Deep Learning

Neural Net Basics

Computer Vision
James Hays

Many slides by Marc’Aurelio Ranzato
Outline

• Neural Networks
• Convolutional Neural Networks
• Variants
  • Detection
  • Segmentation
  • Siamese Networks
• Visualization of Deep Networks
Supervised Learning

\[ (x^i, y^i), i = 1 \ldots P \]  training dataset
\[ x^i \]  i-th input training example
\[ y^i \]  i-th target label
\[ P \]  number of training examples

Goal: predict the target label of unseen inputs.
Supervised Learning: Examples

Classification

Denoising

OCR

"dog"

regression

structured prediction
Supervised Deep Learning

Classification

Denoising

OCR

“dog”

“2 3 4 5”
Project 4: Scene Classification with Deep Nets

Dataset

The dataset to be used in this assignment is the 15-scene dataset, containing natural images in 15 possible scenarios like bedrooms and coasts. It was first introduced by Lazebnik et al., 2006 [1]. The images have a typical size of around 200 by 200 pixels, and serve as a good milestone for many vision tasks. A sample collection of the images can be found below:

![Sample images from the 15-scene dataset](image.png)

Figure 1: Example scenes from each of the categories of the dataset.

Download the data (link at the top), unzip it and put the data folder in the proj4 directory.
1 Part 1: SimpleNet

Introduction

In this project, scene recognition with deep learning, we are going to train a simple convolutional neural net from scratch. We’ll be starting with some modification to the dataloader used in this project to include a few extra pre-processing steps. Subsequently, you will define your own model and optimization function. A trainer class will be provided to you, and you will be able to test out the performance of your model with this complete pipeline of classification problem.

Figure 2: The base SimpleNet architecture for Part 1.
Outline

• Neural Networks
• Convolutional Neural Networks
• Variants
  • Detection
  • Segmentation
  • Siamese Networks
• Visualization of Deep Networks
Neural Networks

Assumptions (for the next few slides):
- The input image is vectorized (disregard the spatial layout of pixels)
- The target label is discrete (classification)
Neural Networks: example

Example of a 2 hidden layer neural network (or 4 layer network, counting also input and output).

\[ x \rightarrow \max(0, W^1 x) \rightarrow \max(0, W^2 h^1) \rightarrow W^3 h^2 \rightarrow o \]

- \( x \) input
- \( h^1 \) 1-st layer hidden units
- \( h^2 \) 2-nd layer hidden units
- \( o \) output
**Def.:** *Forward propagation* is the process of computing the output of the network given its input.
Forward Propagation

\[ x \in \mathbb{R}^D \quad W^1 \in \mathbb{R}^{N_1 \times D} \quad b^1 \in \mathbb{R}^{N_1} \quad h^1 \in \mathbb{R}^{N_1} \]

\[ h^1 = \max(0, W^1 x + b^1) \]

- \( W^1 \): 1-st layer weight matrix or weights
- \( b^1 \): 1-st layer biases

The non-linearity \( u = \max(0, v) \) is called \textbf{ReLU} in the DL literature.
Each output hidden unit takes as input all the units at the previous layer: each such layer is called “\textbf{fully connected}”. 
Forward Propagation

\[
\begin{align*}
  h^1 & \in \mathbb{R}^{N_1}  \\
  W^2 & \in \mathbb{R}^{N_2 \times N_1}  \\
  b^2 & \in \mathbb{R}^{N_2}  \\
  h^2 & \in \mathbb{R}^{N_2} \\
  h^2 &= \max (0, W^2 h^1 + b^2) \\
  W^2 & \quad \text{2-nd layer weight matrix or weights} \\
  b^2 & \quad \text{2-nd layer biases}
\end{align*}
\]
Forward Propagation

\[ x \xrightarrow{\text{max} (0, W^1 x)} h^1 \xrightarrow{\text{max} (0, W^2 h^1)} h^2 \xrightarrow{W^3 h^2} o \]

\[ h^2 \in \mathbb{R}^{N_2}, \quad W^3 \in \mathbb{R}^{N_3 \times N_2}, \quad b^3 \in \mathbb{R}^{N_3}, \quad o \in \mathbb{R}^{N_3} \]

\[ o = \max (0, W^3 h^2 + b^3) \]

*W^3* 3-rd layer weight matrix or weights

*b^3* 3-rd layer biases
Alternative Graphical Representation
**Interpretation**

**Question:** Why do we need many layers?

**Answer:** When input has hierarchical structure, the use of a hierarchical architecture is potentially more efficient because intermediate computations can be re-used. DL architectures are efficient also because they use **distributed representations** which are shared across classes.

\[
[0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ ... ] \quad \text{truck feature}
\]

Exponentially more efficient than a 1-of-N representation (a la k-means)
Interpretation

\[
[1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ldots ] \quad \text{motorbike}
\]

\[
[0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ldots ] \quad \text{truck}
\]
Interpretation

- distributed representations
- feature sharing
- compositionality

Lee et al. “Convolutional DBN's ...” ICML 2009
Interpretation

**Question:** What does a hidden unit do?

**Answer:** It can be thought of as a classifier or feature detector.
**Interpretation**

**Question:** What does a hidden unit do?

**Answer:** It can be thought of as a classifier or feature detector.

**Question:** How many layers? How many hidden units?

**Answer:** Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.
Interpretation

**Question:** What does a hidden unit do?

**Answer:** It can be thought of as a classifier or feature detector.

**Question:** How many layers? How many hidden units?

**Answer:** Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

**Question:** How do I set the weight matrices?

**Answer:** Weight matrices and biases are learned. First, we need to define a measure of quality of the current mapping. Then, we need to define a procedure to adjust the parameters.
How Good is a Network?

\[ \mathbf{x} \xrightarrow{\text{max}(0, W^1 \mathbf{x})} h^1 \xrightarrow{\text{max}(0, W^2 h^1)} h^2 \xrightarrow{W^3 h^2} o \xrightarrow{\text{Loss}} \]

\[ y = [0 \ 0 \ldots 0 \ 1 \ 0 \ldots 0] \]

Probability of class \( k \) given input (softmax):

\[ p(c_k = 1 | \mathbf{x}) = \frac{e^{o_k}}{\sum_{j=1}^{c} e^{o_j}} \]

(Per-sample) Loss; e.g., negative log-likelihood (good for classification of small number of classes):

\[ L(\mathbf{x}, y; \theta) = -\sum_j y_j \log p(c_j | \mathbf{x}) \]
Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

\[ \theta^* = \underset{\theta}{\arg\min} \sum_{n=1}^{P} L(x^n, y^n; \theta) \]
Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

$$\theta^* = \arg \min_\theta \sum_{n=1}^{P} L(x^n, y^n; \theta)$$

**Question:** How to minimize a complicated function of the parameters?

**Answer:** Chain rule, a.k.a. **Backpropagation**! That is the procedure to compute gradients of the loss w.r.t. parameters in a multi-layer neural network.

Rumelhart et al. “Learning internal representations by back-propagating..” Nature 1986
Key Idea: Wiggle To Decrease Loss

Let's say we want to decrease the loss by adjusting $W_{i,j}^1$. We could consider a very small $\epsilon = 1e-6$ and compute:

$$L(x, y; \theta)$$
$$L(x, y; \theta \setminus W_{i,j}^1, W_{i,j}^1 + \epsilon)$$

Then, update:

$$W_{i,j}^1 \leftarrow W_{i,j}^1 + \epsilon \text{sgn}(L(x, y; \theta) - L(x, y; \theta \setminus W_{i,j}^1, W_{i,j}^1 + \epsilon))$$
Derivative w.r.t. Input of Softmax

\[ p(c_k = 1 | \mathbf{x}) = \frac{e^{o_k}}{\sum_j e^{o_j}} \]

\[ L(\mathbf{x}, y; \theta) = -\sum_j y_j \log p(c_j | \mathbf{x}) \quad y = [0 \ 0 \ldots 1 \ 0 \ldots 0] \]

By substituting the first formula in the second, and taking the derivative w.r.t. \( o \) we get:

\[ \frac{\partial L}{\partial o} = p(c | \mathbf{x}) - y \]
Given $\frac{\partial L}{\partial o}$ and assuming we can easily compute the Jacobian of each module, we have:

$$\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial W^3}$$

$$\frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2}$$
Suppose \( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \) is a function such that each of its first-order partial derivatives exist on \( \mathbb{R}^n \). This function takes a point \( \mathbf{x} \in \mathbb{R}^n \) as input and produces the vector \( f(\mathbf{x}) \in \mathbb{R}^m \) as output. Then the Jacobian matrix of \( f \) is defined to be an \( m \times n \) matrix, denoted by \( \mathbf{J} \), whose \((i,j)\)th entry is \( \mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j} \), or explicitly

\[
\mathbf{J} = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix} = \begin{bmatrix}
\nabla^T f_1 \\
\vdots \\
\nabla^T f_m
\end{bmatrix} = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}
\]
Backward Propagation

\[ x \xrightarrow{\max(0,W^1x)} h^1 \xrightarrow{\max(0,W^2h^1)} h^2 \xrightarrow{W^3h^2} \frac{\partial L}{\partial o} \]

\[ y \]

Given \( \frac{\partial L}{\partial o} \) and assuming we can easily compute the Jacobian of each module, we have:

\[
\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial W^3} \quad \quad \quad \frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2}
\]

\[ \text{Loss} \]
Backward Propagation

\[ \frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial W^3} \]

\[ \frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2} \]

\[ \frac{\partial L}{\partial W^3} = (p(c|x) - y) h^2^T \]

\[ \frac{\partial L}{\partial h^2} = W^3^T (p(c|x) - y) \]
Backward Propagation

Given $\frac{\partial L}{\partial h^2}$ we can compute now:

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial W^2} \quad \frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial h^1}$$
Backward Propagation

Given $\frac{\partial L}{\partial h^1}$ we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial h^1} \frac{\partial h^1}{\partial W^1}$$
Backward Propagation

**Question:** Does BPROP work with ReLU layers only?

**Answer:** Nope, any a.e. differentiable transformation works.
Backward Propagation

**Question:** Does BPROP work with ReLU layers only?

**Answer:** Nope, any a.e. differentiable transformation works.

**Question:** What's the computational cost of BPROP?

**Answer:** About twice FPROP (need to compute gradients w.r.t. input and parameters at every layer).
Optimization

Stochastic Gradient Descent (on mini-batches):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in (0, 1)$$

Stochastic Gradient Descent with Momentum:

$$\theta \leftarrow \theta - \eta \Delta$$

$$\Delta \leftarrow 0.9 \Delta + \frac{\partial L}{\partial \theta}$$

Note: there are many other variants...
Toy Example: Synthetic Data

- Target
- Before training
- After 1 epoch
- At the end of training

1 input & 1 output
3 hidden layers, 1000 hidden units
Regression of cosine
Toy Example: Synthetic Data

1 input & 1 output
100 hidden units in each layer

output

input

1 hidden layer
2 hidden layers
3 hidden layers
Toy Example: Synthetic Data

1 input & 1 output
3 hidden layers

10 hiddens
100 hiddens
1000 hiddens

output

input
Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips
This all seems pretty complicated. Why are we using Neural Networks? James’s rough assessment:

<table>
<thead>
<tr>
<th>Learning method</th>
<th>Ease of configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>1</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>10</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>10</td>
</tr>
<tr>
<td>Non-linear SVM</td>
<td>5</td>
</tr>
<tr>
<td>Decision Tree or Random Forest</td>
<td>4</td>
</tr>
</tbody>
</table>
This all seems pretty complicated. Why are we using Neural Networks? James’s rough assessment:

<table>
<thead>
<tr>
<th>Learning method</th>
<th>Ease of configuration</th>
<th>Ease of interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Non-linear SVM</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Decision Tree or Random Forest</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
This all seems pretty complicated. Why are we using Neural Networks? James’s rough assessment:

<table>
<thead>
<tr>
<th>Learning method</th>
<th>Ease of configuration</th>
<th>Ease of interpretation</th>
<th>Speed / memory when training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>10</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>10</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Non-linear SVM</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Decision Tree or Random Forest</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
This all seems pretty complicated. Why are we using Neural Networks? James’s rough assessment:

<table>
<thead>
<tr>
<th>Learning method</th>
<th>Ease of configuration</th>
<th>Ease of interpretation</th>
<th>Speed / memory when training</th>
<th>Speed / memory at test time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Non-linear SVM</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Decision Tree or Random Forest</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>
This all seems pretty complicated. Why are we using Neural Networks? James’s rough assessment:

<table>
<thead>
<tr>
<th>Learning method</th>
<th>Ease of configuration</th>
<th>Ease of interpretation</th>
<th>Speed / memory when training</th>
<th>Speed / memory at test time</th>
<th>Accuracy w/ lots of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Non-linear SVM</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Decision Tree or Random Forest</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
This all seems pretty complicated. Why are we using Neural Networks? James’s rough assessment:

<table>
<thead>
<tr>
<th>Learning method</th>
<th>Ease of configuration</th>
<th>Ease of interpretation</th>
<th>Speed / memory when training</th>
<th>Speed / memory at test time</th>
<th>Accuracy w/ lots of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-linear SVM</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decision Tree or Random Forest</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Representation design matters more for all of these.
Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips
Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips
Fully Connected Layer

Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

**STATIONARITY?** Statistics is similar at different locations

Example: 200x200 image
- 40K hidden units
- Filter size: 10x10
- 4M parameters

**Note:** This parameterization is good when input image is registered (e.g., face recognition).
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary):
Convolution with learned kernels
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer

\[
\begin{pmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{pmatrix}
\]
Convolutional Layer

Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
Convolutional Layer

\[ h_j^n = \max \left( 0, \sum_{k=1}^{K} h_{k}^{n-1} \ast w_{kj}^n \right) \]

output feature map
input feature map
kernel

\[ h_1^n \rightarrow h_1^n \rightarrow h_2^n \rightarrow h_2^n \]

Conv. layer
Convolutional Layer

\[ h_j^n = \max(0, \sum_{k=1}^{K} h_{kj}^{n-1} \ast w_{kj}^n) \]

output feature map
input feature map
kernel

\[ h_1^{n-1}, h_2^{n-1}, h_3^{n-1}, h_1^n, h_2^n \]
Convolutional Layer

\[ h_j^n = \max \left( 0, \sum_{k=1}^{K} h_{kj}^{n-1} * w_{kj}^n \right) \]
Convolutional Layer

**Question:** What is the size of the output? What's the computational cost?

**Answer:** It is proportional to the number of filters and depends on the stride. If kernels have size KxK, input has size DxD, stride is 1, and there are M input feature maps and N output feature maps then:
- the input has size M@DxD
- the output has size N@(D-K+1)xDxD
- the kernels have MxNxKxK coefficients (which have to be learned)
- cost: $M \times K \times K \times N \times (D-K+1) \times (D-K+1)$

**Question:** How many feature maps? What's the size of the filters?

**Answer:** Usually, there are more output feature maps than input feature maps. Convolutional layers can increase the number of hidden units by big factors (and are expensive to compute). The size of the filters has to match the size/scale of the patterns we want to detect (task dependent).
Key Ideas

A standard neural net applied to images:
- scales quadratically with the size of the input
- does not leverage stationarity

Solution:
- connect each hidden unit to a small patch of the input
- share the weight across space

This is called: **convolutional layer**.
A network with convolutional layers is called **convolutional network**.

LeCun et al. “Gradient-based learning applied to document recognition” IEEE 1998
Pooling Layer

Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?
Pooling Layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
Pooling Layer: Examples

Max-pooling:

\[ h_j^n(x, y) = \max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y}) \]

Average-pooling:

\[ h_j^n(x, y) = \frac{1}{K} \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y}) \]

L2-pooling:

\[ h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2} \]

L2-pooling over features:

\[ h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2} \]
Pool Layer

**Question:** What is the size of the output? What's the computational cost?

**Answer:** The size of the output depends on the stride between the pools. For instance, if pools do not overlap and have size KxK, and the input has size DxD with M input feature maps, then:
- output is M@(D/K)x(D/K)
- the computational cost is proportional to the size of the input (negligible compared to a convolutional layer)

**Question:** How should I set the size of the pools?

**Answer:** It depends on how much “invariant” or robust to distortions we want the representation to be. It is best to pool slowly (via a few stacks of conv-pooling layers).
Pooling Layer: Receptive Field Size

If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$
Pooling Layer: Receptive Field Size

If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: 

\((P+K-1) \times (P+K-1)\)
Local Contrast Normalization

\[ h^{i+1}(x, y) = \frac{h^i(x, y) - m^i(N(x, y))}{\sigma^i(N(x, y))} \]
Local Contrast Normalization

\[ h^{i+1}(x, y) = \frac{h^i(x, y) - m^i(N(x, y))}{\sigma^i(N(x, y))} \]

We want the same response.
Local Contrast Normalization

\[ h^{i+1}(x, y) = \frac{h^i(x, y) - m^i(N(x, y))}{\sigma^i(N(x, y))} \]

Performed also across features and in the higher layers.

Effects:
- improves invariance
- improves optimization
- increases sparsity

Note: computational cost is negligible w.r.t. conv. layer.
ConvNets: Typical Stage

One stage (zoom)

- Convol.
- LCN
- Pooling

Filter Bank

Rectification + Contrast Normalization

courtesy of K. Kavukcuoglu
ConvNets: Typical Stage

One stage (zoom)

Conceptually similar to: SIFT, HoG, etc.
ConvNets: Typical Architecture

One stage (zoom)

Whole system
ConvNets: Typical Architecture

Whole system

Input Image → 1st stage → 2nd stage → 3rd stage → Fully Conn. Layers → Class Labels

Conceptually similar to:

SIFT → K-Means → Pyramid Pooling → SVM
Lazebnik et al. “...Spatial Pyramid Matching...” CVPR 2006

SIFT → Fisher Vect. → Pooling → SVM