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

\[ \{(x^i, y^i), \, i = 1 \ldots P\} \quad \text{training dataset} \]

\( x^i \quad \text{i-th input training example} \)

\( y^i \quad \text{i-th target label} \)

\( P \quad \text{number of training examples} \)

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

Classification

Denoising

OCR

“dog”

classification

regression

structured prediction

“2 3 4 5”
Supervised Deep Learning

Classification

Denoising

OCR

“dog”

“2 3 4 5”
Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips
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)

Q**uestion**: what class of functions shall we consider to map the input into the output?

A**nswer**: composition of simpler functions.

F**ollow-up questions**: Why not a linear combination? What are the “simpler” functions? What is the interpretation?

A**nswer**: later...
Neural Networks: example

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

- **x** input
- **h^1** 1-st layer hidden units
- **h^2** 2-nd layer hidden units
- **o** output

Example of a 2 hidden layer neural network (or 4 layer network, counting also input and 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 **ReLU** in the DL literature. Each output hidden unit takes as input all the units at the previous layer: each such layer is called “fully connected”.
Forward Propagation

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

\[ h^2 = \max(0, W^2 h^1) \]

\[ W^3 h^2 \]

\[ h^1 \in \mathbb{R}^{N_1}, \quad W^2 \in \mathbb{R}^{N_2 \times N_1}, \quad b^2 \in \mathbb{R}^{N_2}, \quad h^2 \in \mathbb{R}^{N_2} \]

\[ h^2 = \max(0, W^2 h^1 + b^2) \]

- \( W^2 \) 2-nd layer weight matrix or weights
- \( b^2 \) 2-nd layer biases
Forward Propagation

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

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

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

\[ W^3 \quad \text{3-rd layer weight matrix or weights} \]

\[ b^3 \quad \text{3-rd layer biases} \]
Alternative Graphical Representation

\[ h^k \xrightarrow{\max(0, W^{k+1} h^k)} h^{k+1} \]

\[ h^k \xrightarrow{W^{k+1}} h^{k+1} \]
**Interpretation**

**Question:** Why can't the mapping between layers be linear?

**Answer:** Because composition of linear functions is a linear function. Neural network would reduce to (1 layer) logistic regression.

**Question:** What do ReLU layers accomplish?

**Answer:** Piece-wise linear tiling: mapping is locally linear.
**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.

\[
[0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ldots ] \]

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 0 1 ... ]

motorbike

[0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 ... ]

truck
Interpretation

- prediction of class

• distributed representations
• feature sharing
• compositionality

high-level parts

mid-level parts

low level parts

Input image

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.

**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?**

The network takes input $x$ and processes it through $h^1$, $h^2$, and finally outputs $o$. The output is fed into a loss function $\text{Loss}$.

The probability of class $k$ given input $x$ (softmax):

$$p(c_k = 1 | x) = \frac{e^{o_k}}{\sum_c e^{o_j}}$$

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

$$L(x, y; \theta) = -\sum_j y_j \log p(c_j | x)$$
Training

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:

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

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

\[ L(x, y; \theta) = -\sum_j y_j \log p(c_j | x) \quad y = \left[ \begin{array}{c} 1 \ 0 \ \cdots \ 0 \ 1 \ 0 \ \cdots \ 0 \end{array} \right] \]

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

\[ \frac{\partial L}{\partial o} = p(c | x) - y \]
Backward Propagation

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

\[ \frac{\partial L}{\partial o} \]

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

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

\[ \frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2} \]
Given $\frac{\partial L}{\partial o}$ and assuming we can easily compute the Jacobian of each module, we have:

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

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

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

\[
\frac{\partial L}{\partial h^2} = W^3^T (p(c|x) - y)
\]
Given \( \frac{\partial L}{\partial h^2} \) we can compute now:

\[
\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial W^2} \\
\frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial h^1}
\]
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.

**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).

**Note:** FPROP and BPROP are dual of each other. E.g.,:

<table>
<thead>
<tr>
<th></th>
<th>FPROP</th>
<th>BPROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>![Diagram](FPROP SUM)</td>
<td>![Diagram](BPROP SUM)</td>
</tr>
<tr>
<td>COPY</td>
<td>![Diagram](FPROP COPY)</td>
<td>![Diagram](BPROP COPY)</td>
</tr>
</tbody>
</table>
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...
Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips
Hollow Face Illusion

• https://en.wikipedia.org/wiki/Hollow-Face_illusion
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):
Convolutions 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
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

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

output feature map
input feature map
kernel
Conv. layer

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

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

output feature map

input feature map

kernel

Ranzato
Convolutional Layer

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

output feature map

input feature map

kernel
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 MxDxD
- the output has size N@(D-K+1)x(D-K+1)
- the kernels have MxNxKxK coefficients (which have to be learned)
- cost: $M*K*K*N*(D-K+1)*(D-K+1)$

**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.
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:

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

Average-pooling:

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

L2-pooling:

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

L2-pooling over features:

\[ h^n_j(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2} \]
Pooling 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 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)x(P+K-1)
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 → Pooling

courtesy of K. Kavukcuoglu
ConvNets: Typical Stage

One stage (zoom)

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

One stage (zoom)

Whole system

Input Image

1st stage  2nd stage  3rd stage

Fully Conn. Layers

Class Labels
ConvNets: Typical Architecture

Whole system

Input Image \rightarrow 1^{st} \text{ stage} \rightarrow 2^{nd} \text{ stage} \rightarrow 3^{rd} \text{ stage} \rightarrow \text{Fully Conn. Layers} \rightarrow \text{Class Labels}

Conceptually similar to:

SIFT \rightarrow \text{K-Means} \rightarrow \text{Pyramid Pooling} \rightarrow \text{SVM}
Lazebnik et al. “...Spatial Pyramid Matching...” CVPR 2006

SIFT \rightarrow \text{Fisher Vect.} \rightarrow \text{Pooling} \rightarrow \text{SVM}
ConvNets: Training

All layers are differentiable (a.e.).
We can use standard back-propagation.

Algorithm:
  Given a small mini-batch
  - F-PROP
  - B-PROP
  - PARAMETER UPDATE
Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips
Dataset: ImageNet 2012

Deng et al. “Imagenet: a large scale hierarchical image database” CVPR 2009
Examples of hammer:
Architecture for Classification

Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012
Architecture for Classification

Total nr. params: 60M

- 4M LINEAR (4M)
- 16M FULLY CONNECTED (16M)
- 37M FULLY CONNECTED (37M)
- 442K MAX POOLING (74M)
- 1.3M CONV (224M)
- 884K CONV (149M)
- 307K MAX POOLING (223M)
- 35K LOCAL CONTRAST NORM CONV (105M)

Total nr. flops: 832M

Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012
Optimization

SGD with momentum:
- Learning rate = 0.01
- Momentum = 0.9

Improving generalization by:
- Weight sharing (convolution)
- Input distortions
- Dropout = 0.5
- Weight decay = 0.0005
Results: ILSVRC 2012

**TASK 1 - CLASSIFICATION**

- CNN
- SIFT+FV
- SVM1
- SVM2
- NCM

**Error %**
- 15
- 20
- 25
- 30
- 35

**TASK 2 - DETECTION**

- CNN
- DPM-SVM1
- DPM-SVM2

**Error %**
- 10
- 20
- 30
- 40
- 50

Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012
CONV NETS: EXAMPLES

- Object detection

Szegedy et al. “DNN for object detection” NIPS 2013
CONV NETS: EXAMPLES

- Face Verification & Identification