Deep neural architectures for structured output problems

Soufiane Belharbi
soufiane.belharbi@litislab.fr

Clément Chatelain
clement.chatelain@insa-rouen.fr

LITIS - INSA de Rouen

Joint work with: J.Lerouge, R.Herault, S.Adam, R.Modzelewski, F.Jardin, B.Labbe

May 20, 2015
Plan

1. Structured Output Problems
2. Input/Output Deep Architecture (IODA)
3. Application of IODA to medical image labeling
4. Application of IODA to Facial Landmark Detection
5. Conclusion
6. Future Work on IODA
Traditional Machine Learning Problems

\[ f : \mathcal{X} \rightarrow y \]
- Inputs \( \mathcal{X} \in \mathbb{R}^d \): any type of input
- Outputs \( y \in \mathbb{R} \) for the task: classification, regression, etc.

Machine Learning for Structured Output Problems

\[ f : \mathcal{X} \rightarrow \mathcal{Y} \]
- Inputs \( \mathcal{X} \in \mathbb{R}^d \): any type of input
- Outputs \( \mathcal{Y} \in \mathbb{R}^{d''}, d'' > 1 \) a structured object (dependencies)

See C. Lampert slides [3].
Traditional Machine Learning Problems

\[ f : \mathcal{X} \rightarrow y \]

- Inputs \( \mathcal{X} \in \mathbb{R}^d \): any type of input
- Outputs \( y \in \mathbb{R} \) for the task: classification, regression, ...

Machine Learning for Structured Output Problems

\[ f : \mathcal{X} \rightarrow \mathcal{Y} \]

- Inputs \( \mathcal{X} \in \mathbb{R}^d \): any type of input
- Outputs \( \mathcal{Y} \in \mathbb{R}^{d'}, d' > 1 \) a structured object (dependencies)

See C. Lampert slides [3].
Data = \textit{representation (values)} + \textit{structure (dependencies)}

Text: part-of-speech tagging, translation

\textit{speech} ⇋ \textit{text}

Protein folding

Image

Structured data
### Approaches that Deal with Structured Output Data

- **Kernel based methods:** Kernel Density Estimation (KDE)
- **Discriminative methods:** Structure output SVM
- **Graphical methods:** HMM, CRF, MRF, ...

### Drawbacks

- Perform one single data transformation
- Difficult to deal with *high dimensional* data

### Ideal approach

- Structured output problems
- High dimension data
- Multiple data transformation (complex mapping functions)

**Deep neural networks?**
## Approaches that Deal with Structured Output Data

- **Kernel based methods**: Kernel Density Estimation (KDE)
- **Discriminative methods**: Structure output SVM
- **Graphical methods**: HMM, CRF, MRF, ...

## Drawbacks

- Perform one single data transformation
- Difficult to deal with *high dimensional* data

## Ideal approach

- Structured output problems
- High dimension data
- Multiple data transformation (complex mapping functions)

**Deep neural networks?**
High dimension data OK
Multiple data transformation (complex mapping functions) OK
Structured output problems NO
Plan

1. Structured Output Problems
2. Input/Output Deep Architecture (IODA)
3. Application of IODA to medical image labeling
4. Application of IODA to Facial Landmark Detection
5. Conclusion
6. Future Work on IODA
IODA:

- Incorporate the output structure by learning
- Discover hidden dependencies in the outputs
Training IODA

Input layer

Hidden layer 1

\( X_1 \)

\( X_2 \)

\( X_3 \)

\( X_4 \)

\( X_5 \)

\( X_6 \)

\( X_7 \)
Training IODA

Input layer

$x_1$

$x_2$

$x_3$

$x_4$

$x_5$

$x_6$

$x_7$

Hidden layer 1
Training IODA

Input layer

Hidden layer 1

Hidden layer 2
Training IODA

\[ \mathcal{R}_{in}(x_i; \theta_{in}) = \hat{x}_i \]
Training IODA

Input layer

Hidden layer 1

Hidden layer 2

Output layer

\[ R_{in}(x_i; \theta_{in}) = \hat{x}_i \]
Training IODA

\[
\mathcal{R}_{in}(x_i; \theta_{in}) = \hat{x}_i
\]
Training IODA

\[ \mathcal{R}_{in}(x_i; \theta_{in}) = \hat{x}_i \]
Training IODA

\[ \mathcal{R}_{in}(x_i; \theta_{in}) = \hat{x}_i \]

\[ \mathcal{R}_{out}(y_i; \theta_{out}) = \hat{y}_i \]
Training IODA

\[ R_{\text{in}}(x_i; \theta_{\text{in}}) = \hat{x}_i \]

\[ R_{\text{out}}(y_i; \theta_{\text{out}}) = \hat{y}_i \]

\[ M(x_i; \theta, \theta_{\text{in}}, \theta_{\text{out}}) = \hat{y}_i \]
IODA framework: \( \min_{\theta} \mathcal{L}(\theta, \mathcal{D}(\mathbf{x}, \mathbf{y})) \)

\[
\mathcal{L}(\theta, \mathcal{D}(\mathbf{x}, \mathbf{y})) = \frac{1}{n} \sum_{i=1}^{n} \left[ \mathcal{C}(\mathcal{M}(\mathbf{x}_i; \theta_{\text{in}}, \theta_{\text{out}}), \mathbf{y}_i) \right. \\
\left. + \mathcal{\ell}_{\text{in}}(\mathcal{R}_{\text{in}}(\mathbf{x}_i; \theta_{\text{in}}), \mathbf{x}_i) \right] \\
+ \mathcal{\ell}_{\text{out}}(\mathcal{R}_{\text{out}}(\mathbf{y}_i; \theta_{\text{out}}), \mathbf{y}_i) \\
\]

\( \mathcal{C}(\cdot), \mathcal{\ell}_{\text{in}}(\cdot), \mathcal{\ell}_{\text{out}}(\cdot) \): defined costs.

\( \min_{\theta} \mathcal{L}(\theta, \mathcal{D}(\mathbf{x}, \mathbf{y})) \) is hard to solve \( \Rightarrow \) split \( \mathcal{L}(\theta, \mathcal{D}(\mathbf{x}, \mathbf{y})) \)
IODA framework: \( \min_{\theta} \mathcal{L}(\theta, \mathcal{D}(x, y)) \)

\[
\mathcal{L}(\theta, \mathcal{D}(x, y)) = \frac{1}{n} \sum_{i=1}^{n} \left[ \mathcal{C}(\mathcal{M}(x_i; \theta_{in}, \theta_{out}), y_i) \right. \\
\left. + \mathcal{\ell}_{in}(\mathcal{R}_{in}(x_i; \theta_{in}), x_i) \right\} \\
\left. + \mathcal{\ell}_{out}(\mathcal{R}_{out}(y_i; \theta_{out}), y_i) \right]
\]

\( \mathcal{C}(\cdot), \mathcal{\ell}_{in}(\cdot), \mathcal{\ell}_{out}(\cdot) \): defined costs.

\( \min_{\theta} \mathcal{L}(\theta, \mathcal{D}(x, y)) \) is hard to solve \( \Rightarrow \) split \( \mathcal{L}(\theta, \mathcal{D}(x, y)) \)
Relaxed-simplified version of IODA

1. **Unsupervised training:**
   
   → *Input* dependencies: \( \min_{\theta_{in}} \frac{1}{n} \sum_{i=1}^{n} \ell_{in}(R_{in}(x_i;\theta_{in}), x_i) \)
   
   → *output* dependencies: \( \min_{\theta_{out}} \frac{1}{n} \sum_{i=1}^{n} \ell_{out}(R_{out}(y_i;\theta_{out}), y_i) \)

2. **Standard supervised learning:**
   \[
   \min_{\theta,\theta_{in},\theta_{out}} \frac{1}{n} \sum_{i=1}^{n} C(M(x_i;\theta,\theta_{in},\theta_{out}), y_i) \]

Open source implementation

*Implemented using our library: Crino [1] [Python-Theano based].*
Relaxed-simplified version of IODA

1 Unsupervised training:
   → Input dependencies: \( \min_{\theta_{in}} \frac{1}{n} \sum_{i=1}^{n} \ell_{in}(R_{in}(x_i; \theta_{in}), x_i) \)
   → output dependencies: \( \min_{\theta_{out}} \frac{1}{n} \sum_{i=1}^{n} \ell_{out}(R_{out}(y_i; \theta_{out}), y_i) \)

2 Standard supervised learning:
   \( \min_{\theta, \theta_{in}, \theta_{out}} \frac{1}{n} \sum_{i=1}^{n} C(M(x_i; \theta, \theta_{in}, \theta_{out}), y_i) \)

Open source implementation

*Implemented using our library: Crino [1] [Python-Theano based].*
Plan

1. Structured Output Problems
2. Input/Output Deep Architecture (IODA)
3. Application of IODA to medical image labeling
4. Application of IODA to Facial Landmark Detection
5. Conclusion
6. Future Work on IODA
Image labeling problems

Definition

Assigning a label to each pixel of an image (AKA "semantic segmentation")

Various applications in:
- Document image analysis (text, image, tables, etc.)
- Computer vision (road safety, natural scene understanding)
- Medical imaging (organ, tumour segmentation)
Image labeling problems

Output dependencies

- Local dependencies (neighbouring labels are correlated)
- Structural dependencies (sky is generally above grass)

→ Image labeling can be considered as a structured output problems
Collaboration with the Henri Becquerel Center (Quantif team)

- Sarcopenia is a critical indication for lymphoma treatment
- Can be measured on scanner images by labeling skeletal muscle at L3 (third vertebra)
- 4 min/patient for a senior radiologist

Dataset

- 128 labeled L3 scanner images 512*512 pix
- Reference method from Chung (based on registration)
Input/Output Deep Architecture (IODA) for Image Labeling

IODA architecture for skeletal muscle segmentation
Application of IODA to medical image labeling

Implementation

Architecture (optimized on validation set)

A few figures:
- 428 M parameters (!!)
- Less than an hour for training (GPU, 4Go)
- 201.2 ms for decision
Qualitative results 1/2

(a) CT image  (b) Ground truth

(c) Chung  (d) IODA

Non-sarcopenic patient
Qualitative results 2/2

(a) CT image  (b) Ground truth

(c) Chung  (d) IODA

Sarcopenic patient
## Quantitative results

<table>
<thead>
<tr>
<th>Method</th>
<th>Diff. (%)</th>
<th>Jaccard (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chung (reference method)</td>
<td>-10.6</td>
<td>60.3</td>
</tr>
<tr>
<td>No pre-train DA</td>
<td>0.12</td>
<td>85.88</td>
</tr>
<tr>
<td>Input pre-train DA</td>
<td>0.15</td>
<td>85.91</td>
</tr>
<tr>
<td>Input/Output pre-train DA (IODA)</td>
<td>3.37</td>
<td><strong>88.47</strong></td>
</tr>
</tbody>
</table>
Feed the network with a blank image

Published in *pattern recognition* [4]
Plan

1. Structured Output Problems
2. Input/Output Deep Architecture (IODA)
3. Application of IODA to medical image labeling
4. Application of IODA to Facial Landmark Detection
5. Conclusion
6. Future Work on IODA
Facial landmarks:
set of facial key points with coordinates \((x,y)\)

Task ➞ predict the shape (set of points) given a facial image

⇒ Geometric dependencies ⇒ structured output problem
⇒ Apply IODA (regression task)
Facial landmarks:
set of facial key points with coordinates \((x,y)\)

Task: predict the shape (set of points) given a facial image

Geometric dependencies \(\Rightarrow\) structured output problem

Apply IODA (regression task)
**Facial landmarks:**
set of **facial key points** with coordinates \((x, y)\)

**Task**
→ predict the **shape** (**set of points**) given a facial image

⇒ **Geometric dependencies** ⇒ structured output problem
⇒ Apply IODA (regression task)
Datasets & Performance Measures

- **Datasets:** LFPW (~1000 samples), HELEN (~2300 samples)
- **Performance Measure:**
  - Normalized Root Mean Square Error (NRMSE)
  - Cumulative Distribution Function: $\text{CDF}_{\text{NRMSE}}$
  - Area Under the CDF Curve (AUC) **new**

Architecture (optimized on validation set)

$50 \times 50 = 2500 \quad 1024 \quad 512 \quad 64 \quad 68 \times 2 = 136$

⇒ Total training on GPU takes less than 30 mins.
Datasets & Performance Measures

- Datasets: LFPW(~1000 samples), HELEN(~2300 samples)
- Performance Measure:
  - Normalized Root Mean Square Error (NRMSE)
  - Cumulative Distribution Function: $\text{CDF}_{\text{NRMSE}}$
  - Area Under the CDF Curve (AUC) **new**

Architecture (optimized on validation set)

\[
\begin{align*}
\times &\quad \text{Total training on GPU takes less than 30mins.}
\end{align*}
\]
Datasets & Performance Measures

- Datasets: LFPW (~1000 samples), HELEN (~2300 samples)
- Performance Measure:
  - Normalized Root Mean Square Error (NRMSE)
  - Cumulative Distribution Function: $CDF_{NRMSE}$
  - Area Under the CDF Curve (AUC) **new**

Architecture (optimized on validation set)

$50 \times 50 = 2500 \quad 1024 \quad 512 \quad 64 \quad 68 \times 2 = 136$

$\Rightarrow$ Total training on GPU takes less than 30mins.
Application of IODA to Facial Landmark Detection

Visual results LFPW

No pre-train DA

Input pre-train DA

Input/Output pre-train DA (IODA)

Visual results LFPW
Application of IODA to Facial Landmark Detection

No pre-train DA

Input pre-train DA

Input/Output pre-train DA (IODA)

Visual results HELEN
<table>
<thead>
<tr>
<th></th>
<th>LFPW</th>
<th>HELEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AUC</td>
<td>CDF$_{0.1}$</td>
</tr>
<tr>
<td>Mean shape</td>
<td>66.15%</td>
<td>18.30%</td>
</tr>
<tr>
<td>No pre-train DA 0-0-0</td>
<td>77.60%</td>
<td>50.89%</td>
</tr>
<tr>
<td>Input pre-train DA 1-0-0</td>
<td>79.25%</td>
<td>62.94%</td>
</tr>
<tr>
<td></td>
<td>79.10%</td>
<td>58.48%</td>
</tr>
<tr>
<td></td>
<td>79.51%</td>
<td>65.62%</td>
</tr>
<tr>
<td>Input/Output pre-train DA 1-0-1</td>
<td>80.66%</td>
<td>68.30%</td>
</tr>
<tr>
<td>1-0-0</td>
<td>81.50%</td>
<td>72.32%</td>
</tr>
<tr>
<td>1-0-1</td>
<td>81.00%</td>
<td>71.42%</td>
</tr>
<tr>
<td>1-1-2</td>
<td>81.06%</td>
<td>70.98%</td>
</tr>
<tr>
<td>1-0-3</td>
<td>81.91%</td>
<td>74.55%</td>
</tr>
<tr>
<td>2-0-1</td>
<td>81.32%</td>
<td>72.76%</td>
</tr>
<tr>
<td>2-1-1</td>
<td>81.47%</td>
<td>70.08%</td>
</tr>
<tr>
<td>2-0-2</td>
<td>81.35%</td>
<td>71.87%</td>
</tr>
<tr>
<td>3-0-1</td>
<td>81.62%</td>
<td>72.76%</td>
</tr>
</tbody>
</table>

Performance of mean shape, NDA, IDA and IODA on LFPW and HELEN.
Feed a blank image to a trained network ⇒ what is the output?

The outputs on LFPW

Paper submitted to ECML 2015 (arXiv [2]).
Plan

1. Structured Output Problems
2. Input/Output Deep Architecture (IODA)
3. Application of IODA to medical image labeling
4. Application of IODA to Facial Landmark Detection
5. Conclusion
6. Future Work on IODA
- Fully **neural** based approach
- Able to learn the **output dependencies** in **high dimension**
- Efficient on two real world problems
Plan

1. Structured Output Problems
2. Input/Output Deep Architecture (IODA)
3. Application of IODA to medical image labeling
4. Application of IODA to Facial Landmark Detection
5. Conclusion
6. Future Work on IODA
**Embedded Pre-training** (draft on arXiv):

\[
\mathcal{L}(\theta, \mathcal{D}(x, y)) = \frac{1}{n} \sum_{i=1}^{n} \left[ \lambda_C \mathcal{C}(\mathcal{M}(x_i; \theta, \theta_{in}, \theta_{out}), y_i) \\
+ \lambda_{in} \ell_{in}(\mathcal{R}_{in}(x_i; \theta_{in}), x_i) \\
+ \lambda_{out} \ell_{out}(\mathcal{R}_{out}(y_i; \theta_{out}), y_i) \right]
\]
Use of unlabeled data:

\[
\mathcal{L}(\theta, \mathcal{D}(x, y)) = \frac{1}{n} \sum_{i=1}^{n} \lambda_c \mathcal{C}(\mathcal{M}(x_i; \theta, \theta_{in}, \theta_{out}), y_i) + \frac{1}{n + n_{in}} \sum_{i=1}^{n + n_{in}} \lambda_{in} \mathcal{L}_{in}(\mathcal{R}_{in}(x_i; \theta_{in}), x_i) + \frac{1}{n + n_{out}} \sum_{i=1}^{n + n_{out}} \lambda_{out} \mathcal{L}_{out}(\mathcal{R}_{out}(y_i; \theta_{out}), y_i)
\]

\(n_{in}, n_{out}\) potentially huge unlabeled input, output data.
3 Convolutional IODA:
Convolutional layers are efficient in feature extraction

⇒ Use convolutional layers instead of auto-encoders in the input-layers

ECML, 2015.


Ioda: An input output deep architecture for image labeling. 
Thank you for your attention.
<table>
<thead>
<tr>
<th>Sets</th>
<th>Train samples</th>
<th>Test samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFPW</td>
<td>811</td>
<td>224</td>
</tr>
<tr>
<td>HELEN</td>
<td>2000</td>
<td>330</td>
</tr>
</tbody>
</table>

Number of samples in datasets.
Normalized Root Mean Square Error (NRMSE)

\[
NRMSE(s_p, s_g) = \frac{1}{n \times D} \sum_{i=1}^{n} \| s_{pi} - s_{gi} \|_2,
\]

\( s_p, s_g \) predicted, ground truth shape. \( D \) inter-ocular distance of \( s_g \)

Cumulative Distribution Function: \( CDF_{NRMSE} \)

\[
CDF_x = \frac{\text{CARD}(NRMSE \leq x)}{N}
\]

\( \text{CARD}(.) \) cardinal of a set. \( N \) number of images.

e.g. \( CDF_{0.1} = 0.4 \) means that 40% of images have an NRMSE error less or equal than 0.1

Area Under the CDF Curve (AUC) "new": more numerical precision

- Plot a \( CDF_{NRMSE} \) curve by varying NRMSE in \([0, 0.5]\).
- Calculate the area under this curve.
Normalized Root Mean Square Error (NRMSE)

\[
NRMSE(s_p, s_g) = \frac{1}{n \times D} \sum_{i=1}^{n} \| s_{pi} - s_{gi} \|_2,
\]

\(s_p, s_g\) predicted, ground truth shape. \(D\) inter-ocular distance of \(s_g\)

Cumulative Distribution Function: \(CDF_{NRMSE}\)

\[
CDF_x = \frac{\text{CARD}(NRMSE \leq x)}{N}
\]

\(\text{CARD}(.)\) cardinal of a set. \(N\) number of images.

e.g. \(CDF_{0.1} = 0.4\) means that 40% of images have an NRMSE error less or equal than 0.1

Area Under the CDF Curve (AUC) **new**: more numerical precision

- Plot a \(CDF_{NRMSE}\) curve by varying NRMSE in \([0, 0.5]\).
- Calculate the area under this curve.
Normalized Root Mean Square Error (NRMSE)

\[
NRMSE(s_p, s_g) = \frac{1}{n \times D} \sum_{i=1}^{n} ||s_{pi} - s_{gi}||_2,
\]

\(s_p, s_g\) predicted, ground truth shape. \(D\) inter-ocular distance of \(s_g\)

Cumulative Distribution Function: \(CDF_{NRMSE}\)

\[
CDF_x = \frac{CARD(NRMSE \leq x)}{N}
\]

\(CARD(.)\) cardinal of a set. \(N\) number of images.

e.g. \(CDF_{0.1} = 0.4\) means that 40% of images have an NRMSE error less or equal than 0.1

Area Under the CDF Curve (AUC) **new**: more numerical precision

- Plot a \(CDF_{NRMSE}\) curve by varying NRMSE in [0, 0.5].
- Calculate the area under this curve.
Input layer pre-training using auto-encoders (1)

\[ \hat{x} \leftarrow d \]

\[ d = h(V \times e + c) \quad \text{decoder} \]

\[ e = g(U \times x + a) \quad \text{coder} \]

\[ U = V^T: \text{a tied weight auto-encoder} \]

\[ \hat{x} = \mathcal{R}_{\text{in}}(x) \]
Input layer pre-training using auto-encoders (1)

\[
\hat{x} \leftarrow d \quad x
\]

\[
d = h(V \times e + c) \quad \text{decoder}
\]

\[
e = g(U \times x + a) \quad \text{coder}
\]

\[
U = V^T: \text{a tied weight auto-encoder}
\]

\[
\hat{x} = \mathcal{R}_{in}(x)
\]
Output layer pre-training using auto-encoders (2)

\[
\hat{y} \leftarrow d
\]

\[
y = g(U \times y + a)^{coder}
\]

\[
d = h(V \times e + c)^{decoder}
\]

\[
e = g(U \times y + a)^{coder}
\]

\[
U = V^T: a \text{ tied weight auto-encoder}
\]

\[
\hat{y} = R_{out}(y)
\]
Output layer pre-training using auto-encoders (2)

\[ \hat{y} \leftarrow d \]

\[ d = h(V \times e + c) \] \_\text{decoder} \]

\[ e = g(U \times y + a) \] \_\text{coder} \]

\[ U = V^T : \text{a tied weight auto-encoder} \]

\[ \hat{y} = R_{out}(y) \]