Deep-Learning: general principles + Convolutional Neural Networks

Pr. Fabien Moutarde
Center for Robotics
MINES ParisTech
PSL Université Paris

Fabien.Moutarde@mines-paristech.fr
http://people.mines-paristech.fr/fabien.moutarde

Acknowledgements

During preparation of these slides, I got inspiration and borrowed some slide content from several sources, in particular:

- Yann LeCun + MA Ranzato: slides on « Deep Learning » from the corresponding course at NYU

- Hinton+Bengio+LeCun: slides of the NIPS’2015 tutorial on Deep Learning

- Fei-Fei Li + A.Karpathy + J.Johnson: Stanford course lecture slides on « Convolutional Neural Networks »
Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• Transfer Learning
• Object localization and Semantic segmentation

Deep-Learning recent breakthroughs

• Very significant improvement over State-of-the-Art in Pattern Recognition / Image Semantic Analysis:
  • won many vision pattern recognition competitions (OCR, TSR, object categorization, facial expression,...)
  • deployed in photo-tagging by Facebook, Google, Baidu,...

• Similar dramatic progress in Speech recognition + Natural Language Processing (NLP)
Examples of Deep-Learning important applications


Video analysis for self-driving cars

Examples of Deep-Learning important applications


Deep-Reinforcement Learning for end-to-end driving

Real-time posture estimation by Deep-Learning


Is Deep-Learning « Large-Scale »?

Big and/or « Fat » data

\[
\begin{array}{|c|c|}
\hline
\text{d dimensions} & \text{t tasks} \\
\hline
\text{n samples} & \text{X} & \text{Y} \\
\hline
\end{array}
\]

Deep-Learning: Large MODELS

State-of-the-Art Convolutional Neural Networks contain > 100 layers, millions of parameters
Importance of training data!

Dramatic recent progresses in image classification and visual object categorization not only due to Deep-Learning and convNets:

it was made possible largely thanks to ImageNet dataset, which is a HUGE collection of labelled general-purpose images (1000 categories, > 1 million examples)

Most powerful convNets have been trained on this huge dataset!

What is Deep-Learning?

Learning a **hierarchy** of increasingly abstract **representations**

Increasing level of abstraction
Each stage ~ trainable feature transform

Image recognition
Pixel → edge → texton → motif → part → object

Speech
Sample → spectral band → … → phoneme → word

Text
Character → word → word group → clause → sentence → story

[Figure from Goodfellow]
Importance of « features » in classical Machine-Learning

Examples of *hand-crafted* features

**Haar features**

**Control-points features**

**HoG** (Histogram of Gradients)

Why features should be learnt?

Real data examples for a given task are usually not spreaded everywhere in input space, but rather clustered on a low-dimension « manifold »

Example: Face images of 1000x1000 pixels ➔ « raw » examples are vectors in $\mathbb{R}^{1000000}$ !!

- **BUT:**
  - position = 3 cartesian coord
  - orientation 3 Euler angles
  - 50 muscles in face
  - Luminosity, color

➔ Set of all images of ONE person has ≤ 69 dim

➔ Examples of face images of 1 person *are all in a LOW-dim manifold inside a HUGE-dim space*
Good features
~ « mapping » on manifold

Features learning
(before Deep-Learning)

Speech recognition: early 90's – 2011

- MFCC (fixed)
- Mix of Gaussians (unsupervised)
- Classifier (supervised)

Object Recognition: 2006 - 2012

- SIFT (fixed)
- HoG
- K-means Sparse Coding (unsupervised)
- Pooling
- Classifier (supervised)
Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• Transfer Learning
• Object localization and Semantic segmentation

Convolutional Neural Networks (CNN, or ConvNet)

• Proposed in 1998 by Yann LeCun (french prof. @ NYU, now also AI research director of Facebook)

• For inputs with correlated dims (2D image, 1D signal,...)
• Supervised learning
ConvNets (2)

- Recently won many vision pattern recognition competitions/challenges (OCR, TSR, object categorization, facial expression, …)
- Deployed in photo-tagging by Facebook, Google, Baidu, …
- Also used in real-time video analysis for self-driving cars

Short reminder on what is a (multi-layer) Neural Network

For “Multi-Layer Perceptron” (MLP), neurons type generally “summatim with sigmoid activation”
Reminder on artificial “neurons”

**PRINCIPLE**

\[ O_j = f \left( W_{0j} + \sum_{i=1}^{n} W_{ij} e_i \right) \]

where \( W_{0j} = \text{"bias"} \)

**ACTIVATION FUNCTIONS**

- **Threshold** (Heaviside or sign) \( \rightarrow \text{binary neurons} \)
- **Sigmoid** (logistic or tanh) \( \rightarrow \text{most common for MLPs} \)
- **Identity** \( \rightarrow \text{linear neurons} \)
- **ReLU** (Rectified Linear Unit)
- **Saturation**
- **Gaussian**

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation
Convolution: sliding a 3D filter over image

32x32x3 image

5x5x3 filter

At sliding position \( i,j \)

\[ \sigma(i,j) = b + \overline{W} \cdot \overline{x}_{ij} \]

with \( \overline{x}_{ij} = 5 \times 5 \) image patch in 3 colors
\( \rightarrow \) vector of dim 75, as filter coeffs in \( \overline{W} \)

Non-linear activation:

\( \sigma(i,j) = f(\sigma(i,j)) \)
\( f = \tanh, \text{ReLU}, \ldots \)


« Neural » view of convolution filters and layers

Each convolution FILTER is one set of neuron parameters
Each convolution LAYER is a set of ~imageSize neurons, but they all have same \textbf{SHARED} weights (perform SAME convolution)

\[
O = f\left(W_0 + \sum_{i=1}^{n} W_i e_i \right)
\]

\( W_0 = "\text{bias}" \)
\( f = \text{activation function} \)
### Convolutional layers

One “activation map” for each convolution filter

A convNet: succession of Convolution+activation Layers

NB: each convolution layer processes **FULL DEPTH** of previous activation map (3D convolution!)

### Pooling layers

Goal:
- aggregation over space
- noise reduction,
- small-translation invariance,
- small-scaling invariance
**Pooling layers algorithm details**

**Parameters:**
- pooling size (often 2x2)
- pooling stride (usually = pooling_size)
- Pooling operation: \textit{max}, average, Lp,...

\begin{center}
\begin{tabular}{cccc}
1 & 3 & 2 & 4 \\
1 & 3 & 2 & 4 \\
5 & 7 & 6 & 8 \\
5 & 7 & 6 & 8 \\
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{cc}
1 & 3 \\
5 & 7 \\
\end{tabular} \quad \begin{tabular}{cc}
3 & 4 \\
7 & 8 \\
\end{tabular}
\end{center}

Average pooling \quad \textit{Max pooling}

\textit{Example: 2x2 pooling, stride 2}

**Final classification layer:**
- often classical MLP

\textbf{AlexNet}

...
ConvNet typical architecture: cascade of modules

**Linear Module**
\[ \text{Out} = W \cdot \text{In} + B \]

**ReLU Module (Rectified Linear Unit)**
\[ \text{Out}_i = 0 \text{ if } \text{In}_i < 0 \]
\[ \text{Out}_i = \text{In}_i \text{ otherwise} \]

**Pooling Module**
\[ \text{Out} = \text{Cost Module: Squared Distance} \]
\[ C = \| \text{In}_1 - \text{In}_2 \|^2 \]

**Objective Function**
\[ L(\Theta) = \frac{1}{p} \sum_k C(X_k, Y_k, \Theta) \]

**Outline**

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation
ConvNet training

All successive layers of a convNet forms a Deep neural network (with weigh-sharing inside each conv. Layer, and specific pooling layers).

Training = optimizing values of weights & biases
Method used = gradient descent

⇒ Stochastic Gradient Descent (SGD),
   using back-propagation:
   - Input 1 (or a few) random training sample(s)
   - Propagate
   - Calculate error (loss)
   - Back-propagate through all layers from end to input, to compute gradient
   - Update convolution filter weights

Computing gradient through cascade of modules

- A practical Application of Chain Rule
- Backprop for the state gradients:
  - $dC/dXi-1 = dC/dXi \cdot dXi/dXi-1$
  - $dC/dXi-1 = dC/dXi \cdot dFi(Xi-1, Wi)/dXi-1$
- Backprop for the weight gradients:
  - $dC/dWi = dC/dXi \cdot dXi/dWi$
  - $dC/dWi = dC/dXi \cdot dFi(Xi-1, Wi)/dWi$
Recall of back-prop principle

Smart method for efficient computing of gradient (w.r.t. weights) of a Neural Network cost function, based on chain rule for derivation.

Cost function is \( Q(t) = \sum_m \text{loss}(Y_m, D_m) \), where \( m \) runs over training set examples

Usually, \( \text{loss}(Y_m, D_m) = ||Y_m - D_m||^2 \) [quadratic error]

Total gradient:
\[
W(t+1) = W(t) - \lambda(t) \text{grad}_W(Q(t)) + \mu(t)(W(t) - W(t-1))
\]

Stochastic gradient:
\[
W(t+1) = W(t) - \lambda(t) \text{grad}_W(Q_m(t)) + \mu(t)(W(t) - W(t-1))
\]

where \( Q_m = \text{loss}(Y_m, D_m) \), is error computed on only ONE example randomly drawn from training set at every iteration and \( \lambda(t) = \) learning rate (fixed, decreasing or adaptive), \( \mu(t) = \) momentum

Now, how to compute \( \frac{dQ_m}{dW_{ij}} \)?

Backprop through layers: chain rule derivative computation

\[
\begin{align*}
\delta_j &= (dE_m/d\sigma_j) = (dE_m/dy_j)(dy_j/d\sigma_j) \quad \text{with} \quad E_m = ||Y_m - D_m||^2 \\
\delta_j &= 2(y_j - D_j)f'(\sigma_j) \quad \text{if neuron } j \text{ is an output} \\
\delta_j &= \sum_k W_{jk} \delta_k f'(\sigma_j) = (\sum_k W_{jk} \delta_k) f'(\sigma_j) \quad \text{if neuron } j \text{ is “hidden”}
\end{align*}
\]

⇒ all the \( \delta_j \) can be computed successively from last layer to upstream layers by “error backpropagation” from output
Error surface for neural net are NOT CONVEX!

- Local minima dominate in low-Dim...
- ...but recent work has shown that *saddle points* dominate in high-Dim

- Furthermore, most local minima are close to the global minimum

---

Saddle points in training curves

- Oscillating between two behaviors:
  - Slowly approaching a saddle point
  - Escaping it
Some ConvNet training « tricks »

- **Importance of input normalization**
  (zero mean, unit variance)

- **Importance of weights initialization**
  random but SMALL and prop. to $1/\sqrt{\text{nbInputs}}$

- **Decreasing (or adaptive) learning rate**

- **Importance of training set size**
  ConvNets often have a LARGE number of free parameters
  ➔ train them with a sufficiently large training-set!

- **Avoid overfitting** by:
  - Use of L1 or L2 **regularization** (after some epochs)
  - Use « Dropout » **regularization** (esp. on large FC layers)

---

What is Overfitting?

![Graph](image)

Trying to fit too many free parameters with not enough information can lead to **overfitting**

How to detect overfitting for iterative training?

![Graph](image)

Better = **AVOID overfitting by REGULARIZATION**
Avoid overfitting using L1/L2 regularization

**Regularization** = penalizing too complex models
Often done by adding a special term to cost function

For neural network, the regularization term is just the L2- or L1- norm of the vector of all weights:

\[ K = \sum_m (\text{loss}(Y_m, D_m)) + \beta \sum_{ij} |W_{ij}|^p \quad \text{with } p=2 \text{ (L2) or } p=1 \text{ (L1)} \]

\[ \Rightarrow \text{name } \text{“Weight decay”} \]

---

**DropOut regularization for convNet training**

At each training stage, individual nodes can be temporarily "dropped out" of the net with probability p (usually ~0.5), or re-installed with last values of weights
Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation

Examples of very successful ConvNets

- **LeNet**: 1st successful applications of ConvNets, by Yann LeCun in 1990’s. Used to read zip codes, digits, etc.
- **AlexNet**: Beginning of ConvNet “buzz”: largely outperformed competitors in ImageNet_ILSVRC2012 challenge. Developed by Alex Krizhevsky et al., architecture similar to LeNet (but deeper+larger, and some chained ConvLayers before Pooling). 60 M parameters!
- **ZF Net**: ILSVRC 2013 winner. Developed by Zeiler&Fergus, by modif of AlexNet on some architecture hyperparameters.
- **GoogLeNet**: ILSVRC 2014 winner, developed by Google. Introduced an Inception Module, + AveragePooling instead of FullyConnected layer at output. Dramatic reduction of number of parameters (4M, compared to AlexNet with 60M).
- **VGGNet**: Runner-up in ILSVRC 2014. Very deep (16 CONV/FC layers) → 140M parameters !!
LeNet, for digits/letters recognition [LeCun et al., 1998]

**Input: 32x32 image**

- Conv filters were 5x5, applied at stride 1
- Subsampling (Pooling) layers were 2x2 applied at stride 2
- i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

AlexNet, for image categorisation [Krizhevsky et al. 2012]

**Input: 224x224x3 image**

- 60 million parameters !...
Input: 224x224x3 image

ZFnet [Zeiler & Fergu, 2013]

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

GoogleNet [Szegedy et al., 2014]

ILSVRC 2014 winner (6.7% top 5 error)
ResNet (Residual Net), by Microsoft [He et al., 2015]

- ILSVRC 2015 *large* winner in 5 main tracks (3.6% top 5 error)
- 152 layers!!!
- But novelty = "skip" connections

ResNet global architecture

- 2-3 weeks of training on 8 GPU machine !!
- However, at runtime faster than a VGGNet! (even though it has 8x more layers)
Summary of recent ConvNet history

But most important is the choice of ARCHITECTURAL STRUCTURE

Outline

• Introduction to Deep Learning
• Convolutional Neural Networks (CNN or ConvNets)
  – Intro + Short reminder on Neural Nets
  – Convolution layers & Pooling layers + global architecture
  – Training algorithm + Dropout Regularization
• Useful pre-trained convNets
• Coding frameworks
• Transfer Learning
• Object localization and Semantic segmentation
Good convNets are very big (millions of parameters!)

Training generally performed on BIG datasets

→ Training time more manageable using GPU acceleration for ultra-parallel processing

Programming environments for Deep-Learning

  C++ library, hooks from Python → notebooks
- Torch [http://torch.ch/](http://torch.ch/)
- TensorFlow [https://www.tensorflow.org](https://www.tensorflow.org)
- Theano [http://www.deeplearning.net/software/theano/](http://www.deeplearning.net/software/theano/)
- Lasagne [http://lasagne.readthedocs.io](http://lasagne.readthedocs.io)
  lightweight library to build+train neural nets in Theano
- KERAS [https://keras.io](https://keras.io)
  Python front-end APIs mapped either on Tensor-Flow or Theano back-end

All of them handle transparent use of GPU, and most of them are used in Python code/notebook
Example of convNet code in Keras

```python
model = Sequential()

# 1 set of (Convolution+Pooling) layers, with Dropout
model.add(Convolution2D(conv_depth_1, kernel_size, kernel_size,
border_mode='valid', input_shape=(depth, height, width)))
model.add(MaxPooling2D(pool_size=(pooling_size, pooling_size)))
model.add(Activation('relu'))
model.add(Dropout(drop_prob))

# Now flatten to 1D, and apply 1 Fully Connected layer
model.add(Flatten())
model.add(Dense(hidden_size1, init='lecun_uniform'))
model.add(Activation('sigmoid'))

# Finally add a Softmax output layer, with 1 neuron per class
model.add(Dense(num_classes, init='lecun_uniform'))
model.add(Activation('softmax'))

# Training "session"
sgd = SGD(lr=learning_rate, momentum=0.8) # Optimizer
model.compile(loss='categorical_crossentropy', optimizer=sgd)
model.fit(X_train, Y_train, batch_size=32, nb_epoch=2, verbose=1,
validation_split=valid_proportion)

# Evaluate the trained model on the test set
model.evaluate(X_test, Y_test, verbose=1)
```

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization and Semantic segmentation
Generality of learnt representation + Transfer Learning

By removing last layer(s) (those for classification) of a convNet trained on ImageNet, one obtains a transformation of any input image into a semi-abstract representation, which can be used for learning SOMETHING ELSE (« transfer learning »):

– either by just using learnt representation as features
– or by creating new convNet output and perform learning of new output layers + fine-tuning of re-used layers

Transfer Learning and fine-tuning

• Using a CNN pre-trained on a large dataset, possible to adapt it to another task, using only a SMALL training set!
Transfer Learning code example in Keras

```python
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras import backend as K

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)

# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(input=base_model.input, output=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

# train the model on the new data for a few epochs
model.fit_generator(...)
```

Transfer-Learning for 6-DOF Camera Relocalization

Outline

- Introduction to Deep Learning
- Convolutional Neural Networks (CNN or ConvNets)
  - Intro + Short reminder on Neural Nets
  - Convolution layers & Pooling layers + global architecture
  - Training algorithm + Dropout Regularization
- Useful pre-trained convNets
- Coding frameworks
- Transfer Learning
- Object localization, Semantic segmentation, etc...

Objects visual detection and categorization with DL

Region-based CNN (R-CNN), combines hierarchical features computation part of image classification CNN with a « Region Proposal Network » → objects’ bounding-boxes
ConvNets are currently state-of-the-art for visual objects detection and categorization

Semantic segmentation
Convolutional Encoder-Decoder

**SegNet**

ICnet


Many other competitors for deep-learning semantic segmentation

- U-Net (2015)
- RefineNet (2016)
- Adversarial Network (2016)
- ICnet (2017)

VERY HOT TOPIC !!!
Deep-Convolution for multivariate time-series

MC-DCNN model
(separate 1D temporal convolution of each time-serie)

Deep Gesture Recognition

Work in progress at center for Robotics of MINES ParisTech
(PhD thesis of Guillaume Devineau)

Potential applicability to other kinds of time-series!
Summary and perspectives on ConvNets & Deep-Learning

• Proven advantage of learning features empirically from data

• Large ConvNets require huge amounts of labelled examples data for training

• Current research/progresses = finding efficient global architecture of ConvNets

• Enormous potential of transfer learning on small datasets for restricted/specialized problems

• ConvNets for multivariate time-series?

• Next frontier: methods for combining UNsupervised deep-learning on unlabelled data with supervised training on smaller labelled dataset

Any QUESTIONS?