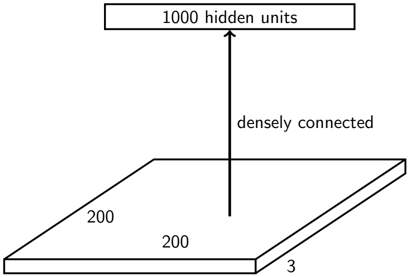
Recall we looked at some hidden layer features for classifying handwritten digits:



This isn't going to scale to full-sized images.

Neural Net on Image

Suppose we want to train a network that takes a 200 $\times$ 200 RGB image as input.

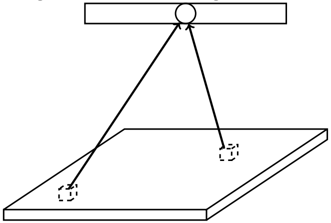


What is the problem with having this as the first layer ?

- Too many parameters! Input size = 200 $\times$ 200 $\times$ 3 = 120K. Parameters = 120K $\times$ 1000 = 120 million.
- What happens if the object in the image shifts a little ?

Neural Net on Image

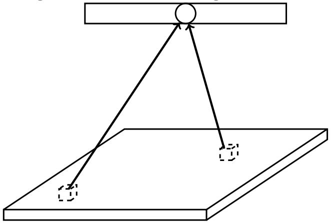
Each feature (hidden unit) looks at the entire image.
Since the image is a BIG thing, we end up with lots of parameters.



But, do we really expect to learn a useful feature at the first layer which depends on pixels that are spatially far away ?

Neural Net on Image

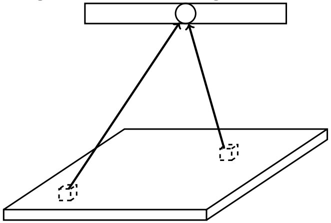
Each feature (hidden unit) looks at the entire image.
Since the image is a BIG thing, we end up with lots of parameters.



But, do we really expect to learn a useful feature at the first layer which depends on pixels that are spatially far away ?
The far away pixels will probably belong to completely different objects (or object sub-parts). Very little correlation.

Neural Net on Image

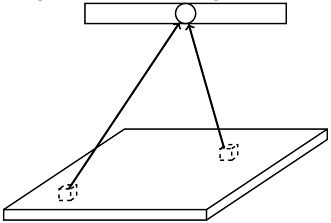
Each feature (hidden unit) looks at the entire image.
Since the image is a BIG thing, we end up with lots of parameters.



But, do we really expect to learn a useful feature at the first layer which depends on pixels that are spatially far away ?
The far away pixels will probably belong to completely different objects (or object sub-parts). Very little correlation.
No point devoting parameters to find correlations that (almost) don't exist.

Neural Net on Image

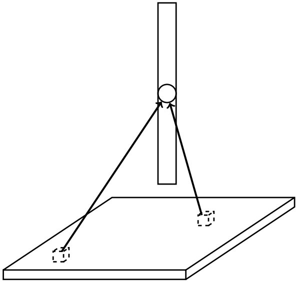
Each feature (hidden unit) looks at the entire image.
Since the image is a BIG thing, we end up with lots of parameters.



But, do we really expect to learn a useful feature at the first layer which depends on pixels that are spatially far away ?
The far away pixels will probably belong to completely different objects (or object sub-parts). Very little correlation.
No point devoting parameters to find correlations that (almost) don't exist.
Long range correlations can be dealt with in the higher layers.

Going Local

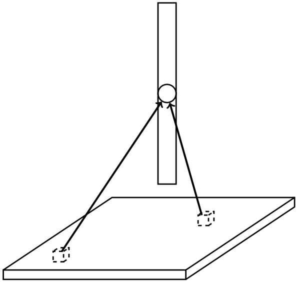
Fully connected



Each hidden unit looks at the entire image.

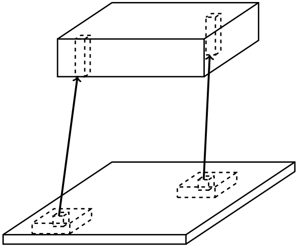
Going Local

Fully connected



Each hidden unit looks at the entire image.

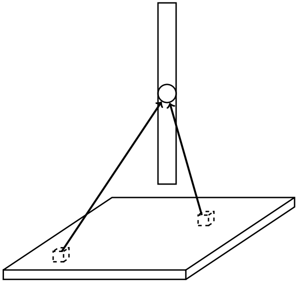
Locally connected



Each column of hidden units looks at a different patch of input.

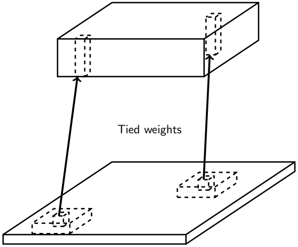
Going Convolutional

Fully connected



Each hidden unit looks at the entire image.

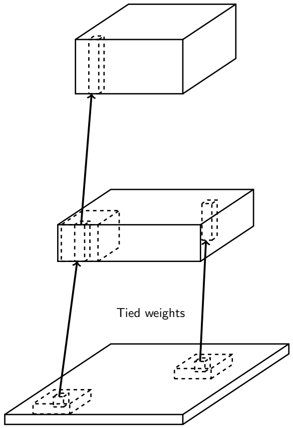
Convolutional



Each column of hidden units looks at a different patch of input.

Going Deeply Convolutional

Stack multiple layers of convolutions



Convolution

In Assignment 1, you expressed the computations in terms of matrix multiplication in order to avoid writing (slow) for loops.

Now we'll introduce a new high-level operation, convolution. Let's look at the 1-D case first.

Convolution

In Assignment 1, you expressed the computations in terms of matrix multiplication in order to avoid writing (slow) for loops.

Now we'll introduce a new high-level operation, convolution. Let's look at the 1-D case first.

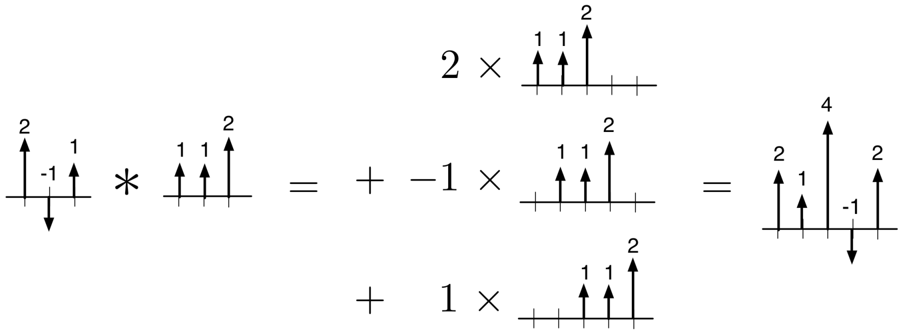
If *a* and *b* are two arrays,

$$(a * b)_i = \sum_t a_t b_{i-t}.$$

Note: indexing conventions are inconsistent. We'll explain them in each case.

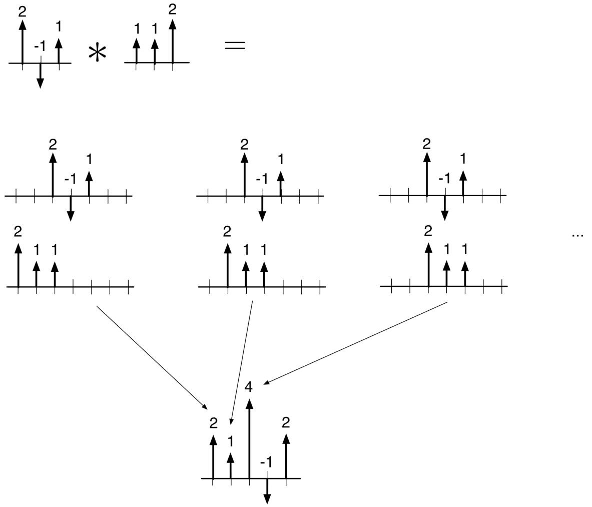
Convolution

Method 1: translate-and-scale



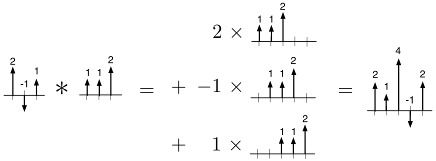
Convolution

Method 2: flip-and-filter

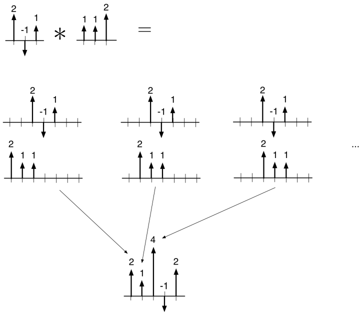


Question 1: Convolution

Compute $(3, 1, 2) * (1, 0, -1)$ using both methods. (Figures from the previous slides are shown here as a reminder.)



translate-and-scale



flip-and-filter

Convolution

Convolution can also be viewed as matrix multiplication:

$$(2, -1, 1) * (1, 1, 2) = \begin{pmatrix} 1 & & \\ 1 & 1 & \\ 2 & 1 & 1 \\ & 2 & 1 \\ & & 2 \end{pmatrix} \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$$

Convolution

Some properties of convolution:

- Commutativity

$$a * b = b * a$$

- Linearity

$$a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c$$

2-D Convolution

2-D convolution is defined analogously to 1-D convolution.

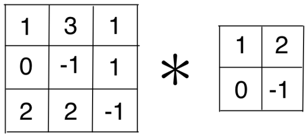
If A and B are two 2-D arrays, then:

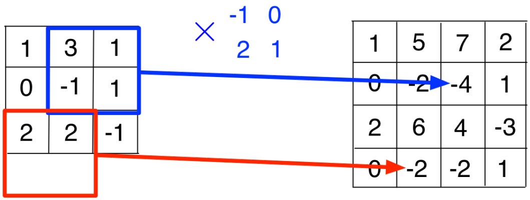
$$(A * B)_{ij} = \sum_s \sum_t A_{st} B_{i-s,j-t}.$$

2-D Convolution

$$\begin{array}{|c|c|c|}\hline 1&3&1\\\hline 0&-1&1\\\hline 2&2&-1\\\hline\end{array} * \begin{array}{|c|c|}\hline 1&2\\\hline 0&-1\\\hline\end{array} = 1\times\begin{array}{|c|c|c|c|}\hline 1&3&1&\\\hline 0&-1&1&\\\hline 2&2&-1&\\\hline &&&\\\hline\end{array} + 2\times\begin{array}{|c|c|c|c|}\hline &1&3&1\\\hline &0&-1&1\\\hline &2&2&-1\\\hline &&&\\\hline\end{array} + -1\times\begin{array}{|c|c|c|c|}\hline &&&\\\hline &1&3&1\\\hline &0&-1&1\\\hline &2&2&-1\\\hline\end{array} = \begin{array}{|c|c|c|c|}\hline 1&5&7&2\\\hline 0&-2&-4&1\\\hline 2&6&4&-3\\\hline 0&-2&-2&1\\\hline\end{array}$$

2-D Convolution





2-D Convolution

What does this convolution kernel do?

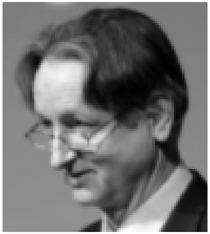
$$* \quad \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

2-D Convolution

What does this convolution kernel do?

$*$

0	1	0
1	4	1
0	1	0



2-D Convolution

What does this convolution kernel do?

$*$

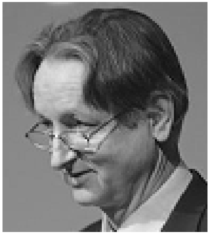
0	-1	0
-1	8	-1
0	-1	0

2-D Convolution

What does this convolution kernel do?

$*$

0	-1	0
-1	8	-1
0	-1	0



2-D Convolution

What does this convolution kernel do?

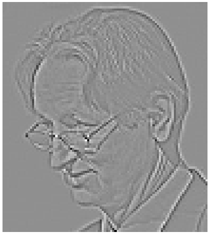
$$*$$

0	-1	0
-1	4	-1
0	-1	0

2-D Convolution

What does this convolution kernel do?

$$* \quad \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array}$$



2-D Convolution

What does this convolution kernel do?

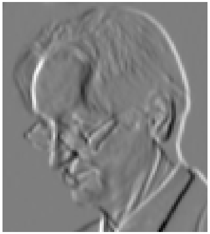
$*$

1	0	-1
2	0	-2
1	0	-1

2-D Convolution

What does this convolution kernel do?

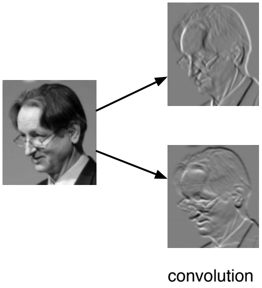
$$* \quad \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array}$$



Convolutional networks

Let's finally turn to convolutional networks. These have two kinds of layers: detection layers (or convolution layers), and pooling layers.

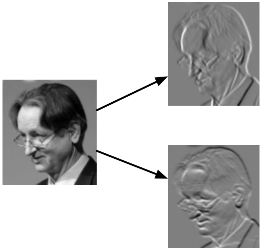
The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.



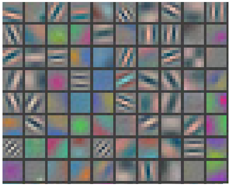
Convolutional networks

Let's finally turn to convolutional networks. These have two kinds of layers: detection layers (or convolution layers), and pooling layers.

The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.



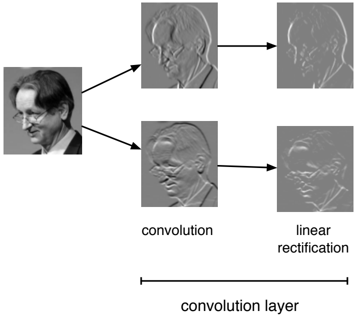
Example first-layer filters



(Zeiler and Fergus, 2013, Visualizing and understanding convolutional networks)

Convolutional networks

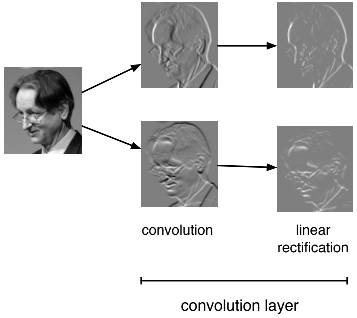
It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$



Why might we do this?

Convolutional networks

It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$

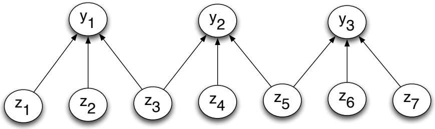


Why might we do this?

- Convolution is a linear operation. Therefore, we need a nonlinearity, otherwise 2 convolution layers would be no more powerful than 1.
- Two edges in opposite directions shouldn't cancel
- Makes the gradients sparse, which helps optimization (recall the backprop exercise from Lecture 6)

Pooling layers

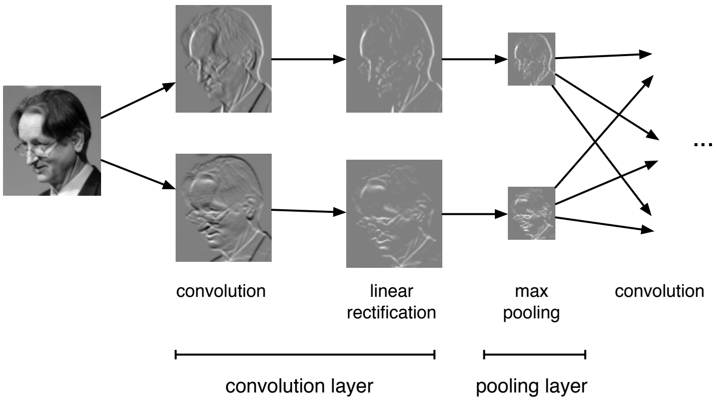
The other type of layer in a pooling layer. These layers reduce the size of the representation and build in invariance to small transformations.



Most commonly, we use max-pooling:

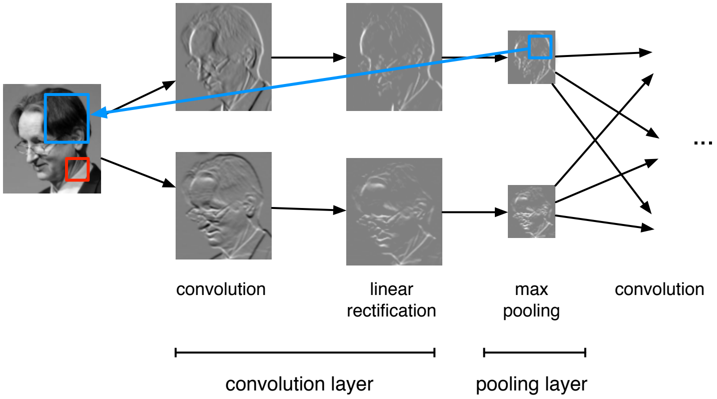
$$y_i = \max_{j \text{ in pooling group}} z_j$$

Convolutional networks



Convolutional networks

Because of pooling, higher-layer filters can cover a larger region of the input than equal-sized filters in the lower layers.



Question 2: Backprop in conv nets

Now let's consider how to train conv nets using backprop.

Question 2: Backprop in conv nets

Now let's consider how to train conv nets using backprop.

We can implement the network in a modular fashion, with a class for each layer type (convolution, pooling, etc.).

- We'll derive the updates for one layer at a time, assuming the surrounding layers have already done their jobs.

Question 2: Backprop in conv nets

Now let's consider how to train conv nets using backprop.

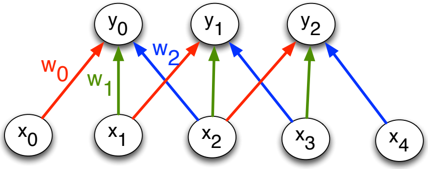
We can implement the network in a modular fashion, with a class for each layer type (convolution, pooling, etc.).

- We'll derive the updates for one layer at a time, assuming the surrounding layers have already done their jobs.

Unlike with recurrent nets, we tend not to get exploding/vanishing gradients. Vanilla backprop with momentum works very well.

Question 2: Backprop in conv nets

In Assignment 1, we expressed the backprop computations in terms of matrix multiplication for efficiency. For conv nets, we express the computations in terms of convolution. Here's a convolution layer with weights $\mathbf{w} = (w_0, w_1, w_2)$. All units are linear.



1. Show how to compute the activations $\mathbf{y} = (y_0, y_1, y_2)$ using convolution.
2. Compute the partial derivatives $\partial C / \partial x_j$ and $\partial C / \partial w_k$ in terms of $\partial C / \partial y_i$.
3. Express the gradients $\nabla_{\mathbf{x}} C$ and $\nabla_{\mathbf{w}} C$ in terms of convolution.

Recall: $(a * b)_i = \sum_t a_t b_{i-t}$.

Question 2: Backprop in conv nets (solution)

Observe that the y_i's are computed by filtering the input with the weights $\mathbf{w}$. Hence, using the flip-and-filter interpretation of convolution, we can write this as $\mathbf{y} = \mathbf{x} * \mathrm{flip}(\mathbf{w})$, where $\mathrm{flip}(\mathbf{w})$ denotes reversing the entries of $\mathbf{w}$.

By applying the chain rule,

$$\begin{aligned} \frac{\partial C}{\partial x_j} &= \sum_i \frac{\partial C}{\partial y_i} \frac{\partial y_i}{\partial x_j} \\ &= \sum_{i \text{ s.t. } 0 \leq j-i \leq 2} \frac{\partial C}{\partial y_i} w_{j-i} \end{aligned}$$

This is just the definition of convolution, i.e. $\nabla_{\mathbf{x}} C = \nabla_{\mathbf{y}} C * \mathbf{w}$.

Question 2: Backprop in conv nets (solution)

To compute $\partial C / \partial w_k$, we need to sum over all the connections that share the weight w_k. This gives us

$$\frac{\partial C}{\partial w_k} = \sum_{i=0}^{2} x_{k+i} \frac{\partial C}{\partial y_i}.$$

This corresponds to filtering $\mathbf{x}$ with the gradient vector $\mathbf{y}$. E.g., the first element $\partial C / \partial w_0$ is given by $\nabla_{\mathbf{y}} C$ dotted with the first window of $\mathbf{x}$, and so on. Using the flip-and-filter interpretation of convolution, we can write this as $\nabla_{\mathbf{w}} C = \mathbf{x} * \mathrm{flip}(\nabla_{\mathbf{y}} C)$.