Introduction to Convolutional Neural Networks

Evangelos Kalogerakis
Motivation

modified slides originally by Adam Coates
Motivation

Learning Algorithm

Is this a Coffee Mug?

+ Coffee Mug
- Not Coffee Mug

modified slides originally by Adam Coates
Need stronger feature representations!

Learning Algorithm

Is this a Coffee Mug?

modified slides originally by Adam Coates
From “swallow” to “deep” mappings (networks)

Images, shapes, natural language have **compositional structure and patterns.**

**Deep neural networks** can learn powerful feature representations capturing those.
Classification basics: Logistic Regression

Suppose you want to predict **mug** or **no mug** in an image.

**Output:**  \( y = 1 \) [coffee mug],  \( y = 0 \) [no coffee mug]

**Input:**  \( x = \{x_1, x_2, \ldots\} \) [pixel intensities, gradients, SIFT, etc]
Classification basics: Logistic Regression

Suppose you want to predict **mug** or **no mug** in an image.

**Output:**  \( y = 1 \)  [*coffee mug*],  \( y = 0 \)  [*no coffee mug*]

**Input:**  \( x = \{x_1, x_2, \ldots\} \)  [*pixel intensities, gradients, SIFT, etc*]

Classification function:

\[
P(y = 1 \mid x) = f(x) = \sigma(w \cdot x)
\]

where  \( w \)  is a **weight vector** (parameters)

\[
\sigma(w \cdot x) = \frac{1}{1 + \exp(-w \cdot x)}
\]
Logistic regression: training

Need to estimate parameters $w$ from training data e.g., images of objects $x_i$ and given labels $y_i$ (mugs/no mugs) ($i=1\ldots N$ training images)

Find parameters that maximize probability of training data

$$\max_w \prod_{i=1}^{N} P(y = 1 | x_i)^{[y_i=1]} [1 - P(y = 1 | x_i)]^{[y_i=0]}$$
Logistic regression: training

Need to estimate parameters $w$ from training data e.g., images of objects $x_i$ and given labels $y_i$ (mugs/no mugs) ($i=1...N$ training images)

Find parameters that maximize probability of training data

$$\max_w \prod_{i=1}^N \sigma(w \cdot x_i)^{[y_i=1]} [1 - \sigma(w \cdot x_i)]^{[y_i=0]}$$
Logistic regression: training

Need to estimate parameters $w$ from training data e.g., images of objects $x_i$ and given labels $y_i$ (mugs/no mugs) ($i=1 \ldots N$ training images)

Find parameters that maximize the log probability of training data

$$\max_w \log \left\{ \prod_{i=1}^{N} \sigma(w \cdot x_i)^{[y_i=1]} [1 - \sigma(w \cdot x_i)]^{[y_i=0]} \right\}$$
Logistic regression: training

Need to estimate parameters $\mathbf{w}$ from training data e.g., images of objects $\mathbf{x}_i$ and given labels $y_i$ (mugs/no mugs) ($i=1 \ldots N$ training images)

Find parameters that **maximize the log probability of training data**

$$\max_{\mathbf{w}} \sum_{i=1}^{N} [y_i == 1] \log \sigma(\mathbf{w} \cdot \mathbf{x}_i) + [y_i == 0] \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x}_i))$$
Logistic regression: training

Need to estimate parameters $\mathbf{w}$ from training data e.g., images of objects $\mathbf{x}_i$ and given labels $\mathbf{y}_i$ (mugs/no mugs) ($i=1 \ldots N$ training images)

Find parameters that minimize the negative log probability of training data

$$
\min_{\mathbf{w}} - \sum_{i=1}^{N} [\mathbf{y}_i = 1] \log \sigma(\mathbf{w} \cdot \mathbf{x}_i) + [\mathbf{y}_i = 0] \log (1 - \sigma(\mathbf{w} \cdot \mathbf{x}_i))
$$
Logistic regression: training

Need to estimate parameters $w$ from training data e.g., images of objects $x_i$ and given labels $y_i$ (mugs/no mugs) ($i=1...N$ training images)

This is called \textbf{(negative) log likelihood}:

$$
\min_w \sum_{i=1}^{N} [y_i = 1] \log \sigma(w \cdot x_i) + [y_i = 0] \log (1 - \sigma(w \cdot x_i))
$$

$L(w)$
Logistic regression: training

Need to estimate parameters $w$ from training data e.g., images of objects $x_i$ and given labels $y_i$ (mugs/no mugs) ($i = 1 \ldots N$ training images)

We now have an optimization problem:

$$
\min_w \sum_{i=1}^N \left[ y_i = 1 \right] \log \sigma(w \cdot x_i) + \left[ y_i = 0 \right] \log(1 - \sigma(w \cdot x_i))
$$

$$
\frac{\partial L(w)}{\partial w_d} = \sum_i x_{i,d} \left[y_i - \sigma(w \cdot x_i)\right]
$$

(partial derivative for $d^{th}$ parameter)
How can we minimize/maximize a function?

**Gradient descent:** Given a random initialization of parameters and a step rate $\eta$, update them according to:

$$w_{new} = w_{old} - \eta \nabla L(w)$$

See also **quasi-Newton** and **IRLS** methods
Regularization

**Overfitting**: few training data and number of parameters is large!

Penalize large weights:

\[
\min_w L(w) + \lambda \sum_d w_d^2
\]

Called *ridge regression* (or L2 regularization)
Back to our original example...

Learning Algorithm

Is this a Coffee Mug?

classification boundary

modified slides originally by Adam Coates
How can we learn better feature representations?

Is this a Coffee Mug?

classification boundary

Learning Algorithm

modified slides originally by Adam Coates
"Traditional" recognition pipeline

Fixed/engineered descriptors + trained classifier/regressor
"New" recognition pipeline

**Trained** descriptors + **trained** classifier/regressor

![Diagram](modified slides originally by Adam Coates)
From “swallow” to “deep” mappings (networks)

Logistic regression: output is a **direct function of inputs**. Think of it as a net:

\[ y = f(x) = \sigma(w \cdot x) \]
Neural network

Introduce latent nodes that play the role of learned feature representations.
Neural network

Same as logistic regression but now our output function has **multiple stages** ("layers", "modules").

\[
x \xrightarrow{\sigma(W^{(1)} \cdot x)} h \xrightarrow{\sigma(W^{(2)} \cdot h)} y
\]

Intermediate representation  Prediction

where \( W^{(i)} = \begin{bmatrix} w_1^{(i)} \\ w_2^{(i)} \\ \vdots \\ w_m^{(i)} \end{bmatrix} \)

*modified slides originally by Adam Coates*
Biological Neurons
Analogy with biological networks
Neural network

Stack up several layers:

\[ x_1 x_2 x_3 \ldots x_d \]

\[ h_1 h_2 h_3 \ldots h_m \]

\[ h_1' h_2' h_3' \ldots h_n' \]

\[ y \]

modified slides originally by Adam Coates
Forward propagation

Process to compute output:

\[ x_1 \ x_2 \ x_3 \ \ldots \ x_d \ 1 \]
Forward propagation

Process to compute output:

\[ x \xrightarrow{\sigma(W^{(1)} \cdot x)} h \]
Forward propagation

Process to compute output:

\[ x \xrightarrow{\sigma(W^{(1)} \cdot x)} h \xrightarrow{\sigma(W^{(2)} \cdot h)} h' \]
Forward propagation

Process to compute output:

\[ \mathbf{x} \rightarrow \sigma(\mathbf{W}^{(1)} \cdot \mathbf{x}) \rightarrow \mathbf{h} \rightarrow \sigma(\mathbf{W}^{(2)} \cdot \mathbf{h}) \rightarrow \mathbf{h}' \rightarrow \sigma(\mathbf{W}^{(3)} \cdot \mathbf{h}') \rightarrow \mathbf{y} \]
Multiple outputs

\[
\begin{align*}
  x & \rightarrow \sigma(W^{(1)} \cdot x) \\
  h & \rightarrow \sigma(W^{(2)} \cdot h) \\
  h' & \rightarrow \sigma(W^{(3)} \cdot h') \\
  y & \\
\end{align*}
\]
How can you learn the parameters?

Use a loss function e.g., for classification:

$$L(w) = -\sum_{i=1}^{\text{output } t} \sum [y_{i,t} == 1] \log f_i(x_i) + [y_{i,t} == 0] \log(1 - f_t(x_i))$$

In case of regression i.e., for predicting continuous outputs:

$$L(w) = \sum_{i} \sum_{\text{output } t} [y_{i,t} - f_i(x_i)]^2$$
Backpropagation

For each training example $i$ (omit index $i$ for clarity):

For each output:

$$\delta^{(3)}_t = y_t - f(x)$$

$$\frac{\partial L(w)}{\partial w_{t,n}^{(3)}} = \delta^{(3)}_t h_n$$
Backpropagation

For each training example $i$ (omit index $i$ for clarity):

$$\delta_n^{(2)} = \sigma'(w_n^{(2)} \cdot h) \sum_t w_{t,n}^{(3)} \delta_t^{(3)}$$

Note: $\sigma'(\cdot) = \sigma(\cdot)[1 - \sigma(\cdot)]$

$$\frac{\partial L(w)}{\partial w_{n,m}^{(2)}} = \delta_n^{(2)} h_m$$
Backpropagation

For each training example $i$ (omit index $i$ for clarity):

$$\delta_m^{(1)} = \sigma'(w_m^{(1)} \cdot x) \sum_n w_{n,m}^{(2)} \delta_n^{(2)}$$

$$\frac{\partial L(w)}{\partial w_{m,d}^{(1)}} = \delta_m^{(1)} x_d$$
Is this magic?

All these are derivatives derived analytically using the **chain rule**!

Gradient descent is expressed through **backpropagation of messages** $\delta$ following the structure of the model.
Training algorithm

For each training example [in a batch]

1. **Forward propagation** to compute outputs per layer
2. **Back propagate** messages $\delta$ from top to bottom layer
3. Multiply messages $\delta$ with inputs to compute **derivatives per layer**
4. **Accumulate the derivatives** from that training example

Apply the gradient descent rule
Yet, this does not work so easily...
Yet, this does not work so easily...

• **Non-convex**: Local minima; convergence criteria.

• Optimization becomes difficult with **many layers**.

• Hard to diagnose and **debug malfunctions**.

• **Many things turn out to matter**:
  • Choice of nonlinearities.
  • Initialization of parameters.
  • Optimizer parameters: step size, schedule.
Non-linearities

- **Choice of functions inside network matters.**
  - Sigmoid function yields highly non-convex loss functions
  - Some other choices often used:

  \[
  \begin{align*}
  \tanh(\cdot) & = 1 - \tanh(\cdot)^2 \\
  \text{abs}(\cdot) & = \text{sign}(\cdot) \\
  \text{ReLU}(\cdot) & = \max\{0, \cdot\}
  \end{align*}
  \]

  “Rectified Linear Unit”
  \[\rightarrow\ \text{Increasingly popular.}\]
  \[\text{[Nair & Hinton, 2010]}\]
Initialization

• **Usually small random values.**
  - Try to choose so that typical input to a neuron avoids saturating

• **Initialization schemes for weights used as input to a node:**
  - tanh units: Uniform[-r, r]; sigmoid: Uniform[-4r, 4r].
  - See [Glorot et al., AISTATS 2010]

\[ r = \sqrt{6/(\text{fan-in} + \text{fan-out})} \]

• **Unsupervised pre-training**
Step size

• **Fixed step-size**
  - try many, choose the best...
  - pick size with least test error on a validation set after T iterations

• **Dynamic step size**
  - decrease after T iterations
  - if simply the objective is not decreasing much, cut step by half
Momentum/L2 regularization

Modify stochastic/batch gradient descent:

Before: \( \Delta w = \eta \nabla_w L(w), \quad w = w - \Delta w \)

With momentum: \( \Delta w = \mu \Delta w_{\text{previous}} + \eta \nabla_w L(w), \quad w = w - \Delta w \)

“Smooth” estimate of gradient from several steps of gradient descent:

• High-curvature directions cancel out, low-curvature directions “add up” and accelerate.

Other techniques: Adagrad, Adadelta...

Add L2 regularization to the loss function:

\[ \Delta w = \eta \nabla_w (L(w) + \lambda \| w \|_2) \]
Yet, things will not still work well!
Main problem

- Extremely large number of connections.
- More parameters to train.
- Higher computational expense.
Local connectivity

Reduce parameters with local connections!
Neurons as convolution filters

Now think of neurons as convolutional filters acted on small adjacent (possibly overlapping) windows.

Window size is called “receptive field” size and spacing is called “step” or “stride”.

modified slides originally by Adam Coates
Extract repeated structure

Apply the **same filter** (weights) throughout the image

Dramatically reduces the number of parameters
Convolution reminder [animated]

(red numbers are filter values)

Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution
Can have many filters!

Response per pixel $p$, per filter $f$ for a transfer function $g$:

$$h_{p,f} = g(\mathbf{w}_f \cdot \mathbf{x}_p)$$

modified slides originally by Adam Coates
Example: multiple 3D filters working on multiple channels

<table>
<thead>
<tr>
<th>Input Volume (+pad 1) (7x7x3)</th>
<th>Filter W0 (3x3x3)</th>
<th>Filter W1 (3x3x3)</th>
<th>Output Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w0[:, :, 0]</td>
<td>w1[:, :, 0]</td>
<td>o[:, :, 0]</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>1 -1 -1</td>
<td>6 2 3</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>-1 1 0</td>
<td>2 -3 1</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>0 0 -1</td>
<td>4 1 -3</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>-1 1 0</td>
<td>-3 -2 -3</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>-1 1 0</td>
<td>-10 -4 -1</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
<td>1 1 1</td>
<td>-8 0 -2</td>
</tr>
</tbody>
</table>
Pooling

Apart from hidden layers dedicated to convolution, we can have layers dedicated to extract **locally invariant** descriptors

Max pooling:
\[ h_{p',f} = \max_p(x_p) \]

Mean pooling:
\[ h_{p',f} = \text{avg}_p(x_p) \]

Fixed filter (e.g., Gaussian):
\[ h_{p',f} = \mathbf{w}_{\text{gaussian}} \cdot x_p \]

Progressively reduce the resolution of the image, so that the next convolutional filters are applied on larger scales

[Scherer et al., ICANN 2010]
[Boureau et al., ICML 2010]
A mini convolutional neural network

Interchange convolutional and pooling (subsampling) layers.

In the end, **unwrap all feature maps into a single feature vector** and pass it through the classical (**fully connected**) neural network.

Source: http://deeplearning.net/tutorial/lenet.html
LeNet

Initial architecture from LeCun et al., 1998:
Convolutional layers with tanh non-linearity
Max-pooling layers
Stochastic gradient descent
Applied to digit recognition
AlexNet

Proposed architecture from Krizhevsky et al., NIPS 2012:
Convolutional layers with Rectified linear units
Max-pooling layers
Stochastic gradient descent on GPU with momentum, L2 regularization, dropout
Applied to image classification (ImageNet competition – top runner & game changer)
Application: ImageNet classification

Top result in ILSVRC 2012  [~85%, Top-5 accuracy]

Krizhevsky et al., NIPS 2012
Learned representations

Think of convolution filters as optimized feature templates capturing various hierarchical patterns (edges, local structures, sub-parts, parts...)

see Matthew D. Zeiler and Rob Fergus, Visualizing and Understanding Convolutional Networks, 2014
Multi-view CNNs for shape analysis

$\textbf{CNN}_1$: a ConvNet extracting image features

*Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller
*Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015*
All image features are combined by view pooling...

*View pooling*: element-wise max-pooling across all views

*Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller, Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015*
... then passed through CNN₂ and to generate final prediction

CNN₂: a second ConvNet producing shape descriptors

*Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller, Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015*
Train on image datasets!

CNNs pre-trained on ImageNet (leverage large image datasets for training shape analysis techniques!)

Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller
Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015
... and then fine-tune on 3D datasets!

Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller
Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015
Volumetric CNNs

Key idea: represent a shape as a volumetric image with binary voxels.

Learn filters operating on these volumetric data.

3D ShapeNets: A Deep Representation for Volumetric Shapes, 2015*
Volumetric CNNs

3D ShapeNets: A Deep Representation for Volumetric Shapes, 2015
Comparison

Shape retrieval evaluation in ModelNet40:

Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller, Multi-view Convolutional Neural Networks for 3D Shape Recognition, 2015
Summary

CNNs can learn **highly discriminative, hierarchical, powerful feature representations** for image and shape analysis.

Deep learning and CNNs have revolutionized computer vision, robotics, NLP, machine learning: solve hard tasks, achieve performance comparable to humans.

*Why do we still use far-from-optimal, ‘old-style’ descriptors in CG?*

Deep learning has also shown very promising results in image and shape synthesis [see Data-Driven Shape Analysis and Processing, EG’16 STAR report].