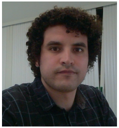
Deep neural networks and structured output problems

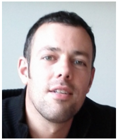
presentation of my current PhD work

ISP seminar. UCL, Louvain-la-Neuve 2016



Soufiane Belharbi

Romain Hérault



Clément Chatelain

Sébastien Adam

soufiane.belharbi@insa-rouen.fr

LITIS lab., Apprentissage team - INSA de Rouen, France

Dec.12^{t}h.2016

My PhD work

1. S. Belharbi, R.Hérault, C. Chatelain, S. Adam, ***Deep multi-task learning with evolving weights***, in conference: European Symposium on Artificial Neural Networks (ESANN), 2016
2. S. Belharbi, C. Chatelain, R.Hérault, S. Adam, ***A regularization scheme for structured output problems: an application to facial landmark detection***. 2016. submitted to Pattern Recognition journal (PR). ArXiv: arxiv.org/abs/1504.07550
3. S. Belharbi, R.Hérault, C. Chatelain, R. Modzelewski, S. Adam, M. Chastan, S. Thureau, ***Spotting L3 slice in CT scans using deep convolutional network and transfer learning***. To be submitted to Medical Image Analysis journal (MIA). 2016.

Quick-informal introduction to Machine Learning

What is Machine Learning (ML)?

ML is programming computers (algorithms) to optimize a performance criterion using **example data or past experience**.

Learning a task

Learn general models from data to perform a specific task f.

$$f_{\mathbf{w}} : \mathbf{x} \longrightarrow \mathbf{y}$$

$\mathbf{x}$: input
$\mathbf{y}$: output (target, label)
$\mathbf{w}$: parameters of f
$f(\mathbf{x}; \mathbf{w}) = \mathbf{y}$

From training to predicting the future: **Learn to predict**

1. Train the model using data examples $(\mathbf{x}, \mathbf{y})$
2. Predict the $\mathbf{y}_{new}$ for the new coming $\mathbf{x}_{new}$

Quick-informal introduction to Machine Learning

What is Machine Learning (ML)?

ML is programming computers (algorithms) to optimize a performance criterion using **example data or past experience.**

Learning a task

Learn general models from data to perform a specific task f.

$$f_{\mathbf{w}} : \mathbf{x} \longrightarrow \mathbf{y}$$

$\mathbf{x}$: input
$\mathbf{y}$: output (target, label)
$\mathbf{w}$: parameters of f
$f(\mathbf{x}; \mathbf{w}) = \mathbf{y}$

From training to predicting the future: **Learn to predict**

1. Train the model using data examples $(\mathbf{x}, \mathbf{y})$
2. Predict the $\mathbf{y}_{new}$ for the new coming $\mathbf{x}_{new}$

Quick-informal introduction to Machine Learning

What is Machine Learning (ML)?

ML is programming computers (algorithms) to optimize a performance criterion using **example data or past experience.**

Learning a task

Learn general models from data to perform a specific task f.

$$f_{\mathbf{w}} : \mathbf{x} \longrightarrow \mathbf{y}$$

$\mathbf{x}$: input
$\mathbf{y}$: output (target, label)
$\mathbf{w}$: parameters of f
$f(\mathbf{x}; \mathbf{w}) = \mathbf{y}$

From training to predicting the future: **Learn to predict**

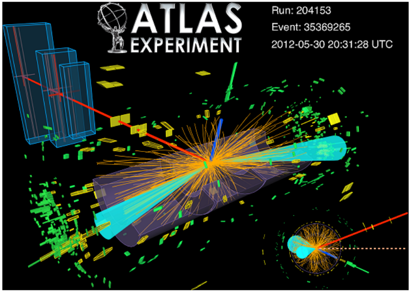
1. Train the model using data examples $(\mathbf{x}, \mathbf{y})$
2. Predict the $\mathbf{y}_{new}$ for the new coming $\mathbf{x}_{new}$

Machine Learning applications

- Face detection/recognition
- Image classification
- Handwriting recognition(postal address recognition, signature verification, writer verification, historical document analysis (DocExplore `http://www.docexplore.eu`))
- Speech recognition, Voice synthesizing
- Natural language processing (sentiment/intent analysis, statistical machine translation, Question answering (Watson), Text understanding/summarizing, text generation)
- Anti-virus, anti-spam
- Weather forecast
- Fraud detection at banks
- Mail targeting/advertising
- Pricing insurance premiums
- Predicting house prices in real estate companies
- Win-tasting ratings
- Self-driving cars, Autonomous robots
- Factory Maintenance diagnostics
- Developing pharmaceutical drugs (combinatorial chemistry)
- Predicting tastes in music (Pandora)
- Predicting tastes in movies/shows (Netflix)
- Search engines (Google)
- Predicting interests (Facebook)
- Web exploring (sites like this one)
- Biometrics (finger prints, iris)
- Medical analysis (image segmentation, disease detection from symptoms)
- Advertisements/Recommendations engines, predicting other books/products you may like (Amazon)
- Computational neuroscience, bioinformatics/computational biology, genetics
- Content (image, video, text) categorization
- Suspicious activity detection
- Frequent pattern mining (super-market)
- Satellite/astronomical image analysis

ML in physics

Event detection at CERN (The European Organization for Nuclear Research)

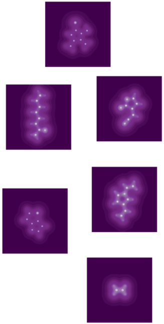


⇒ Use ML models to determine the probability of the event being of interest.

⇒ **Higgs Boson Machine Learning Challenge** (`https://www.kaggle.com/c/higgs-boson`)

ML in quantum chemistry

Computing the electronic density of a molecule
$\Rightarrow$ Instead of using physics laws, use ML (**FAST**).



See Stéphane Mallat et al. work: `https://matthewhirn.files.wordpress.com/2016/01/hirn_pasc15.pdf`

How to estimate $f_{\mathbf{w}}$?

Models

- Parametric ($\mathbf{w}$) vs. non-parametric
- Estimate $f_{\mathbf{w}}$ = train the model using data
- Training: supervised (use $(\mathbf{x}, \mathbf{y})$) vs. unsupervised (use only $\mathbf{x}$)
- Training = optimizing an objective cost

Different models to learn $f_{\mathbf{w}}$

- Kernel models (support vector machine (SVM))
- Decision tree
- Random forest
- Linear regression
- K-nearest neighbor
- Graphical models
  - Bayesian networks
  - Hidden Markov Models (HMM)
  - Conditional Random Fields (CRF)
- Neural networks (Deep learning): DNN, CNN, RBM, DBN, RNN.

How to estimate $f_{\mathbf{w}}$?

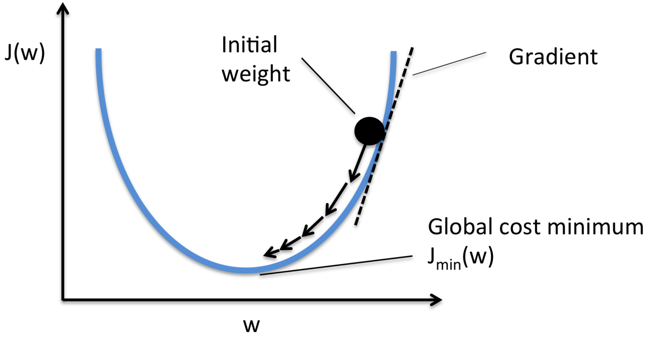
Models

- Parametric ($\mathbf{w}$) vs. non-parametric
- Estimate $f_{\mathbf{w}}$ = train the model using data
- Training: supervised (use $(\mathbf{x}, \mathbf{y})$) vs. unsupervised (use only $\mathbf{x}$)
- Training = optimizing an objective cost

Different models to learn $f_{\mathbf{w}}$

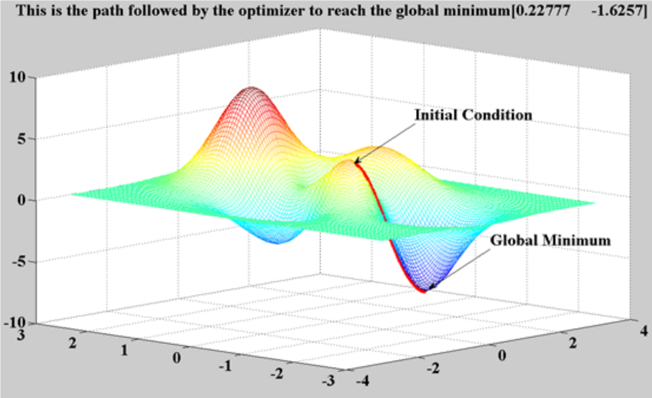
- Kernel models (support vector machine (SVM))
- Decision tree
- Random forest
- Linear regression
- K-nearest neighbor
- Graphical models
  - Bayesian networks
  - Hidden Markov Models (HMM)
  - Conditional Random Fields (CRF)
- Neural networks (Deep learning): DNN, CNN, RBM, DBN, RNN.

Optimization using Stochastic Gradient Descent (SGD)



$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \frac{\partial \mathcal{J}(\mathcal{D};\mathbf{w})}{\partial \mathbf{w}}$. $\mathcal{D}$ is a set of data.

Optimization using Stochastic Gradient Descent (SGD)



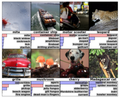
$$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \frac{\partial \mathcal{J}(\mathcal{D};\mathbf{w})}{\partial \mathbf{w}}.$$

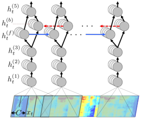
My PhD work

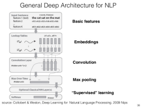
1. S. Belharbi, R.Hérault, C. Chatelain, S. Adam, ***Deep multi-task learning with evolving weights***, in conference: European Symposium on Artificial Neural Networks (ESANN), 2016
2. S. Belharbi, C. Chatelain, R.Hérault, S. Adam, ***A regularization scheme for structured output problems: an application to facial landmark detection***. 2016. submitted to Pattern Recognition journal (RP). ArXiv: arxiv.org/abs/1504.07550
3. S. Belharbi, R.Hérault, C. Chatelain, R. Modzelewski, S. Adam, M. Chastan, S. Thureau, ***Spotting L3 slice in CT scans using deep convolutional network and transfer learning***. To be submitted to Medical Analysis journal (MIA). 2016.

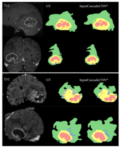
Deep learning Today

Deep learning state of the art









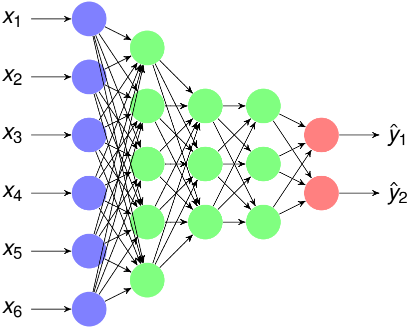
What is new today?

- Large data
- Calculation power (GPUS, clouds)

$\Rightarrow$ optimization

- Dropout
- Momentum, AdaDelta, AdaGrad, RMSProp, Adam, Adamax
- Maxout, Local response normalization, local contrast normalization, batch normalization
- RELU
- CNN, RBM, RNN

Deep neural networks (DNN)



- Feed-forward neural network
- Back-propagation error
- Training **deep** neural networks is **difficult**

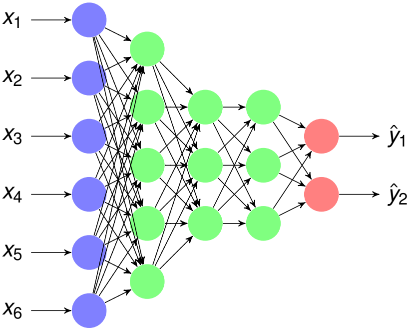
  ⇒ Vanishing gradient

  ⇒ Pre-training technique [Y.Bengio et al. 06, G.E.Hinton et al. 06]

  ⇒ More parameters ⇒ Need more data

  ⇒ Use unlabeled data

Deep neural networks (DNN)



- Feed-forward neural network
- Back-propagation error
- Training **deep** neural networks is **difficult**

  ⇒ Vanishing gradient

  ⇒ Pre-training technique [Y.Bengio et al. 06, G.E.Hinton et al. 06]

  ⇒ More parameters ⇒ Need more data

  ⇒ Use unlabeled data

Semi-supervised learning

General case:

$$\mathit{Data} = \{ \underbrace{\mathit{labeled\ data}\ (\mathbf{x}, \mathbf{y})}_{\text{expensive (money, time), few}}, \underbrace{\mathit{unlabeled\ data}\ (\mathbf{x}, --)}_{\text{cheap, abundant}} \}$$

E.g:

- Collect images from the internet
- Medical images

$\Rightarrow$ semi-supervised learning:

Exploit unlabeled data to improve the **generalization**

Semi-supervised learning

General case:

$$Data = \{ \underbrace{labeled\ data\ (\mathbf{x}, \mathbf{y})}_{\text{expensive (money, time), few}}, \underbrace{unlabeled\ data\ (\mathbf{x}, --)}_{\text{cheap, abundant}} \}$$

E.g:

- Collect images from the internet
- Medical images

$\Rightarrow$ semi-supervised learning:

Exploit unlabeled data to improve the **generalization**

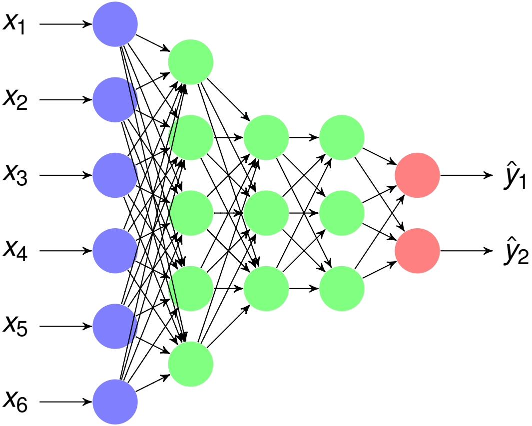
Pre-training and semi-supervised learning

The pre-training technique can exploit the unlabeled data

A **sequential** transfer learning performed in 2 steps:

1. **Unsupervised task** ($\mathbf{x}$ labeled and unlabeled data)
2. **Supervised task** ($(\mathbf{x}, \mathbf{y})$ labeled data)

Layer-wise pre-training: auto-encoders

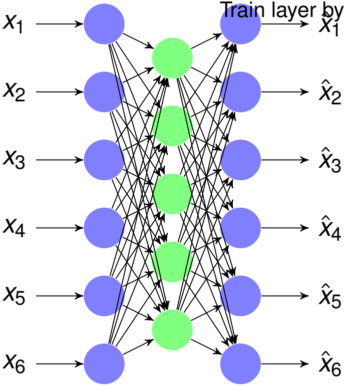


A DNN to train

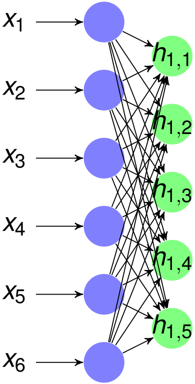
Layer-wise pre-training: auto-encoders

1) Step 1: **Unsupervised layer-wise pre-training**

Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



Layer-wise pre-training: auto-encoders



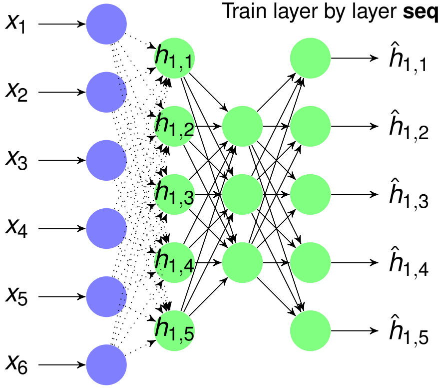
1) Step 1: **Unsupervised layer-wise pre-training**

Train layer by layer **sequentially** using **only x** (labeled or unlabeled)

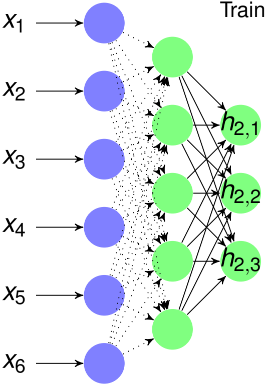
Layer-wise pre-training: auto-encoders

1) Step 1: **Unsupervised layer-wise pre-training**

Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



Layer-wise pre-training: auto-encoders



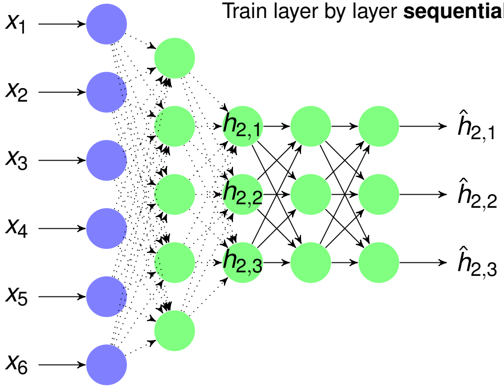
1) Step 1: **Unsupervised layer-wise pre-training**

Train layer by layer **sequentially** using **only x** (labeled or unlabeled)

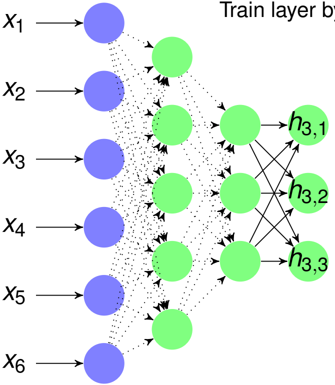
Layer-wise pre-training: auto-encoders

1) Step 1: **Unsupervised layer-wise pre-training**

Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



Layer-wise pre-training: auto-encoders



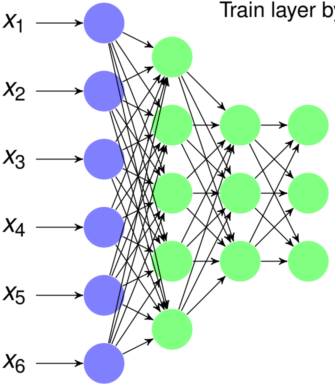
1) Step 1: **Unsupervised layer-wise pre-training**

Train layer by layer **sequentially** using **only x** (labeled or unlabeled)

Layer-wise pre-training: auto-encoders

1) Step 1: **Unsupervised layer-wise pre-training**

Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



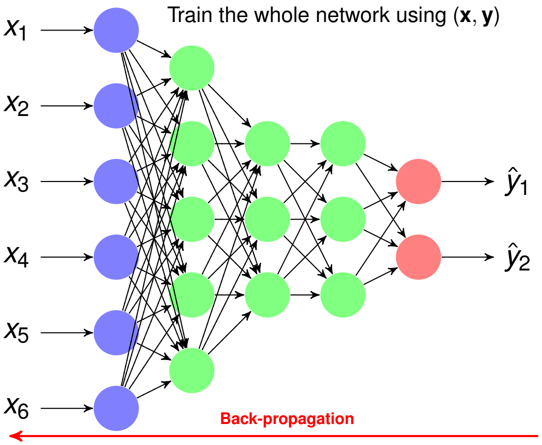
At each layer:

⇒ What hyper-parameters to use? When to stop training?

⇒ How to make sure that the pre-training improves the supervised task?

Layer-wise pre-training: auto-encoders

2) Step 2: **Supervised training**



Pre-training technique: Pros and cons

Pros

- Improve generalization
- Can exploit unlabeled data
- Provide better initialization than random
- Train deep networks

  $\Rightarrow$ Circumvent the vanishing gradient problem

Cons

- Add more hyper-parameters
- No good stopping criterion during pre-training phase

Good criterion for the unsupervised task

But

May not be good for the supervised task

Pre-training technique: Pros and cons

Pros

- Improve generalization
- Can exploit unlabeled data
- Provide better initialization than random
- Train deep networks
  $\Rightarrow$ Circumvent the vanishing gradient problem

Cons

- Add more hyper-parameters
- No good stopping criterion during pre-training phase

Good criterion for the unsupervised task
But
May not be good for the supervised task

Proposed solution

Why is it difficult in practice?

$\Rightarrow$ **Sequential** transfer learning

Possible solution:

$\Rightarrow$ **Parallel** transfer learning

Why in parallel?

- Interaction between tasks
- Reduce the number of hyper-parameters to tune
- Provide **one stopping criterion**

Proposed solution

Why is it difficult in practice?

$\Rightarrow$ **Sequential** transfer learning

Possible solution:

$\Rightarrow$ **Parallel** transfer learning

Why in parallel?

- Interaction between tasks
- Reduce the number of hyper-parameters to tune
- Provide **one stopping criterion**

Proposed solution

Why is it difficult in practice?

$\Rightarrow$ **Sequential** transfer learning

Possible solution:

$\Rightarrow$ **Parallel** transfer learning

Why in parallel?

- Interaction between tasks
- Reduce the number of hyper-parameters to tune
- Provide **one stopping criterion**

Parallel transfer learning: Tasks combination

$$\text{Train cost} = \textbf{supervised task} + \underbrace{\textbf{unsupervised task}}_{\text{reconstruction}}$$

l labeled samples, u unlabeled samples, $\mathbf{w}_{sh}$: shared parameters.

Reconstruction (auto-encoder) task:

$$\mathcal{J}_r(\mathcal{D}; \mathbf{w}' = \{\mathbf{w}_{sh}, \mathbf{w}_r\}) = \sum_{i=1}^{l+u} \mathcal{C}_r(\mathcal{R}(\mathbf{x}_i; \mathbf{w}'), \mathbf{x}_i) \ .$$

Supervised task:

$$\mathcal{J}_s(\mathcal{D}; \mathbf{w} = \{\mathbf{w}_{sh}, \mathbf{w}_s\}) = \sum_{i=1}^{l} \mathcal{C}_s(\mathcal{M}(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i) \ .$$

Weighted tasks combination

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

$\lambda_s, \ \lambda_r \in [0, 1]$: importance weight, $\lambda_s + \lambda_r = 1$.

Parallel transfer learning: Tasks combination

$$\text{Train cost} = \textbf{supervised task} + \underbrace{\textbf{unsupervised task}}_{\text{reconstruction}}$$

l labeled samples, u unlabeled samples, $\mathbf{w}_{sh}$: shared parameters.

Reconstruction (auto-encoder) task:

$$\mathcal{J}_r(\mathcal{D}; \mathbf{w}' = \{\mathbf{w}_{sh}, \mathbf{w}_r\}) = \sum_{i=1}^{l+u} \mathcal{C}_r(\mathcal{R}(\mathbf{x}_i; \mathbf{w}'), \mathbf{x}_i) \ .$$

Supervised task:

$$\mathcal{J}_s(\mathcal{D}; \mathbf{w} = \{\mathbf{w}_{sh}, \mathbf{w}_s\}) = \sum_{i=1}^{l} \mathcal{C}_s(\mathcal{M}(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i) \ .$$

Weighted tasks combination

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

$\lambda_s, \ \lambda_r \in [0, 1]$: importance weight, $\lambda_s + \lambda_r = 1$.

Parallel transfer learning: Tasks combination

$$\text{Train cost} = \textbf{supervised task} + \underbrace{\textbf{unsupervised task}}_{\text{reconstruction}}$$

l labeled samples, u unlabeled samples, $\mathbf{w}_{sh}$: shared parameters.

Reconstruction (auto-encoder) task:

$$\mathcal{J}_r(\mathcal{D}; \mathbf{w}' = \{\mathbf{w}_{sh}, \mathbf{w}_r\}) = \sum_{i=1}^{l+u} \mathcal{C}_r(\mathcal{R}(\mathbf{x}_i; \mathbf{w}'), \mathbf{x}_i) \ .$$

Supervised task:

$$\mathcal{J}_s(\mathcal{D}; \mathbf{w} = \{\mathbf{w}_{sh}, \mathbf{w}_s\}) = \sum_{i=1}^{l} \mathcal{C}_s(\mathcal{M}(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i) \ .$$

Weighted tasks combination

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

$\lambda_s, \ \lambda_r \in [0, 1]$: importance weight, $\lambda_s + \lambda_r = 1$.

Parallel transfer learning: Tasks combination

$$\text{Train cost} = \textbf{supervised task} + \underbrace{\textbf{unsupervised task}}_{\text{reconstruction}}$$

l labeled samples, u unlabeled samples, $\mathbf{w}_{sh}$: shared parameters.

Reconstruction (auto-encoder) task:

$$\mathcal{J}_r(\mathcal{D}; \mathbf{w}' = \{\mathbf{w}_{sh}, \mathbf{w}_r\}) = \sum_{i=1}^{l+u} \mathcal{C}_r(\mathcal{R}(\mathbf{x}_i; \mathbf{w}'), \mathbf{x}_i) \ .$$

Supervised task:

$$\mathcal{J}_s(\mathcal{D}; \mathbf{w} = \{\mathbf{w}_{sh}, \mathbf{w}_s\}) = \sum_{i=1}^{l} \mathcal{C}_s(\mathcal{M}(\mathbf{x}_i; \mathbf{w}), \mathbf{y}_i) \ .$$

Weighted tasks combination

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

$\lambda_s,\ \lambda_r \in [0, 1]$: importance weight, $\lambda_s + \lambda_r = 1$.

Tasks combination with evolving weights

Weighted tasks combination:

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

$\lambda_s, \ \lambda_r \in [0, 1]$: importance weight, $\lambda_s + \lambda_r = 1$.

Problem

How to **fix** λ_s, λ_r?

Intuition

At the end of the training, only $\mathcal{J}_s$ should matters

Tasks combination with evolving weights (our contribution)

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s(t) \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r(t) \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

t: learning epochs, $\lambda_s(t), \ \lambda_r(t) \in [0, 1]$: importance weight, $\lambda_s(t) + \lambda_r(t) = 1$.

Tasks combination with evolving weights

Weighted tasks combination:

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

$\lambda_s, \ \lambda_r \in [0, 1]$: importance weight, $\lambda_s + \lambda_r = 1$.

Problem

How to **fix** λ_s, λ_r?

Intuition

At the end of the training, only $\mathcal{J}_s$ should matters

Tasks combination with evolving weights (our contribution)

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s(t) \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r(t) \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

t: learning epochs, $\lambda_s(t), \ \lambda_r(t) \in [0, 1]$: importance weight, $\lambda_s(t) + \lambda_r(t) = 1$.

Tasks combination with evolving weights

Weighted tasks combination:

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

$\lambda_s, \ \lambda_r \in [0, 1]$: importance weight, $\lambda_s + \lambda_r = 1$.

Problem

How to **fix** λ_s, λ_r?

Intuition

At the end of the training, only $\mathcal{J}_s$ should matters

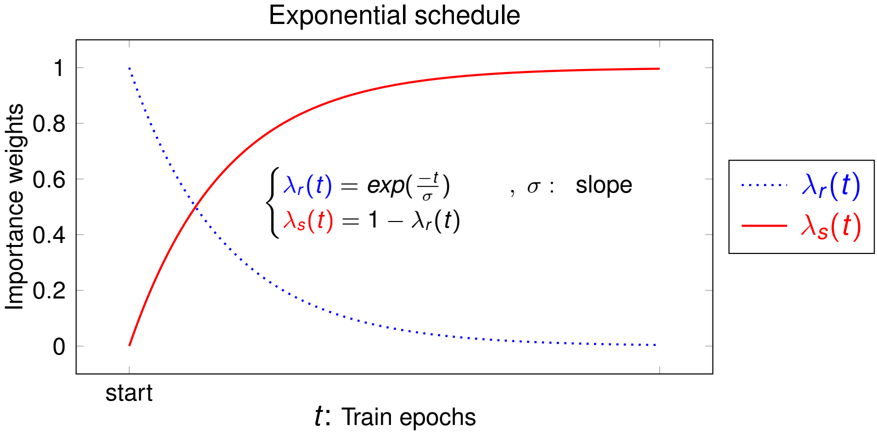
Tasks combination with evolving weights (our contribution)

$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s, \mathbf{w}_r\}) = \lambda_s(t) \cdot \mathcal{J}_s(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_s\}) + \lambda_r(t) \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{sh}, \mathbf{w}_r\}) \ .$$

t: learning epochs, $\lambda_s(t), \ \lambda_r(t) \in [0, 1]$: importance weight, $\lambda_s(t) + \lambda_r(t) = 1$.

Tasks combination with evolving weights

$$\mathcal{J}(\mathcal{D};\{\mathbf{w}_{sh},\mathbf{w}_s,\mathbf{w}_r\}) = \lambda_s(t)\cdot\mathcal{J}_s(\mathcal{D};\{\mathbf{w}_{sh},\mathbf{w}_s\})+\lambda_r(t)\cdot\mathcal{J}_r(\mathcal{D};\{\mathbf{w}_{sh},\mathbf{w}_r\})$$



Tasks combination with evolving weights: Optimization

Tasks combination with evolving weights (our contribution)

$$\mathcal{J}(\mathcal{D};\{\mathbf{w}_{sh},\mathbf{w}_s,\mathbf{w}_r\}) = \lambda_s(t)\cdot\mathcal{J}_s(\mathcal{D};\{\mathbf{w}_{sh},\mathbf{w}_s\}) + \lambda_r(t)\cdot\mathcal{J}_r(\mathcal{D};\{\mathbf{w}_{sh},\mathbf{w}_r\})\ .$$

t: learning epochs, $\lambda_s(t),\ \lambda_r(t) \in [0,1]$: importance weight, $\lambda_s(t)+\lambda_r(t)=1$.

Algorithm 1 Training our model for one epoch

1: $\mathcal{D}$ is the *shuffled* training set. B a mini-batch.
2: **for** B in $\mathcal{D}$ **do**
3: Make a gradient step toward $\mathcal{J}_r$ using B (update $\mathbf{w}'$)
4: $B_s \Leftarrow$ labeled examples of B,
5: Make a gradient step toward $\mathcal{J}_s$ using B_s (update $\mathbf{w}$)
6: **end for**

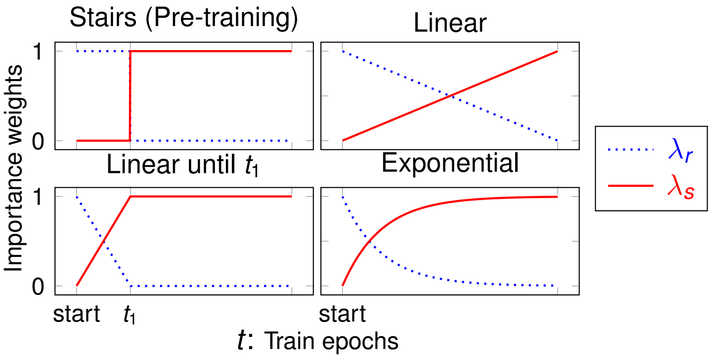
[R.Caruana 97, J.Weston 08, R.Collobert 08, Z.Zhang 15]

Experimental protocol

Objective: Compare Training DNN using different approaches:

- No pre-training (base-line)
- With pre-training (Stairs schedule)
- Parallel transfer learning (proposed approach)

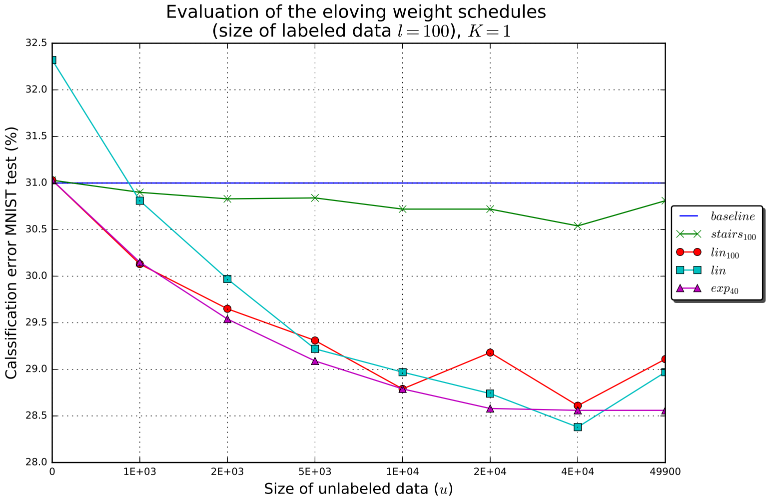
Studied evolving weights schedules:



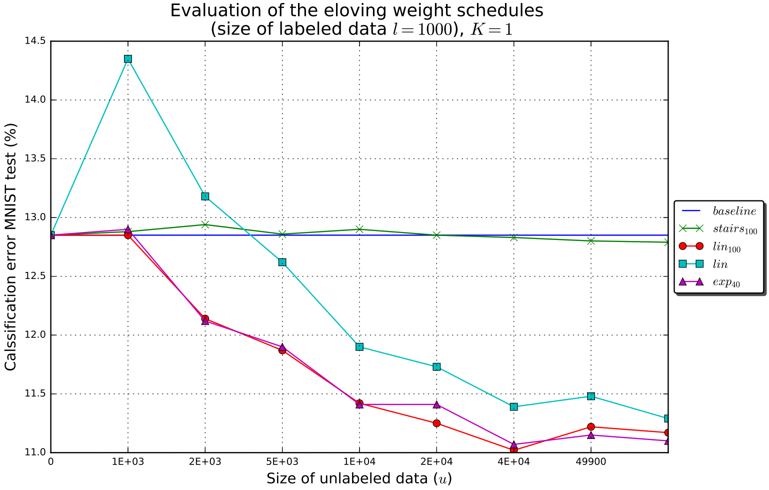
Experimental protocol

- **Task**: Classification (MNIST)
- **Number of hidden layers** K: 1, 2, 3, 4.
- **Optimization**:
  - **Epochs**: 5000
  - **Batch size**: 600
  - **Options**: **No** regularization, **No** adaptive learning rate
- **Hyper-parameters of the evolving schedules**:
  - t_1: 100
  - σ: 40

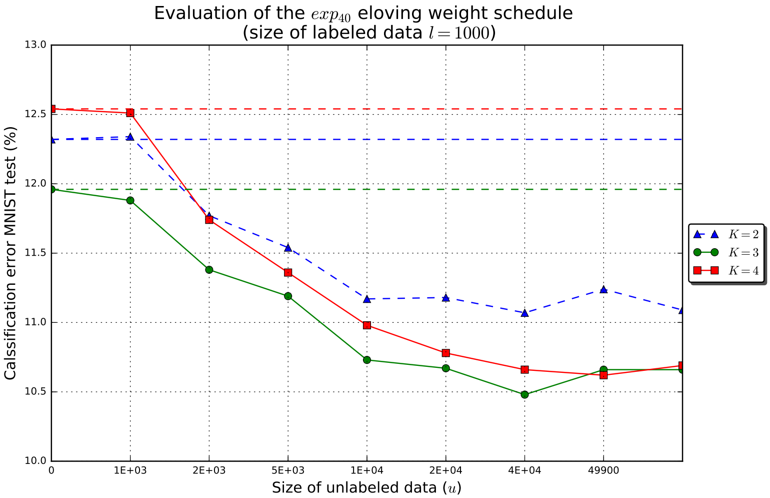
Shallow networks: $(K = 1,\ l = 1E2)$



Shallow networks: $(K = 1,\ l = 1E3)$



Deep networks: exponential schedule ($l = 1E3$)



Conclusion

- An alternative method to the pre-training.

  Parallel transfer learning with evolving weights
- Improve generalization easily.
- Reduce the number of hyper-parameters ($t_1,\ \sigma$)

Perspectives

- Optimization
- **Extension to structured output problems**

$$\begin{aligned}\text{Train cost} &= \textbf{supervised task}\\ &+ \textbf{Input unsupervised task}\\ &+ \textbf{Output unsupervised task}\end{aligned}$$

My PhD work

1. S. Belharbi, R.Hérault, C. Chatelain, S. Adam, ***Deep multi-task learning with evolving weights***, in conference: European Symposium on Artificial Neural Networks (ESANN), 2016
2. S. Belharbi, C. Chatelain, R.Hérault, S. Adam, ***A regularization scheme for structured output problems: an application to facial landmark detection***. 2016. submitted to Pattern Recognition journal (RP). ArXiv: arxiv.org/abs/1504.07550
3. S. Belharbi, R.Hérault, C. Chatelain, R. Modzelewski, S. Adam, M. Chastan, S. Thureau, ***Spotting L3 slice in CT scans using deep convolutional network and transfer learning***. To be submitted to Medical Analysis journal. 2016.

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.

Traditional Machine Learning Problems

$$f : \mathcal{X} \to y$$

- 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

$$f : \mathcal{X} \to \mathcal{Y}$$

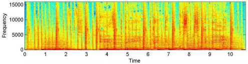
- Inputs $\mathcal{X} \in \mathbb{R}^d$: any type of input
- Outputs $\mathcal{Y} \in \mathbb{R}^{d'}, d' > 1$ a structured object (*dependencies*)

See C. Lampert slides.

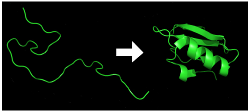
Data $=$ *representation* (*values*) $+$ *structure* (*dependencies*)

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

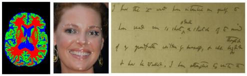
Text: part-of-speech tagging, translation



speech $\rightleftarrows$ *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

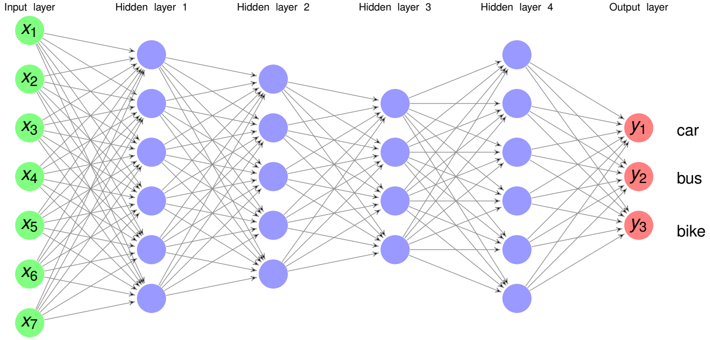
- Perform one single data transformation
- Difficult to deal with *high dimensional* data

Ideal approach

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

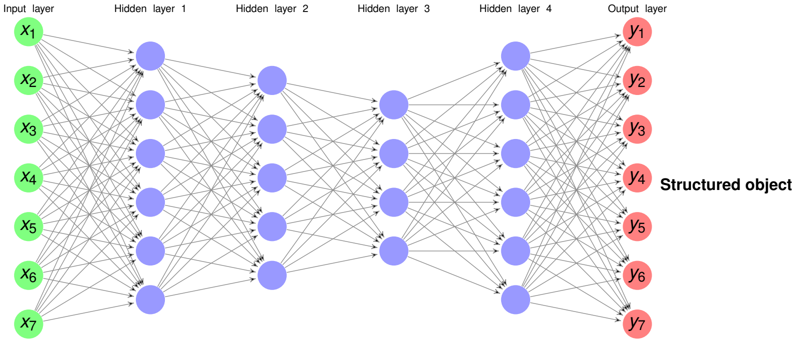
Deep neural networks?

Traditional Deep neural Network

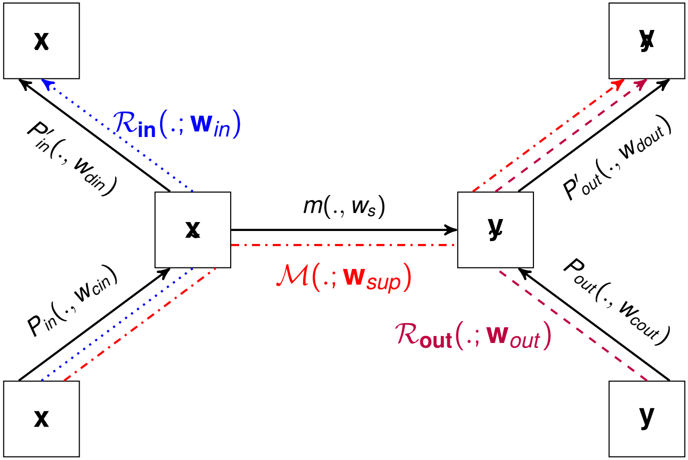


- High dimension data OK
- Multiple data transformation (complex mapping functions) OK
- Structured output problems NO

High dimensional output:



Proposed framework



Proposed framework

$\mathcal{F}$: all the **x**, $\mathcal{L}$: all the **y**, $\mathcal{S}$: all supervised data

Input task

- $$\hat{\mathbf{x}} = \mathcal{R}_{in}\left(\mathbf{x}; \mathbf{w}_{in}\right) = P'_{in}\left(\tilde{\mathbf{x}} = P_{in}\left(\mathbf{x}; \mathbf{w}_{cin}\right); \mathbf{w}_{din}\right) \ ,$$
- $$\mathcal{J}_{in}(\mathcal{F}; \mathbf{w}_{in}) = \frac{1}{\operatorname{card} \mathcal{F}} \sum_{x \in \mathcal{F}} \mathcal{C}_{in}(\mathcal{R}_{in}(\mathbf{x}; \mathbf{w}_{in}), \mathbf{x}) \ .$$

Output task

- $$\hat{\mathbf{y}} = \mathcal{R}_{out}\left(\mathbf{y}; \mathbf{w}_{out}\right) = P'_{out}\left(\tilde{\mathbf{y}} = P_{out}\left(\mathbf{y}; \mathbf{w}_{cout}\right); \mathbf{w}_{dout}\right) \ ,$$
- $$\mathcal{J}_{out}(\mathcal{L}; \mathbf{w}_{out}) = \frac{1}{\operatorname{card} \mathcal{L}} \sum_{y \in \mathcal{L}} \mathcal{C}_{out}(\mathcal{R}_{out}(\mathbf{y}; \mathbf{w}_{out}), \mathbf{y}) \ .$$

Main task

- $$\hat{\mathbf{y}} = \mathcal{M}\left(\mathbf{x}; \mathbf{w}_{sup}\right) = P'_{out}\left(m\left(P_{in}\left(\mathbf{x}; \mathbf{w}_{cin}\right); \mathbf{w}_{s}\right); \mathbf{w}_{dout}\right) \ ,$$
- $$\mathcal{J}_{s}(\mathcal{S}; \mathbf{w}_{sup}) = \frac{1}{\operatorname{card} \mathcal{S}} \sum_{(x,y) \in \mathcal{S}} \mathcal{C}_{s}(\mathcal{M}(x; \mathbf{w}_{sup}), y) \ .$$

Tasks combination

$$\mathcal{J}(\mathcal{D};\mathbf{w}) = \lambda_{sup}(t)\cdot\mathcal{J}_s(\mathcal{S};\mathbf{w}_{sup}) + \lambda_{in}(t)\cdot\mathcal{J}_{in}(\mathcal{F};\mathbf{w}_{in}) + \lambda_{out}(t)\cdot\mathcal{J}_{out}(\mathcal{L};\mathbf{w}_{out}) ,$$

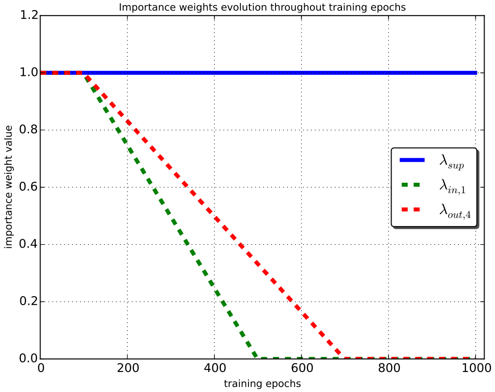


Figure 5: Linear evolution of the importance weights during training.

Framework training

Algorithm 2 Training our framework for one epoch

$\mathcal{D}$ is the *shuffled* training set. B a mini-batch.
for B in $\mathcal{D}$ **do**
  $B_{\mathcal{S}} \Leftarrow$ examples of B that contain both $(\mathbf{x}, \mathbf{y})$
  $B_{\mathcal{F}} \Leftarrow$ all the $\mathbf{x}$ samples of B
  $B_{\mathcal{L}} \Leftarrow$ all the $\mathbf{y}$ samples of B
  Update $\mathbf{w}_{in}$:
   $\rightarrow$ Make a gradient step toward $\mathcal{J}_{in}$ using $B_{\mathcal{F}}$
  Update $\mathbf{w}_{out}$:
   $\rightarrow$ Make a gradient step toward $\mathcal{J}_{out}$ using $B_{\mathcal{L}}$
  Update $\mathbf{w}_{sup}$:
   $\rightarrow$ Make a gradient step toward $\mathcal{J}_{s}$ using $B_{\mathcal{S}}$
  Update λ_{sup}, λ_{in} and λ_{out}
end for

Framework evaluation

Task: Facial landmark detection. Localize 68 points (x,y).

Experiments: setup

- Datasets: LFPW (1035 images), HELEN (2330 images)
- Architecture: MLP with 4 hidden layers: 1025, 2500, 136, 64.
- In: 50x50. Output: 68x2
- Data augmentation, no data augmentation

Experiments: Results (No data augmentation)

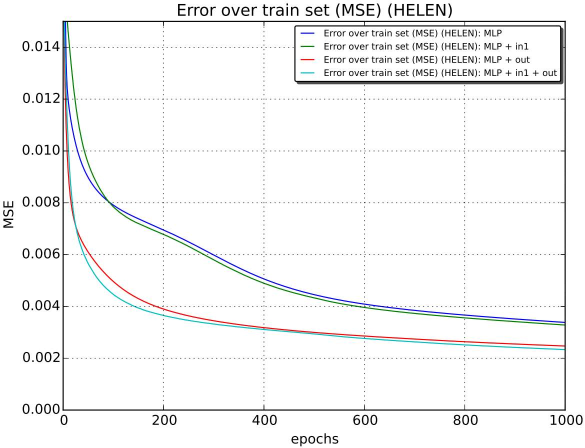


Figure 7: MSE during training epochs over HELEN train set using different training setups for the MLP (no augmentation).

Experiments: Results (No data augmentation)

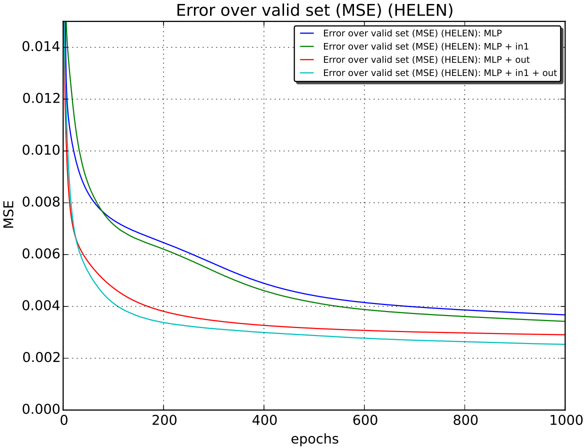


Figure 8: MSE during training epochs over HELEN valid set using different training setups for the MLP (no augmentation).

Experiments: Results (No data augmentation)

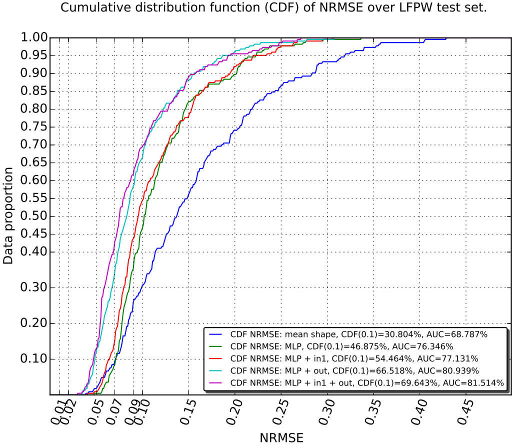


Figure 9: CDF curves of different configurations on LFPW.

Experiments: Results (No data augmentation)

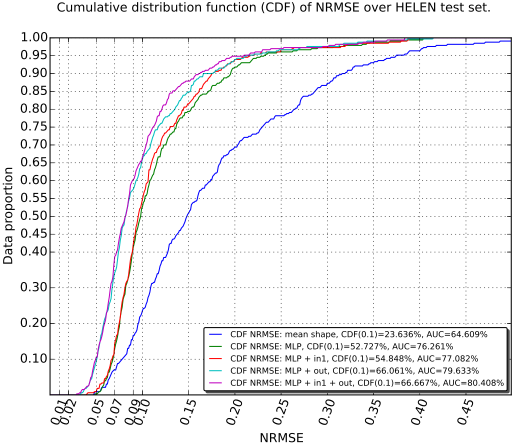


Figure 10: CDF curves of different configurations on HELEN.

Experiments: Results (With data augmentation)

Table 1: MSE over LFPW: train and valid sets, at the end of training with and without data augmentation.

	No augmentation		With augmentation	
	MSE train	MSE valid	MSE train	MSE valid
Mean shape	7.74×10^{-3}	8.07×10^{-3}	7.78×10^{-3}	8.14×10^{-3}
MLP	3.96×10^{-3}	4.28×10^{-3}	-	-
MLP + in	3.64×10^{-3}	3.80×10^{-3}	1.44×10^{-3}	2.62×10^{-3}
MLP + out	2.31×10^{-3}	2.99×10^{-3}	1.51×10^{-3}	2.79×10^{-3}
MLP + in + out	$\mathbf{2.12 \times 10^{-3}}$	$\mathbf{2.56 \times 10^{-3}}$	$\mathbf{1.10 \times 10^{-3}}$	$\mathbf{2.23 \times 10^{-3}}$

Experiments: Results (With data augmentation)

Table 2: **AUC** and **$CDF_{0.1}$** performance over LFPW test dataset with and without data augmentation.

	No augmentation		**with augmentation**	
	AUC	**$CDF_{0.1}$**	**AUC**	**$CDF_{0.1}$**
Mean shape	68.78%	30.80%	77.81%	22.33%
MLP	76.34%	46.87%	-	-
MLP + in	77.13%	54.46%	80.78%	67.85%
MLP + out	80.93%	66.51%	81.77%	67.85%
MLP + in + out	**81.51%**	**69.64%**	**82.48%**	**71.87%**

Table 3: **AUC** and **$CDF_{0.1}$** performance over HELEN test dataset with and without data augmentation.

	No augmentation		**With augmentation**	
	AUC	**$CDF_{0.1}$**	**AUC**	**$CDF_{0.1}$**
Mean shape	64.60%	23.63%	64.76%	23.23%
MLP	76.26%	52.72%	-	-
MLP + in	77.08%	54.84%	79.25%	63.33%
MLP + out	79.63%	66.60%	80.48%	65.15%
MLP + in + out	**80.40%**	**66.66%**	**81.27%**	**71.51%**

Experiments: Visual results

Figure 11: Examples of prediction on LFPW test set. For visualizing errors, red segments have been drawn between ground truth and predicted landmark. Top row: MLP. Bottom row: MLP+in+out. (no data augmentation)

Experiments: Visual results

Figure 12: Examples of prediction on HELEN test set. Top row: MLP. Bottom row: MLP+in+out. (no data augmentation)

Conclusion

- Generic regularization scheme for structured output problems based on transfer learning
- Exploit input/output unlabeled data
- Speedup convergence and improve generalization
- Code at github:
  https://github.com/sbelharbi/structured-output-ae

Perspectives

- Evolve the importance weight according to the train/validation error.
- Explore other evolving schedules (toward automatic schedule)

My PhD work

1. S. Belharbi, R.Hérault, C. Chatelain, S. Adam, ***Deep multi-task learning with evolving weights***, in conference: European Symposium on Artificial Neural Networks (ESANN), 2016
2. S. Belharbi, C. Chatelain, R.Hérault, S. Adam, ***A regularization scheme for structured output problems: an application to facial landmark detection***. 2016. submitted to Pattern Recognition journal (PR). ArXiv: arxiv.org/abs/1504.07550
3. S. Belharbi, R.Hérault, C. Chatelain, R. Modzelewski, S. Adam, M. Chastan, S. Thureau, ***Spotting L3 slice in CT scans using deep convolutional network and transfer learning***. To be submitted to Medical Image Analysis journal. 2016.

The problem: L3 slice localization

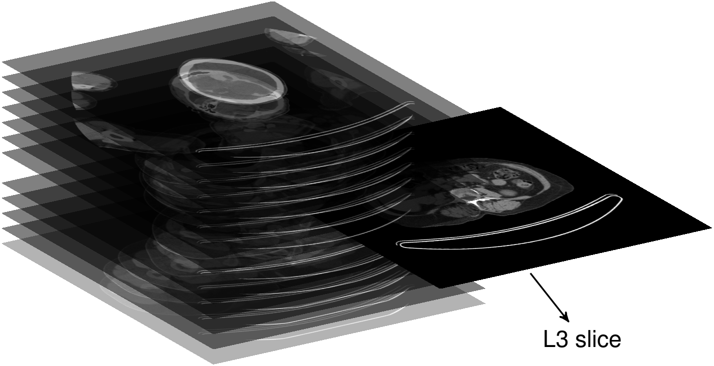


Figure 13: Finding the L3 slice within a whole CT scan.

$\rightarrow$ Over a dataset of **642 CT scans**, we obtained an **average localization error** of **1.82 slice** ($<$ ***5mm***).

The problem: L3 slice localization

Informal statement

Given a CT scan of a part of a body, find the slice which corresponds to the L3 slice from thousands of slices.
The L3 slice contains the 3^{rd} lumbar vertebra.

Difficulties

- Inter-patients **variability**.
- Visual **similarity** of the L3 slice.
- The need to use the **context** to localize the L3 slice.

$\Longrightarrow$ **Machine Learning**

Possible approaches

Classification (discrete value)

Classify each slice for: “L3” or “Not L3”:

- Simple, ☺
- No context, ☹

Sequence labeling

Label all the slices (vertebrae): L1, L2, L3, . . . :

- Global analysis: context, ☺
- Existing work with promising results, ☺
- Requires labeling every slice, ☹

Regression (real value)

Predict the height (position) of the L3 slice inside the CT scan:

- Global analysis: context, ☺
- Requires labeling only the L3 slice position, ☺

Possible approaches: Difficulties

Figure 14: Two slices from the same patient: a L3 (up) and a non L3 (L2) (down). The similar shapes of both vertebraes prevent from taking a robust decision given a single slice.

Regression for L3 detection

Which model?

- Deep learning, Convolutional neural network (CNN).
- No manual feature extraction.
- State of the art in vision.
- Requires fixed input size (when using dense layers).

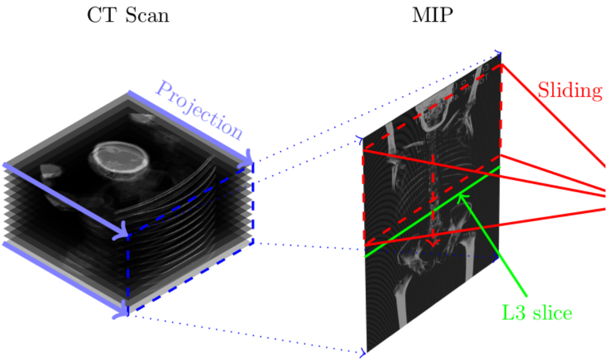
Some numbers . . .

- Input space: $\underbrace{1 \text{ scan} = N \times 512 \times 512}_{\text{Problem 1: large input space}}$, with $400 < N < 1200$.
- Dataset with annotated L3 position: $\underbrace{\text{642 patients}}_{\text{Problem 2: few data}}$. (L3CT1 dataset)
- $\underbrace{\textit{Variability}}_{\text{Problem 3: Different input size}}$ of the height of each scan.

Regression for L3 detection

Problem 1: Input dimension space

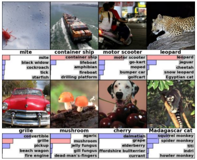
- $131M$ inputs for one example (large input dimension):
  $\Longrightarrow$ Frontal or lateral **Maximum Intensity Projection (MIP)**.
- $512 \times 512 \times N \Longrightarrow 512 \times N$.
- Conserves pertinent information (skeletal structure)



Regression for L3 detection

Problem 2: Few data (642 patients) [1]

- Train CNN from scratch → poor results.
  $\Longrightarrow$ Use pre-trained CNNs over **large datasets**
- Alexnet, GoogleNet, VGG16, VGG19, . . . for **classification**
- Pre-trained models over ImageNet: 14 millions of natural images [Fei-Fei and Russakovsky 2013].



Regression for L3 detection

Problem 2: Few data (642 patients) [2]

$\Longrightarrow$ **Transfer learning**

Exploit **pre-trained filters** over natural images, Next, **refine** them over L3 detection task.

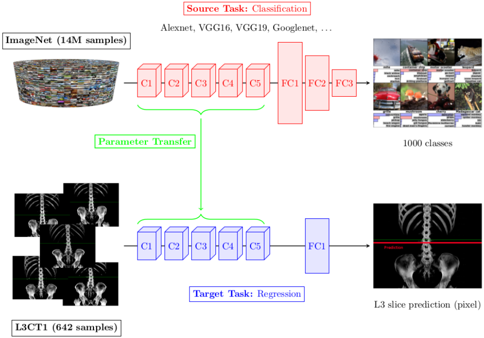


Figure 15: System overview. Layers C_i are Convolutionnal layers, while FC_i denote Full Connected layers. Convolution parameters of previously learnt ImageNet classifier are used as initial values of corresponding L3 regressor layers to overcome the lack of CT examples

Regression for L3 detection

Problem 3: Different input size

- Classical problem
- Use sliding window technique
- Use post-processing

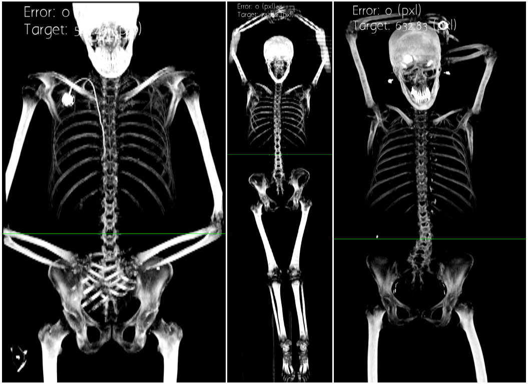


Figure 16: Examples of normalized frontal MIP images with the L3 slice position.

Regression for L3 detection

Problem 3: Different input size

- Classical problem
- Use sliding window technique
- Use post-processing

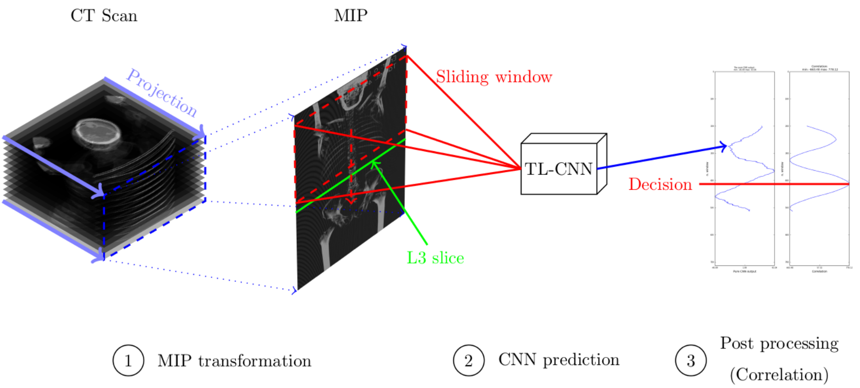


Figure 17: System overview describing the three important stage of our approach : MIP transformation, TL-CNN prediction, and post processing.

Regression for L3 detection

Problem 3: Different input size

- Classical problem
- Use sliding window technique
- Use post-processing: correlation

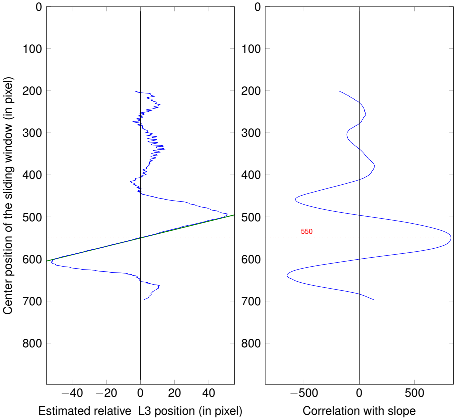


Figure 18: [left]: CNN output sequence obtained using for $H = 400$ and $a = 50$ on a test CT scan. The sequence contains the typical straight line of slope -1 centered on the L3 (the theoretical line is plotted in green), surrounded by random values. [right]: correlation between the CNN output sequence and the theoretical. The maximum of correlation indicates the position of the L3.

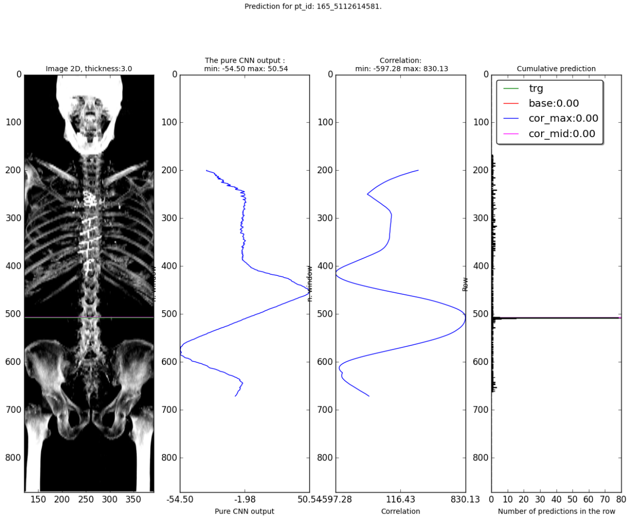
Regression for L3 detection: Quantitative results

Cross-validation:

	CNN4	Alexnet	VGG16	VGG19	Googlenet
fold 0	2.85 ± 2.37	2.21 ± 2.11	2.06 ± 4.39	1.89 ± 1.77	1.81 ± 1.74
fold 1	3.12 ± 2.90	2.44 ± 2.41	1.78 ± 2.09	1.96 ± 2.10	3.84 ± 12.86
fold 2	3.12 ± 3.20	2.47 ± 2.38	1.54 ± 1.54	1.65 ± 1.73	2.62 ± 2.52
fold 3	2.98 ± 2.38	2.42 ± 2.23	1.96 ± 1.62	1.76 ± 1.75	2.22 ± 1.79
fold 4	1.87 ± 1.58	2.69 ± 2.41	1.74 ± 1.96	1.90 ± 1.83	2.20 ± 2.20
Average	2.78 ± 2.48	2.45 ± 2.42	$\mathbf{1.82 \pm 2.32}$	1.83 ± 1.83	2.54 ± 4.22

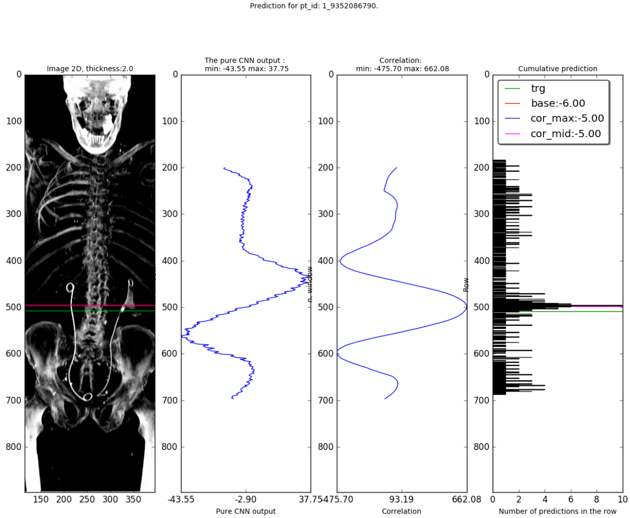
Table 4: Error expressed in slice over all the folds using different models: CNN4 (Homemade model), and Alexnet/VGG16/VGG19/GoogleNet (Pre-trained models).

Regression for L3 detection: Qualitative results



Localization error: 0 coupes.

Regression for L3 detection: Qualitative results



Localization error: 6 coupes.

Regression for L3 detection: Evaluation time

	Number of parameters	Average processing time (seconds/CT scan)
CNN4	55 K	04.46
Alexnet	2 M	06.37
VGG16	14 M	13.28
VGG19	20 M	16.02
GoogleNet	6 M	17.75

Table 5: Number of parameters vs. evaluation time over a GPU (K40).

Can be speedup more by increasing the window stride (without loosing in performance).

VGG16:

- **stride=1**: $\sim$ **13** $\boldsymbol{seconds/CT\ scan}$ with a an error of **1.82** $\pm$ **2.32**.
- **stride=4**: $\sim$ **02** $\boldsymbol{seconds/CT\ scan}$ with a an error of **1.91** $\pm$ **2.69**.

Regression for L3 detection: CNN vs. Radiologists

Setup

1. **New** evaluation set: **43 CT scans** annotated by the same **reference radiologist** (who annotated the L3CT1 dataset).
2. Ask **3 other radiologists** to localize the L3 slice.
3. Perform this experiment **twice**.

Errors (slices) / operator	CNN4	VGG16	Ragiologist #1	Radiologist #2	Radiologist #3
Review1	2.37 ± 2.30	1.70 ± 1.65	0.81 ± 0.97	0.72 ± 1.51	0.51 ± 0.62
Review2	2.53 ± 2.27	1.58 ± 1.83	0.77 ± 0.68	0.95 ± 1.61	0.86 ± 1.30

Table 6: Comparison of the performance of both the automatic systems and radiologists. The L3 annotations given by the reference radiologist vary between the two reviews.

Regression for L3 detection: Conclusion

- Interesting results.
- Adapted pipeline: pre-processing, CNN, post-processing.
- Use of transfer learning alleviates the need of large training set.
- Generic framework: can be easily adapted for detecting other subjects given the required annotation.

My PhD work

1. S. Belharbi, R.Hérault, C. Chatelain, S. Adam, ***Deep multi-task learning with evolving weights***, in conference: European Symposium on Artificial Neural Networks (ESANN), 2016
2. S. Belharbi, C. Chatelain, R.Hérault, S. Adam, ***A regularization scheme for structured output problems: an application to facial landmark detection***. 2016. submitted to Pattern Recognition journal (PR). ArXiv: arxiv.org/abs/1504.07550
3. S. Belharbi, R.Hérault, C. Chatelain, R. Modzelewski, S. Adam, M. Chastan, S. Thureau, ***Spotting L3 slice in CT scans using deep convolutional network and transfer learning***. To be submitted to Medical Image Analysis journal (MIA). 2016.

Questions

Thank you for your attention,

Questions?

soufiane.belharbi@insa-rouen.fr