

# Deep multi-task learning with evolving weights

Machine learning - computer vision

published in *European Symposium on Artificial Neural Networks (ESANN 2016)*



Soufiane Belharbi



Romain Hérault



Clément Chatelain



Sébastien Adam

[soufiane.belharbi@insa-rouen.fr](mailto:soufiane.belharbi@insa-rouen.fr)

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



JDD, Le Havre. 14 June, 2016

# 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

## 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

## 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**

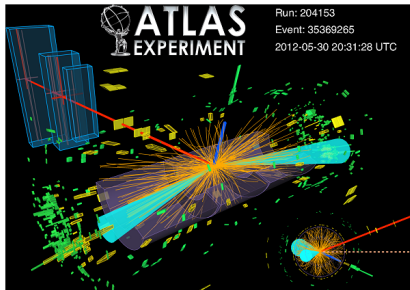
- 1 Train the model using data examples  $(\mathbf{x}, \mathbf{y})$
- 2 Predict the  $\mathbf{y}_{new}$  for the new coming  $\mathbf{x}_{new}$

# 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)

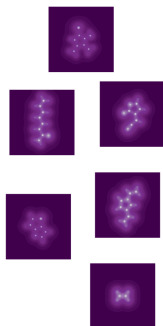


⇒ Use ML models to determine the probability of the event being of interest.

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

# ML in quantum chemistry

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



See Stéphane Mallat et al. work: [https://matthewhirn.files.wordpress.com/2016/01/hirn\\_pasc15.pdf](https://matthewhirn.files.wordpress.com/2016/01/hirn_pasc15.pdf)

# How to estimate $f_w$ ?

## Models

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

## Different models to learn $f_w$

- 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 $f_w$ ?

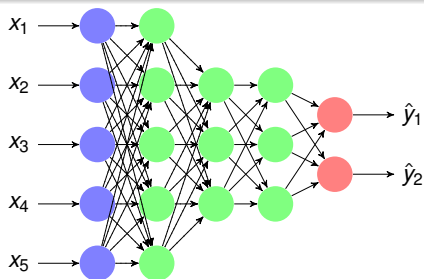
## Models

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

## Different models to learn $f_w$

- 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.

# Deep neural networks (DNN)

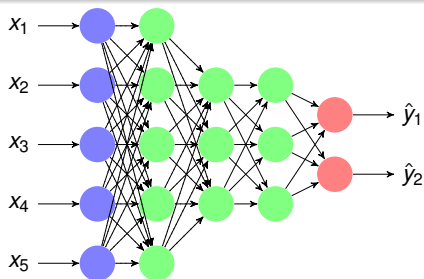


- State of the art in many task: computer vision, natural language processing.
- Training requires large data
- To speed up the training: use GPUs cards
- Training **deep** neural networks is **difficult**
  - ⇒ Vanishing gradient
  - ⇒ More parameters ⇒ Need more data

Some solutions:

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

# Deep neural networks (DNN)

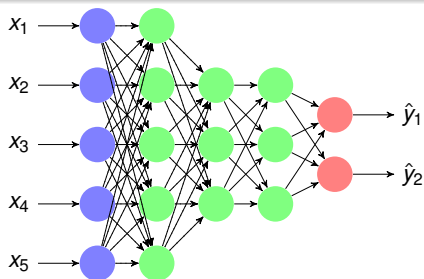


- State of the art in many task: computer vision, natural language processing.
- Training requires large data
- To speed up the training: use GPUs cards
- Training **deep** neural networks is **difficult**
  - ⇒ Vanishing gradient
  - ⇒ More parameters ⇒ Need more data

Some solutions:

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

# Deep neural networks (DNN)



- State of the art in many task: computer vision, natural language processing.
- Training requires large data
- To speed up the training: use GPUs cards
- Training **deep** neural networks is **difficult**
  - ⇒ Vanishing gradient
  - ⇒ More parameters ⇒ Need more data

Some solutions:

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

# Semi-supervised learning

General case:

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

E.g: medical images

⇒ semi-supervised learning:

Exploit unlabeled data to improve the **generalization**

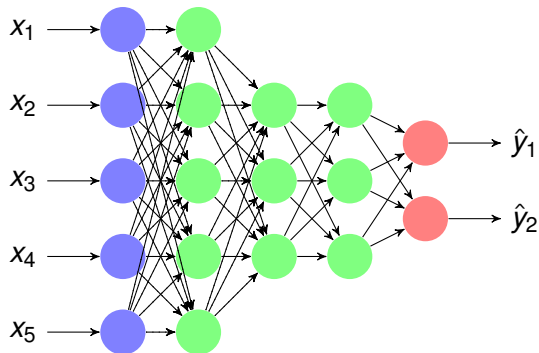
# Pre-training and semi-supervised learning

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

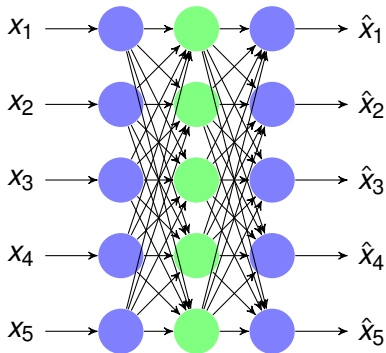


A DNN to train

# Layer-wise pre-training: auto-encoders

## 1) Step 1: Unsupervised layer-wise training

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

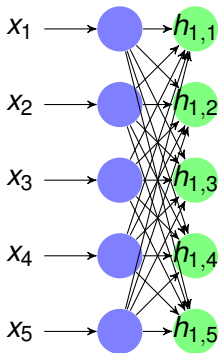




# Layer-wise pre-training: auto-encoders

## 1) Step 1: Unsupervised layer-wise training

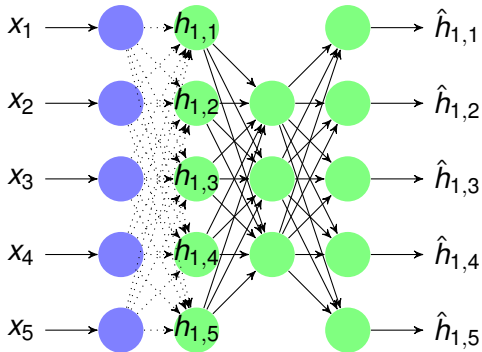
Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



## Layer-wise pre-training: auto-encoders

## 1) Step 1: Unsupervised layer-wise training

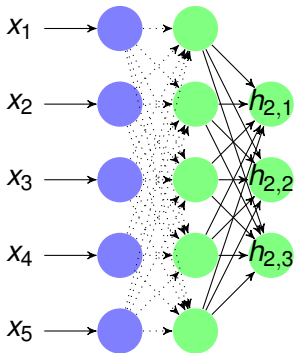
Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



## Layer-wise pre-training: auto-encoders

## 1) Step 1: Unsupervised layer-wise training

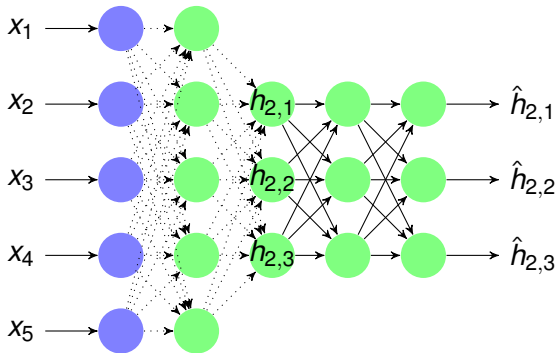
Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



# Layer-wise pre-training: auto-encoders

## 1) Step 1: Unsupervised layer-wise training

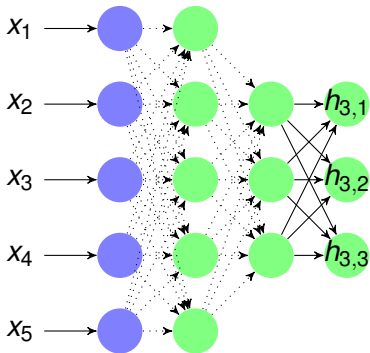
Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



# Layer-wise pre-training: auto-encoders

## 1) Step 1: Unsupervised layer-wise training

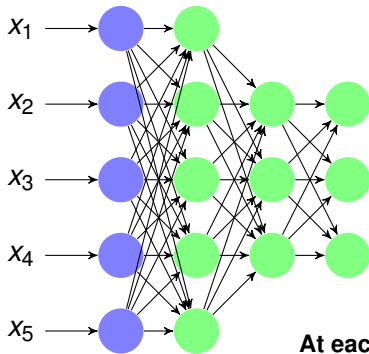
Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



# Layer-wise pre-training: auto-encoders

## 1) Step 1: Unsupervised layer-wise training

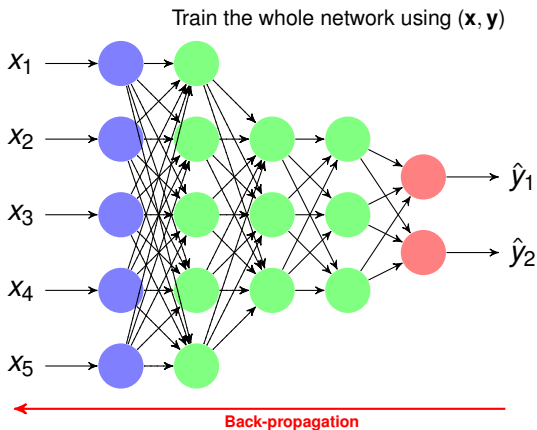
Train layer by layer **sequentially** using **only x** (labeled or unlabeled)



- ⇒ When to stop training?
- ⇒ What hyper-parameters to use?
- ⇒ How to make sure that the 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
- ⇒ 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
  - ⇒ 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?

⇒ **Sequential** transfer learning

Possible solution:

⇒ **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

Train cost = supervised task + unsupervised task  
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,  $\lambda_S + \lambda_r = 1$ .

### Problems

- How to **fix**  $\lambda_S, \lambda_r$ ?
- 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,  $\lambda_S + \lambda_r = 1$ .

## Problems

- How to **fix**  $\lambda_S, \lambda_r$ ?
- 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,  $\lambda_S + \lambda_r = 1$ .

## Problems

- How to **fix**  $\lambda_S, \lambda_r$ ?
- At the end of the training, only  $\mathcal{J}_S$  should matters

## Tasks combination with evolving weights (our contribution)

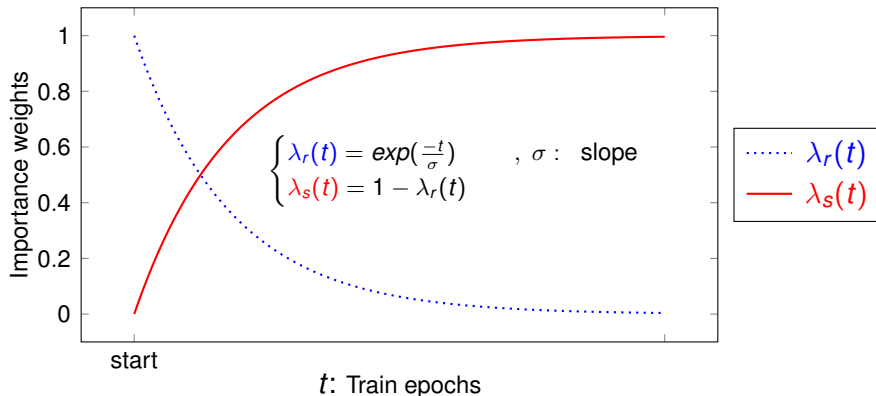
$$\mathcal{J}(\mathcal{D}; \{\mathbf{w}_{Sh}, \mathbf{w}_S, \mathbf{w}_r\}) = \lambda_S(t) \cdot \mathcal{J}_S(\mathcal{D}; \{\mathbf{w}_{Sh}, \mathbf{w}_S\}) + \lambda_r(t) \cdot \mathcal{J}_r(\mathcal{D}; \{\mathbf{w}_{Sh}, \mathbf{w}_r\}) .$$

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

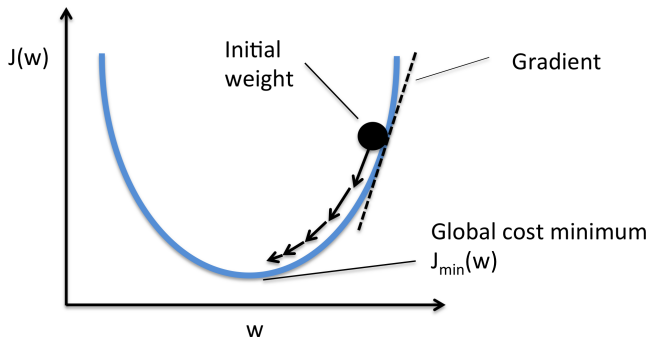
## Tasks combination with evolving weights

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

Exponential schedule



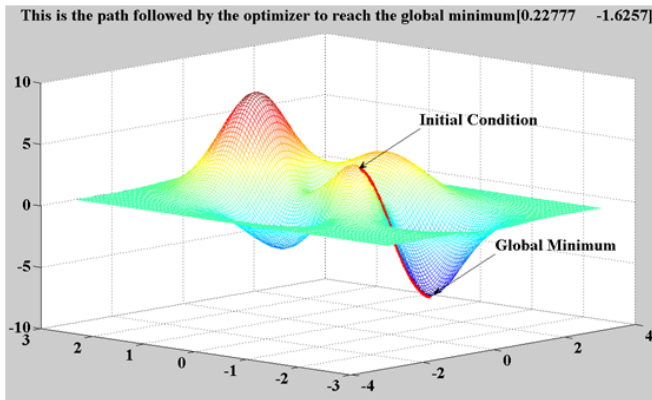
## Optimization using gradient descent (GD)



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



# Optimization using gradient descent (GD)



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

# 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,  $\lambda_s(t) + \lambda_r(t) = 1$ .

---

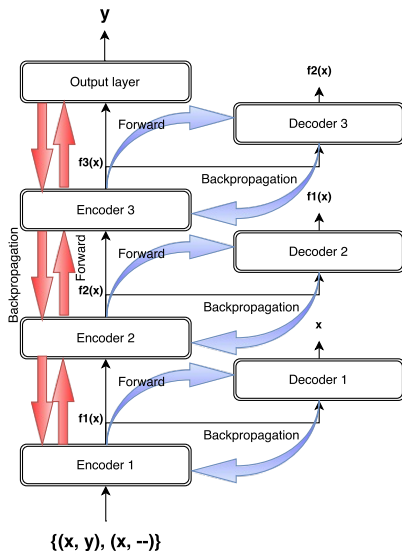
### Algorithm 1 Training our model for one epoch

---

- 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]

# Overview of the model

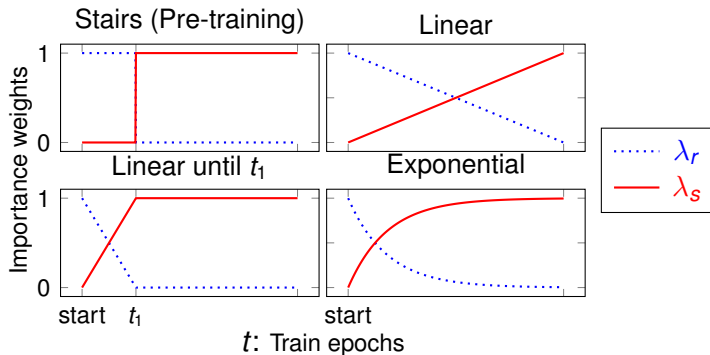


# Experimental protocol

**Objective:** Compare Training DNN using different approaches:

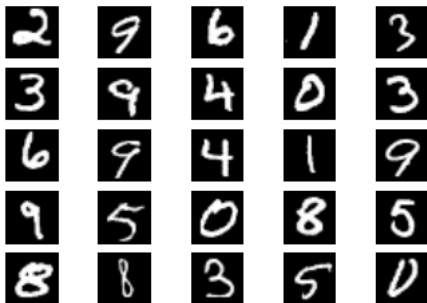
- No pre-training (base-line)
- With pre-training (Stairs schedule)
- Parallel transfer learning (proposed approach)

**Studied evolving weights schedules:**



## MNIST dataset: digits dataset

Random Sampling of MNIST

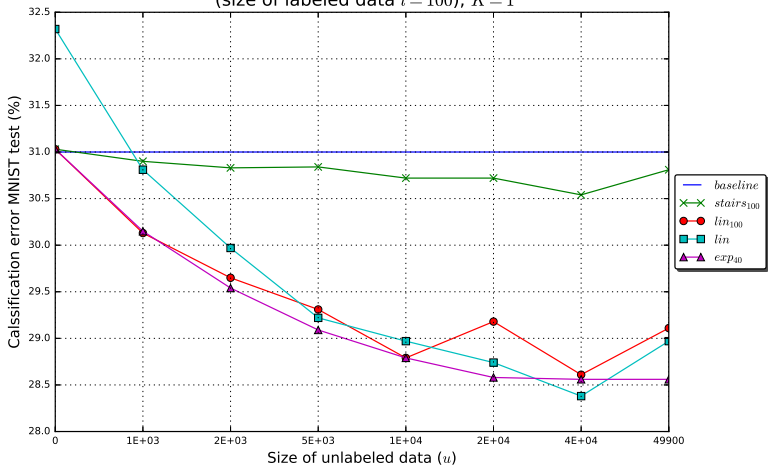


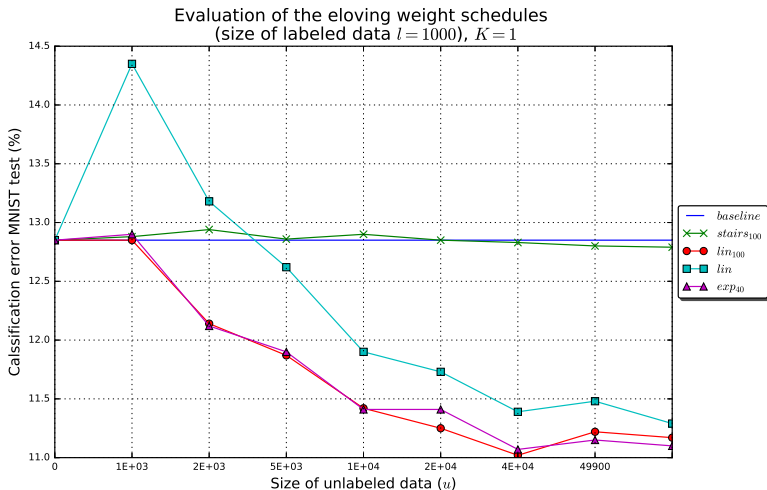
Train set: 60 000 samples.

Test set: 10 000 samples.

# 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
  - $\sigma$ : 40

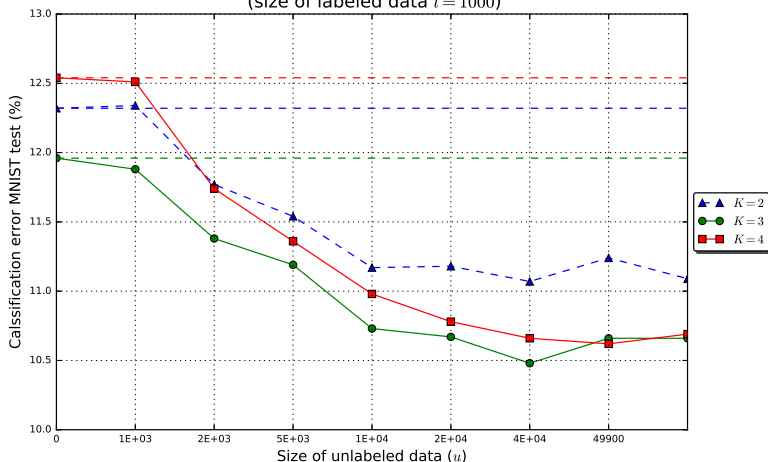
Shallow networks: ( $K = 1$ ,  $l = 1E2$ )Evaluation of the evolving weight schedules  
(size of labeled data  $l = 100$ ),  $K = 1$ 

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



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

Evaluation of the  $exp_{40}$  evolving weight schedule  
(size of labeled data  $l = 1000$ )



# 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

- Evolve the importance weight according to the train/validation error.
- Explore other evolving schedules (toward automatic schedule)
- Adjust the learning rate: Adadelta, Adagrad, RMSProp
- **Extension to structured output problems**

Train cost = **supervised task**  
+ **Input unsupervised task**  
+ **Output unsupervised task**

## Questions

Thank you for your attention,

Questions?