Deep Learning 2
Neural Net Basics

Computer Vision
James Hays

Many slides by Marc’Aurelio Ranzato
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

classification

regression

structured prediction
Supervised Deep Learning

Classification

Denoising

OCR

“dog”

“2345”
Outline

- Supervised Neural Networks
- Convolutional Neural Networks
- Examples
- Tips
Neural Networks: example

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

\[\begin{align*}
x & \in \mathbb{R}^D \\
W^1 & \in \mathbb{R}^{N_1 \times D} \\
b^1 & \in \mathbb{R}^{N_1} \\
h^1 & \in \mathbb{R}^{N_1}
\end{align*}\]

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

\[W^1\] 1-st layer weight matrix or weights

\[b^1\] 1-st layer biases

The non-linearity \(u = \text{max}(0, v)\) 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}”.

\[\text{Ranzato}\]
Forward Propagation

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

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

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

- \[ W^2 \quad \text{2-nd layer weight matrix or weights} \]
- \[ b^2 \quad \text{2-nd layer biases} \]
Forward Propagation

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

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

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

- \( W^3 \): 3-rd layer weight matrix or weights
- \( b^3 \): 3-rd layer biases
Alternative Graphical Representation

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

\[ h^k \xrightarrow{W^{k+1}} h^{k+1} \]
How Good is a Network?

\[ y = \begin{bmatrix} 1 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \end{bmatrix} \]

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):

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

Ranzato
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 $W_{i, j}$.
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

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

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

\[ y = [ \hat{0} \ 0 \ \ldots \ 0 \ \hat{1} \ 0 \ \ldots \ 0 ] \]

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

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

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

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

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

$$\frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2}$$

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

$$\frac{\partial L}{\partial h^2} = W^{3T} (p(c|x) - y)_{23}$$
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:

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

Conv. layer

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

\[
h_j^n = \max(0, \sum_{k=1}^K h^{n-1}_k * w^n_{kj})
\]

output feature map

input feature map

kernel
\[ h_j^n = \max (0, \sum_{k=1}^{K} h_{k}^{n-1} \ast w_k^j) \]
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.

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:

\[ h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2} \]
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 $K \times K$, and the input has size $D \times D$ with $M$ input feature maps, then:
- output is $M \times (D/K) \times (D/K)$
- 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).
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: $(P+K-1) \times (P+K-1)$
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)
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
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 → 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
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
CONV NETS: EXAMPLES

- OCR / House number & Traffic sign classification

Ciresan et al. “MCDNN for image classification” CVPR 2012
Jaderberg et al. “Synthetic data and ANN for natural scene text recognition” arXiv 2014
CONV NETS: EXAMPLES

- Scene Parsing

Farabet et al. “Learning hierarchical features for scene labeling” PAMI 2013
Pinheiro et al. “Recurrent CNN for scene parsing” arxiv 2013
CONV NETS: EXAMPLES

- Face Verification & Identification

Dataset: ImageNet 2012

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

LINEAR
FULLY CONNECTED
FULLY CONNECTED
MAX POOLING
CONV
CONV
CONV
MAX POOLING
LOCAL CONTRAST NORM
CONV
MAX POOLING
LOCAL CONTRAST NORM
CONV

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

Total nr. params: 60M

4M
LINEAR

16M
FULLY CONNECTED

37M
FULLY CONNECTED

MAX POOLING

442K
CONV

1.3M
CONV

884K
CONV

MAX POOLING

LOCAL CONTRAST NORM

307K
CONV

MAX POOLING

LOCAL CONTRAST NORM

35K
CONV

Total nr. flops: 832M

4M

16M

37M

74M

224M

149M

223M

105M

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 %**

**TASK 2 - DETECTION**

- CNN
- DPM-SVM1
- DPM-SVM2

**Error %**

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

- Object detection

Szegedy et al. “DNN for object detection” NIPS 2013