Slides:

goo.gl/jvxqZB

Repo:

goo.gl/AfkCn6

Convolutional Neural Networks

Machine Learning Workshop Series

Michal Kazmierski

February 28, 2018

Imperial College Data Science Society

Theory

Motivating example - image classification

Problem: Given a set $\mathcal{S}$ of images with associated class labels:

$$\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

where

$$x_i \in \mathbb{R}^{W \times H \times C}$$

(W, H, C - width, height and channels), and

$$y_i = 0, 1, \ldots, K$$

find a function f that maps $x_i \to y_i$:

$$f : \mathcal{X} \to \mathcal{Y}$$

A typical classification setup.

Example - Fashion-MNIST

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

First attempt - standard neural network

Flatten each image to 1D vector.

Compute the network predictions:

$$f(\mathbf{X}) = \hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{W_1} + \mathbf{b_1})\mathbf{W_2} + \mathbf{b_2}$$

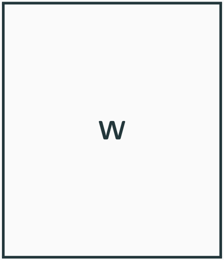
Compute the loss function:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$$

Find the gradients of the loss function wrt. weights through backpropagation and update the parameters.

Problem - large number of parameters

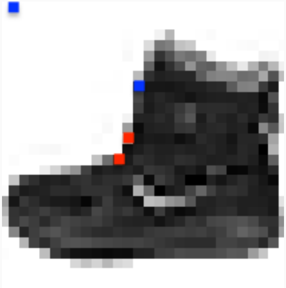
For a 256×256 RGB image (not large for today's standards!), a single neuron would have $256 \times 256 \times 3 = 196,608$ parameters. For a hidden layer of size 64, that's 12,582,912 parameters just for the first layer.



(input size $\times$ number of neurons)

Problem - spatial relationships

Every neuron in layer l_{i-1} is connected to every neuron in layer l_i $\rightarrow$ spatial relationships are ignored.



Problem - feature location

In classification, we don't need to know the exact location of a feature (e.g. a face) - its presence is enough.

Image source

A llama is a llama regardless of its location in the image

Convolutional neural networks

Convolutional neural network is a composition of differentiable functions (layers), just like a standard neural network. Each layer takes an input **volume** (3D array) and transforms it into another volume:

$$f : \mathbb{R}^{W_{in} \times H_{in} \times C_{in}} \to \mathbb{R}^{W_{out} \times H_{out} \times C_{out}}$$

Convolutional neural networks - building blocks

The basic convnet layers are:

- **convolutional layer**
- **pooling layer**
- **nonlinearity** - identical to standard neural nets, e.g. ReLU ($\max(0, x)$)
- **fully-connected layer** - equivalent to a standard neural net layer

Convolutional layer

Instead of multiplying by the input by the weight matrix:

$$f(\mathbf{X}) = \mathbf{XW} + \mathbf{b}$$

convolve the input with a *kernel*.

Kernels

Kernel:

$$\mathbf{W} \in \mathbb{R}^{M \times N \times C_{in}}$$

where M, N - hyperparameters

C_{in} - number of input channels

Forward pass:

$$\begin{aligned} f(\mathbf{X}_{ij}) &= (\mathbf{W} * \mathbf{X}_{ij}) + \mathbf{b} \\ &= \sum_{m=0}^{M} \sum_{n=0}^{N} \sum_{c=0}^{C_i} \mathbf{X}_{i-m,j-n,c} \mathbf{W}_{m,n,c} + \mathbf{b} \end{aligned}$$

Produces a 2D *activation map*.

Repeat for every kernel in the layer - each activation map is a new 'channel'.

Convolutional layer - hyperparameters

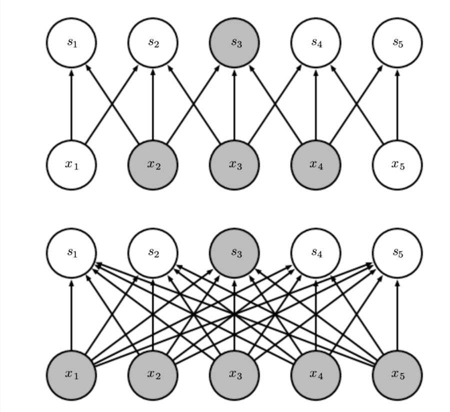
- **the width and height of each kernel** (M, N) - often $M = N$
- **the number of kernels to use** $\rightarrow$ the number of output channels
- **stride and padding** - will be discussed in the Codelab

Why use convolutions?

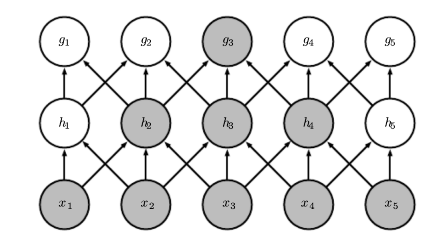
Convolutions adapt the neural network framework specifically for images through:

- **local connectivity**
- **shared parameters (tied weights)**
- **equivariance to translation**

Local connectivity



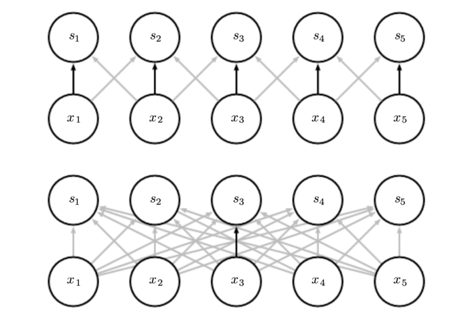
Local connectivity



Shared parameters

The same kernel is used for the whole input volume $\rightarrow$ less parameters.

Assuming $\mathbf{W} \in \mathbb{R}^{3\times3\times3}$ that's $3^3 = 27$ parameters (+ 1 for bias) per neuron.



Translational equivariance

Let $g(\mathbf{X}) = \mathbf{X}'$ be a transformation such that $\mathbf{X}'_{ij} = \mathbf{X}_{(i-1)j}$, that is, shifting (translating) all pixels one unit to the right. Then,

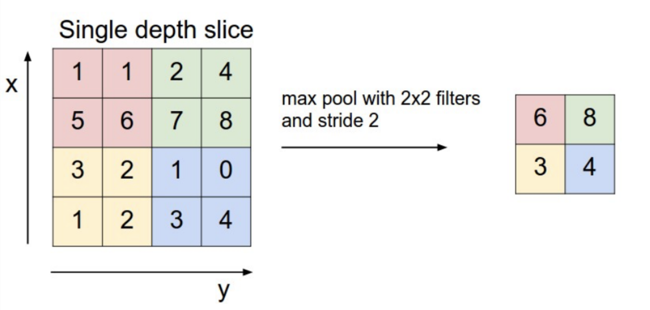
$$
\begin{aligned}
(\mathbf{W} * g(\mathbf{X}))_{ij} = (\mathbf{W} * \mathbf{X}')_{ij} &= \sum_{m=0}^{M} \sum_{n=0}^{N} \sum_{c=0}^{C_{in}} \mathbf{X}'_{(i-m)(j-n)c} \mathbf{W}_{mnc} \\
&= \sum_{m=0}^{M} \sum_{n=0}^{N} \sum_{c=0}^{C_{in}} \mathbf{X}_{(i-1-m)(j-n)c} \mathbf{W}_{mnc} \\
&= (\mathbf{W} * \mathbf{X})_{(i-1)j} \\
&= g((\mathbf{W} * \mathbf{X})_{ij})
\end{aligned}
$$

The neuron will fire at location corresponding to the translated feature.

Pooling

Keeps only the maximum value from a region.

- reduces spatial resolution
- reduces the number of parameters required
- increases invariance to small changes in input



Fully-connected layers

For image classification - stack a shallow standard (dense) neural network to output the class probabilities. It has the usual affine layers of the form:

$$f(\mathbf{X}) = \mathbf{XW} + \mathbf{b}$$

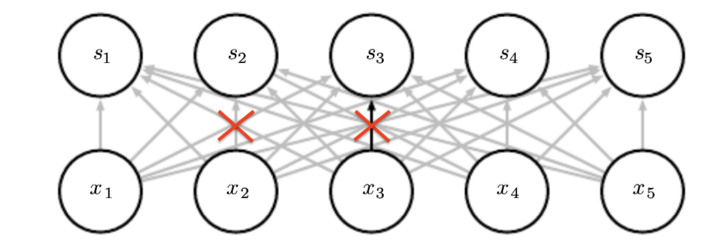
followed by the *softmax activation*:

$$\sigma(\mathbf{X}) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

- it simply normalizes the scores so that they can be interpreted as probabilities.

Dropout

Set every activation to 0 with probability p.



This acts as regularization $\rightarrow$ reduces overfitting, also increases training speed (usually only done for fully-connected layers).

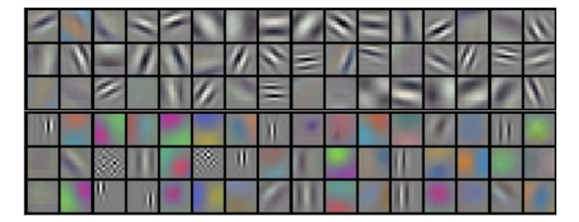
Convolutional neural network - full architecture

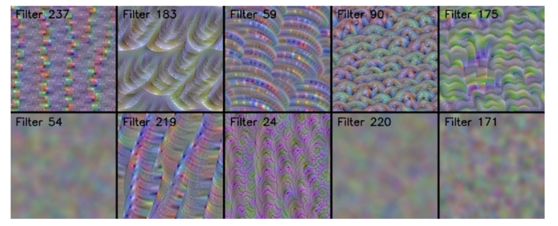
input $\rightarrow$ [**conv** $\rightarrow$ **ReLU** $\rightarrow$ **max pool**] $\times n$ $\rightarrow$ [**FC** $\rightarrow$ **dropout**] $\times m$

Architecture design is tricky - use one of the published architectures, e.g.

- VGG
- Inception v3
- ResNet
- Xception

What does a convnet learn?





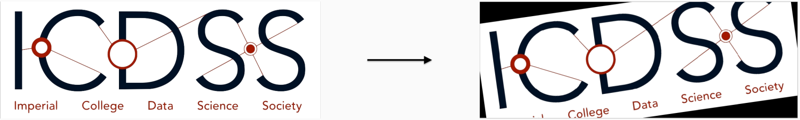
Transfer learning

Features learned by lower layers are very general $\rightarrow$ useful in different problems. Take the learned parameters (conv kernels) from a pre-trained network, 'freeze' them and only train the fully-connected layers.

Data augmentation

A simple way to increase the amount of training data - apply geometrical transforms (rotations, shears, etc.).

Can improve generalization - convnets are invariant to small shifts in the input, but not necessarily other transforms.



Questions?

Codelab

Setup

1. Create a `Github` account.
2. Sign-in `cocalc` using your `Github` credentials.
3. Create a new project in `cocalc`.
4. Clone (green button at top RHS) in `zip` format the **Neural Networks** repository.
5. `Upload` the `zip` file to newly created `cocalc` project.
6. Click on the `zip` file and extract the compressed files.
7. Navigate to the extracted folder `Neural-Networks-master/notebooks/Demo.ipynb`
8. Change the `kernel` by: `Kernel` → `Change Kernel` → `Python 3 (Anaconda)`

References i

cocalc.
Collaborative Calculation in the Cloud, 2018.
Online; accessed 28 Feb 2018; available at
https://cocalc.com/.

Github.
Built for developers, 2018.
Online; accessed 28 Feb 2018; available at
https://github.com.

Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the participants individually and, unless expressly stated to the contrary, are not the opinion or position of the ICDSS, its cosponsors, or its committees. The ICDSS does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented. Attendees should note that sessions are video-recorded and may be published in various media, including print, audio and video formats without further notice.