UTLC
Unsupervised Transfer Learning Challenge

Grégoire Mesnil\textsuperscript{1,2}, Yann Dauphin\textsuperscript{1}, Xavier Glorot\textsuperscript{1}, Salah Rifai\textsuperscript{1}, Yoshua Bengio\textsuperscript{1} \textit{et al.}

\textsuperscript{1} LISA, Université de Montréal, Canada
\textsuperscript{2} LITIS, Université de Rouen, France

July 2\textsuperscript{nd} 2011
Plan

1 Introduction

2 Deep Architecture
   • Preprocessing
   • Feature Extraction
   • Postprocessing

3 Results

4 Summary
UTL Challenge
Presentation

Dates:
Phase 1: Unsupervised Learning; start: January 3, end: March 4.
Phase 2: Transfer Learning; start: March 4, end: April 15.

Five different Data sets:

<table>
<thead>
<tr>
<th>data set</th>
<th># samples</th>
<th>dimension</th>
<th>sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVICENNA</td>
<td>150205</td>
<td>120</td>
<td>0 %</td>
</tr>
<tr>
<td>HARRY</td>
<td>69652</td>
<td>5000</td>
<td>98 %</td>
</tr>
<tr>
<td>RITA</td>
<td>111808</td>
<td>7200</td>
<td>1 %</td>
</tr>
<tr>
<td>SYLVESTER</td>
<td>572820</td>
<td>100</td>
<td>0 %</td>
</tr>
<tr>
<td>TERRY</td>
<td>217034</td>
<td>47236</td>
<td>99 %</td>
</tr>
</tbody>
</table>
**ALC**: Area under Learning Curve

![Graph showing ALC (Area under Learning Curve) with data points for 1 to 64 samples per class.](image)

1 to 64 samples per class
How to evaluate the performance of one model without any label or prior knowledge on the training set?
How to evaluate the performance of one model without any label or prior knowledge on the training set?

- proxy: ALC Valid versus Test (Phase 1)
How to evaluate the performance of one model without any label or prior knowledge on the training set?

- proxy: ALC Valid versus Test (Phase 1)
- valid ALC returned by the competition servers (Phase 1 & 2)
How to evaluate the performance of one model without any label or prior knowledge on the training set?

- proxy: ALC Valid versus Test (Phase 1)
- valid ALC returned by the competition servers (Phase 1 & 2)
- ALC with the given labels (Phase 2)
How to evaluate the performance of one model without any label or prior knowledge on the training set?

- proxy: ALC Valid versus Test (Phase 1)
- **valid ALC returned by the competition servers (Phase 1 & 2)**
- ALC with the given labels (Phase 2)
How to evaluate the performance of one model without any label or prior knowledge on the training set?

- proxy: ALC Valid versus Test (Phase 1)
- **valid ALC returned by the competition servers** (Phase 1 & 2)
- ALC with the given labels (Phase 2)

From phase 1 to phase 2, we over-explored the hyperparameters of the next models to grab the 1st place.
Deep Architecture

Stack different blocks

We used this template:

- **Pre-processing**: PCA w/wo whitening, Contrast Normalization, Uniformization
- **Feature Extraction**: Rectifiers, DAE, CAE, $\mu$-ss-RBM
- **Post-processing**: Transductive PCA
Deep Architecture
Stack different blocks

We used this template:

- **Pre-processing**: PCA w/wo whitening, Contrast Normalization, Uniformization
- **Feature Extraction**: Rectifiers, DAE, CAE, $\mu$-ss-RBM
- **Post-processing**: Transductive PCA
Deep Architecture
Stack different blocks

We used this template:

- **Pre-processing**: PCA w/wo whitening, Contrast Normalization, Uniformization
- **Feature Extraction**: Rectifiers, DAE, CAE, \( \mu \)-ss-RBM
- **Post-processing**: Transductive PCA
Deep Architecture
Stack different blocks

We used this template:

1. **Pre-processing**: PCA w/wo whitening, Contrast Normalization, Uniformization
2. **Feature Extraction**: Rectifiers, DAE, CAE, $\mu$-ss-RBM
3. **Post-processing**: Transductive PCA
Plan

1. Introduction

2. Deep Architecture
   - Preprocessing
   - Feature Extraction
   - Postprocessing

3. Results

4. Summary
Preprocessing

Given a training set \( D = \{ x^{(j)} \}_{j=1}^n \) where \( x^{(j)} \in \mathbb{R}^d \):

- **Uniformization** (t-IDF)
  Rank all the \( x_i^{(j)} \) and map them to [0, 1]

- **Contrast Normalization**
  For each \( x^{(j)} \), compute its mean \( \mu^{(j)} = \sum_{i=1}^d x_i^{(j)} \) and its deviation \( \sigma^{(j)} \).
  \( x^{(j)} \leftarrow (x^{(j)} - \mu^{(j)})/\sigma^{(j)} \)

- **Principal Component Analysis**
  with/without whitening
  i.e divide by the squared root eigen value or not.
Given a training set $D = \{x^{(j)}\}_{j=1}^n$ where $x^{(j)} \in \mathbb{R}^d$ :

- **Uniformization** (t-IDF)
  Rank all the $x_i^{(j)}$ and map them to $[0, 1]$

- **Contrast Normalization**
  For each $x^{(j)}$, compute its mean $\mu^{(j)} = \sum_{i=1}^d x_i^{(j)}$ and its deviation $\sigma^{(j)}$. $x^{(j)} \leftarrow (x^{(j)} - \mu^{(j)}) / \sigma^{(j)}$

- **Principal Component Analysis**
  with/without whitening
  i.e divide by the squared root eigen value or not.
Preprocessing

Given a training set \( D = \{ x^{(j)} \}_{j=1}^{n} \) where \( x^{(j)} \in \mathbb{R}^d \):

- **Uniformization** (t-IDF)
  
  Rank all the \( x_i^{(j)} \) and map them to \([0, 1]\)

- **Contrast Normalization**
  
  For each \( x^{(j)} \), compute its mean \( \mu^{(j)} = \sum_{i=1}^{d} x_i^{(j)} \) and its deviation \( \sigma^{(j)} \).
  
  \[
  x^{(j)} \leftarrow (x^{(j)} - \mu^{(j)}) / \sigma^{(j)}
  \]

- **Principal Component Analysis**
  
  with/without whitening
  
  i.e divide by the squared root eigen value or not.
Preprocessing

Given a training set $\mathcal{D} = \{x^{(j)}\}_{j=1}^{n}$ where $x^{(j)} \in \mathbb{R}^d$:

- **Uniformization** (t-IDF)
  Rank all the $x_i^{(j)}$ and map them to $[0, 1]$

- **Contrast Normalization**
  For each $x^{(j)}$, compute its mean $\mu^{(j)} = \sum_{i=1}^{d} x_i^{(j)}$ and its deviation $\sigma^{(j)}$. $x^{(j)} \leftarrow (x^{(j)} - \mu^{(j)})/\sigma^{(j)}$

- **Principal Component Analysis**
  with/without whitening
  i.e divide by the squared root eigen value or not.
Plan

1 Introduction

2 Deep Architecture
   - Preprocessing
   - Feature Extraction
   - Postprocessing

3 Results

4 Summary
Feature Extraction

$\mu$-ss-RBM

$\mu$-Spike & Slab Restricted Boltzmann Machine modelizes the interaction between three random vectors:

1. visible vector $v$ representing the observed data
2. binary “spike” variables $h$
3. real-valued “slab” variables $s$
Feature Extraction

\(\mu\)-ss-RBM

\textbf{\(\mu\)-Spike & Slab Restricted Boltzmann Machine} modelizes the interaction between three random vectors:

1. visible vector \(v\) representing the observed data
2. binary “\textit{spike}” variables \(h\)
3. real-valued “\textit{slab}” variables \(s\)

It is defined by the energy function:

\[
E(v, s, h) = -\sum_{i=1}^{N} v^T W_i s_i h_i + \frac{1}{2} v^T \left( \Lambda + \sum_{i=1}^{N} \Phi_i h_i \right) v \\
+ \sum_{i=1}^{N} \frac{1}{2} s_i^T \alpha_i s_i - \sum_{i=1}^{N} \mu_i^T \alpha_i s_i h_i - \sum_{i=1}^{N} b_i h_i + \sum_{i=1}^{N} \mu_i^T \alpha_i \mu_i h_i,
\]

In training, we use **Persistent Contrastive Divergence** with a **Gibbs Sampling** procedure.
**Feature Extraction**

$\mu$-ss-RBM


Pools of filters learned on CIFAR-10
A **Denoising Autoencoder** is an autoencoder trained to **denoise** artificially corrupted training samples.

**Corruption** e.g. \( \tilde{x} = x + \epsilon \) where \( \epsilon \sim \mathcal{N}(0, \sigma^2) \)

Encoder: \( h(\tilde{x}) = s(W\tilde{x} + b) \) where \( s \) is the sigmoid function.

Decoder: \( r(\tilde{x}) = W^T h(\tilde{x}) + b' \) (tied weights).
A **Denoising Autoencoder** is an autoencoder trained to denoise artificially corrupted training samples.

**Corruption** e.g. \( \tilde{x} = x + \epsilon \) where \( \epsilon \sim \mathcal{N}(0, \sigma^2) \)

Encoder: \( h(\tilde{x}) = s(W\tilde{x} + b) \) where \( s \) is the sigmoid function.

Decoder: \( r(\tilde{x}) = W^T h(\tilde{x}) + b' \) (tied weights).

Different loss functions to be minimized using stochastic gradient descent:

- \( \| r(\tilde{x}) - x \|^2_2 \) (linear reconstruction and MSE)
- \( \| s(r(\tilde{x})) - x \|^2_2 \) (non-linear reconstruction)
- \( -\sum_i x_i \log r(\tilde{x}_i) - (1 - x_i) \log(1 - r(\tilde{x}_i)) \) (cross-entropy)
A Contractive Autoencoder encourages an invariance of the representation by penalizing the sensitivity of its encoder to the training inputs characterized with:

\[
\| J_f(x) \|_F^2 = \sum_{ij} \left( \frac{\partial h_j(x)}{\partial x_i} \right)^2
\]
A **Contractive Autoencoder** encourages an **invariance** of the representation by penalizing the **sensitivity** of its encoder to the training inputs characterized with:

\[ \| J_f(x) \|_F^2 = \sum_{ij} \left( \frac{\partial h_j(x)}{\partial x_i} \right)^2 \]

To avoid useless constant representations, this term is counterbalanced by a **reconstruction error** and use **tied weights** (decoder and encoder share the same weights):

\[ \| s(r(x)) - x \|_2^2 + \lambda \| J_f(x) \|_F^2 \]

where \( \lambda \) controls the tradeoff between both penalties.
Feature Extraction
Contractive Autoencoders

more details in S. Rifai, P. Vincent, X. Muller, X. Glorot and Y. Bengio
Contractive Auto-Encoders: Explicit Invariance During Feature Extraction, ICML 2011.

Random selection of 4000 filters learned on CIFAR-10
**Feature Extraction**

**Rectifiers** use the activation function $\max(0, Wx + b)$ and therefore create sparse representation with true zeros. Those are used to be trained as Denoising Autoencoders.

**Feature Extraction**

**Rectifiers** use the activation function $\text{max}(0, Wx + b)$ and therefore create sparse representation with **true** zeros. Those are used to be trained as Denoising Autoencoders.

**more details** in X.Glorot, A.Bordes and Y.Bengio, *Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach*, ICML 2011.

For huge sparse distributions, e.g:
- input dimension is 50,000
- embedding dimension is 1,000

$\Rightarrow$ **decoding** requires 50,000,000 operations. **Expensive**...
Reconstruction sampling: reconstruct all the non-zeros elements and a small random subset of the zeros elements and speed-up training.
Reconstruction sampling: reconstruct all the non-zeros elements and a small random subset of the zeros elements and **speed-up** training.

Plan

1. Introduction

2. Deep Architecture
   - Preprocessing
   - Feature Extraction
   - Postprocessing

3. Results

4. Summary
The feature extraction is performed on the training set while a **Transductive PCA** is a PCA trained not on the training set but on the valid (or test) set.

- Trained on the representation learned by the feature extraction process.
- Only retains dominant variations on the test or validation test.
- Validation of the number of components on the valid set (assume there is the same number of classes in the test and valid set).
Postprocessing
Transductive PCA

The feature extraction is performed on the training set while a Transductive PCA is a PCA trained not on the training set but on the valid (or test) set.

- Trained on the representation learned by the feature extraction process.
- Only retains dominant variations on the test or validation test.
- Validation of the number of components on the valid set (assume there is the same number of classes in the test and valid set).
Postprocessing
Transductive PCA

The feature extraction is performed on the training set while a **Transductive PCA** is a PCA trained not on the training set but on the valid (or test) set.

- Trained on the **representation** learned by the feature extraction process.
- Only retains dominant variations on the test or validation test.
- Validation of the **number of components** on the valid set (assume there is the same number of classes in the test and valid set).
Postprocessing
Transductive PCA

The feature extraction is performed on the training set while a Transductive PCA is a PCA trained not on the training set but on the valid (or test) set.

- Trained on the representation learned by the feature extraction process.
- Only retains dominant variations on the test or validation test.
- Validation of the number of components on the valid set (assume there is the same number of classes in the test and valid set).
Computation
How much time?

From preprocessing to postprocessing, the time spent for training is at most 12 hours for every model...
Computation
How much time?

From preprocessing to postprocessing, the time spent for training is at most 12 hours for every model...
Once you have found the good hyperparameters! And there is a lot.
From preprocessing to postprocessing, the time spent for training is at most 12 hours for every model...

Once you have found the good hyperparameters! And there is a lot.

**Software**: Theano (Python Library)
**Hardware**: GPU (Geforce GTX 580)

[Theano](http://deeplearning.net/)

---

**Computation**

**How much time?**
input dimension is 5,000 (98% sparse) **Human actions**
input dimension is 47,236 (99% sparse) **Natural Language Processing**
input dimension is 100 (no sparsity) **Ecology**

**Stacking effect PCA-8**
input dimension is 100 (no sparsity) **Ecology**

**Stacking effect PCA-8 // CAE-6**
input dimension is 100 (no sparsity) **Ecology**

**Stacking effect PCA-8 // CAE-6 // CAE-6**
input dimension is 100 (no sparsity) Ecology

Stacking effect PCA-8 // CAE-6 // CAE-6 // PCA-1
input dimension is 100 (no sparsity) **Ecology**

**Stacking effect compared to raw data**

![Diagram showing area under the ROC curve (AUC) vs. Log2(Number of training examples). The AUC value for SylvesterVALID is 0.7279.](image)
Overall
Best models

ALC computed at each stage on the five data sets.
We proposed a successful deep approach decomposed in three steps:

- Preprocessing
- Feature Extraction
- Postprocessing

We ranked 4th in the phase 1 and 1st in the phase 2.

more details in our JMLR paper:
UTLC
Unsupervised Transfer Learning Challenge

Grégoire Mesnil\textsuperscript{1,2}, Yann Dauphin\textsuperscript{1}, Xavier Glorot\textsuperscript{1}, Salah Rifai\textsuperscript{1}, Yoshua Bengio\textsuperscript{1} \textit{et al.}

\textsuperscript{1} LISA, Université de Montréal, Canada
\textsuperscript{2} LITIS, Université de Rouen, France

Thanks for your attention. Questions?