Introduction to Machine Learning

Deep Learning

Barnabás Póczos
Credits

Many of the pictures, results, and other materials are taken from:

- Ruslan Salakhutdinov
- Joshua Bengio
- Geoffrey Hinton
- Yann LeCun
Contents

- Definition and Motivation
- Deep architectures
  - Convolutional networks
- Applications
**Defintion:** Deep architectures are composed of *multiple levels* of non-linear operations, such as neural nets with many hidden layers.

$$y_i = \tanh\left(\sum_j W_{ij} x_j + W_{i0}\right)$$

$$y_i = \sigma\left(\sum_j W_{ij} x_j + W_{i0}\right)$$

$$\frac{1}{1 + \exp[-(\sum_j W_{ij} x_j + W_{i0})]}$$

**Input layer**

$$\sum w_i x_i$$

$$net = \sum_{i=0}^{n} w_i x_i$$

$$o = \sigma(net) = \frac{1}{1 + e^{-net}}$$
Goal: Deep learning methods aim at

- learning *feature hierarchies*
- where features from higher levels of the hierarchy are formed by lower level features.

edges, local shapes, object parts

Low level representation

Figure is from Yoshua Bengio
Some complicated functions cannot be efficiently represented (in terms of number of tunable elements) by architectures that are too shallow.

Deep architectures might be able to represent some functions otherwise not efficiently representable.

More formally:

Functions that can be compactly represented by a depth $k$ architecture might require an exponential number of computational elements to be represented by a depth $k - 1$ architecture.

The consequences are

- **Computational**: We don’t need exponentially many elements in the layers
- **Statistical**: poor generalization may be expected when using an insufficiently deep architecture for representing some functions.
The Polynomial circuit:

\[(x_1x_2)(x_2x_3) + (x_1x_2)(x_3x_4) + (x_2x_3)^2 + (x_2x_3)(x_3x_4)\]
Deep Convolutional Networks
Deep Convolutional Networks

Compared to standard feedforward neural networks with similarly-sized layers,

- CNNs have much fewer connections and parameters
- and so they are easier to train,
- while their theoretically-best performance is likely to be only slightly worse.

LeNet 5

Convolution

Continuous functions:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \, d\tau = \int_{-\infty}^{\infty} f(t - \tau) g(\tau) \, d\tau.$$ 

Discrete functions:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m] = \sum_{m=-\infty}^{\infty} f[n - m] g[m]$$

If discrete g has support on \{-M,...,M\}:

$$(f * g)[n] = \sum_{m=-M}^{M} f[n - m] g[m]$$
Convolution

If discrete $g$ has support on \{-M,\ldots,M\}:

$$(f \ast g)[n] = \sum_{m=-M}^{M} f[n-m]g[m]$$

**Product of polynomials**

kernel of the convolution

$$[1, 2] \ast [3, 2, 5] = (x + 2) \ast (3x^2 + 2x + 5) = 3x^3 + 8x^2 + 9x + 10$$

$$[1 \times 3 + 2 \times 0, 1 \times 2 + 2 \times 3, 1 \times 5 + 2 \times 2, 1 \times 0 + 2 \times 5] = [3, 8, 9, 10]$$
2-Dimensional Convolution

\[ y_{00} = x_{00}w_{00} + x_{01}w_{01} + x_{10}w_{10} + x_{11}w_{11} \]
2-Dimensional Convolution

\[ y_{01} = x_{01}w_{00} + x_{02}w_{01} + x_{12}w_{10} + x_{12}w_{11} \]
2-Dimensional Convolution

\[ f[x, y] * g[x, y] = \sum_{n_1 = -\infty}^{\infty} \sum_{n_2 = -\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2] \]

https://graphics.stanford.edu/courses/cs178/applets/convolution.html
LeNet 5, LeCun 1998

- **Input:** 32x32 pixel image. Largest character is 20x20 (All important info should be in the center of the receptive fields of the highest level feature detectors)
- **Cx:** Convolutional layer (C1, C3, C5)
- **Sx:** Subsample layer (S2, S4)
- **Fx:** Fully connected layer (F6)
- Black and White pixel values are normalized:
  E.g. White = -0.1, Black = 1.175 (Mean of pixels = 0, Std of pixels = 1)
Convolutional Layer

layer m-1

hidden layer m
LeNet 5, Layer C1

C1: Convolutional layer with 6 feature maps of size 28x28.

Each unit of C1 has a 5x5 receptive field in the input layer.

- Topological structure
- Sparse connections
- Shared weights

(5*5+1)*6=156 parameters to learn

Connections: (5*5+1)*28*28*6=122304

If it was fully connected, we had (32*32+1)*(28*28)*6 parameters = connections.
S2: Subsampling layer with 6 feature maps of size 14x14

2x2 nonoverlapping receptive fields in C1

\[ S2^k_{i,j} = \tanh(w_1^k \sum_{s,t=0}^{1} C1^k_{2i-s,2j-t} + w_2^k). \]

Layer S2: 6*2=12 trainable parameters.

Connections: 14*14*(2*2+1)*6=5880
LeNet 5, Layer C3

- C3: Convolutional layer with 16 feature maps of size 10x10
- Each unit in C3 is connected to several 5x5 receptive fields at identical locations in S2

Layer C3:

1516 trainable parameters.

\[(3\times5\times5+1)\times6+(4\times5\times5+1)\times9+(6\times5\times5+1)\]

Connections: 151600

\[(3\times5\times5+1)\times6\times10\times10+(4\times5\times5+1)\times9\times10\times10+(6\times5\times5+1)\times10\times10\]
LeNet 5, Layer S4

- S4: Subsampling layer with 16 feature maps of size 5x5
- Each unit in S4 is connected to the corresponding 2x2 receptive field at C3

\[
S^k_{ij} = \tanh(w^k_1 \sum_{s,t=0}^{1} C^k_{2i-s,2j-t} + w^k_2).
\]

Layer S4: 16*2=32 trainable parameters.
Connections: 5*5*(2*2+1)*16=2000
- C5: Convolutional layer with 120 feature maps of size 1x1
- Each unit in C5 is connected to all 16 5x5 receptive fields in S4

Layer C5: $120 \times (16 \times 25 + 1) = 48120$ trainable parameters and connections (Fully connected)
**Layer F6:** 84 fully connected units. $84 \times (120+1) = 10164$ trainable parameters and connections.

**Output layer:** 10RBF (One for each digit)

$$y_i = \sum_{j=1}^{84} (x_j - w_{ij})^2, \quad i = 1, \ldots, 10.$$ 

84 = 7x12, stylized image.

84 parameters, 84*10 connections

**Weight update:** Backpropagation
MINIST Dataset

- 60,000 original datasets
  - Test error: 0.95%
- 540,000 artificial distortions
- + 60,000 original
  - Test error: 0.8%
Misclassified examples

True label -> Predicted label

| 4 → 6 | 3 → 5 | 8 → 2 | 2 → 1 | 5 → 3 | 4 → 8 | 2 → 8 | 3 → 5 | 6 → 5 | 7 → 3 |
| 9 → 4 | 8 → 0 | 7 → 8 | 5 → 3 | 8 → 7 | 0 → 6 | 3 → 7 | 2 → 7 | 8 → 3 | 9 → 4 |
| 8 → 2 | 5 → 3 | 4 → 8 | 3 → 9 | 6 → 0 | 9 → 8 | 4 → 9 | 6 → 1 | 9 → 4 | 9 → 1 |
| 9 → 4 | 2 → 0 | 6 → 1 | 3 → 5 | 3 → 2 | 9 → 5 | 6 → 0 | 6 → 0 | 6 → 0 | 6 → 8 |
| 4 → 6 | 7 → 3 | 9 → 4 | 4 → 6 | 2 → 7 | 9 → 7 | 4 → 3 | 9 → 4 | 9 → 4 | 9 → 4 |
| 8 → 7 | 4 → 2 | 8 → 4 | 3 → 5 | 8 → 4 | 6 → 5 | 8 → 5 | 3 → 8 | 3 → 8 | 9 → 8 |
| 1 → 5 | 9 → 8 | 6 → 3 | 0 → 2 | 6 → 5 | 9 → 5 | 0 → 7 | 1 → 6 | 4 → 9 | 2 → 1 |
| 2 → 8 | 8 → 5 | 4 → 9 | 7 → 2 | 7 → 2 | 6 → 5 | 9 → 7 | 6 → 1 | 5 → 6 | 5 → 0 |
| 4 → 9 | 2 → 8 |
LeNet 5 in Action
LeNet 5, Shift invariance
LeNet 5, Noise resistance
LeNet 5, Unusual Patterns
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton,
Advances in Neural Information Processing Systems 2012

Alex Net
ImageNet Classification error throughout years and groups

- Deeper Network in Network
- Deep
- DNN First Blood

ImageNet

- 15M images
- 22K categories
- Images collected from Web
- Human labelers (Amazon’s Mechanical Turk crowd-sourcing)
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
  - 1K categories
  - 1.2M training images (~1000 per category)
  - 50,000 validation images
  - 150,000 testing images

- RGB images
- Variable-resolution, but this architecture scales them to 256x256 size
Classification goals:

- Make 1 guess about the label (Top-1 error)
- make 5 guesses about the label (Top-5 error)
The Architecture

Typical nonlinearities: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$f(x) = (1 + e^{-x})^{-1}$ (logistic function)

Here, however, Rectified Linear Units (ReLU) are used: $f(x) = \max(0, x)$

**Empirical observation:** Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units

A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line)
The first convolutional layer filters the 224×224×3 input image with 96=2*48 kernels of size 11×11×3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in the kernel map. 224/4=56
The pooling layer: form of non-linear down-sampling. Max-pooling partitions the input image into a set of rectangles and, for each such sub-region, outputs the maximum value.
The Architecture

- Trained with stochastic gradient descent
- on two NVIDIA GTX 580 3GB GPUs
- for about a week

- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- 5 convolutional layer, 3 fully connected layer
- Final feature layer: 4096-dimensional

- Rectified Linear Units, overlapping pooling, dropout trick
- Randomly extracted 224x224 patches for more data
Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations.

We employ two distinct forms of data augmentation:

- image translation
- horizontal reflections
- changing RGB intensities
**Dropout**: set the output of each hidden neuron to zero w.p. 0.5.

- The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in backpropagation.
- So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.
- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.
- It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
- Without dropout, our network exhibits substantial overfitting.
- Dropout roughly doubles the number of iterations required to converge.
96 convolutional kernels of size $11\times11\times3$ learned by the first convolutional layer on the $224\times224\times3$ input images.

The top 48 kernels were learned on GPU1 while the bottom 48 kernels were learned on GPU2

Looks like Gabor wavelets, ICA filters...
Results on the test data:
  top-1 error rate: 37.5%
  top-5 error rate: 17.0%

ILSVRC-2012 competition:
  15.3% classification error
  2nd best team: 26.2% classification error
Results

<table>
<thead>
<tr>
<th>mite</th>
<th>container ship</th>
<th>motor scooter</th>
<th>leopard</th>
</tr>
</thead>
<tbody>
<tr>
<td>mite</td>
<td>container ship</td>
<td>motor scooter</td>
<td>leopard</td>
</tr>
<tr>
<td>black widow</td>
<td>lifeboat</td>
<td>go-kart</td>
<td>jaguar</td>
</tr>
<tr>
<td>cockroach</td>
<td>amphibian</td>
<td>moped</td>
<td>cheetah</td>
</tr>
<tr>
<td>tick</td>
<td>fireboat</td>
<td>bumper car</td>
<td>snow leopard</td>
</tr>
<tr>
<td>starfish</td>
<td>drilling platform</td>
<td>golf cart</td>
<td>Egyptian cat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grille</th>
<th>mushroom</th>
<th>cherry</th>
<th>Madagascar cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>convertible</td>
<td>agaric</td>
<td>dalmatian</td>
<td>squirrel monkey</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>grape</td>
<td>spider monkey</td>
</tr>
<tr>
<td>pickup</td>
<td>jelly fungus</td>
<td>elderberry</td>
<td>titi</td>
</tr>
<tr>
<td>beach wagon</td>
<td>gill fungus</td>
<td>currant</td>
<td>indri</td>
</tr>
<tr>
<td>fire engine</td>
<td>dead-man's-fingers</td>
<td>howler monkey</td>
<td></td>
</tr>
</tbody>
</table>
Results: Image similarity

six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.
Thanks for your Attention! 😊