Réseaux de Neurones Profonds, Apprentissage de Représentations

Thierry Artières

ECM, Equipe QARMA @LIS, AMU, CNRS

October 9, 2018

Outline

Recurrent NNs

Main features

- May handle data of different dimension w.r.t. traditional FeedForward Models
- Useful for dealing with
  - Sequences : Text (sentiment, translation, parsing...), Speech, Videos, Time series..
  - Trees : Syntaxic parse tree etc
- State space models' like architecture
  - Links to state space models

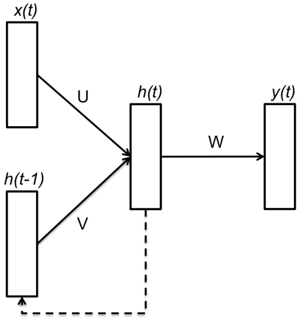
$$s(t) = f(s(t-1), x(t)) \text{ and } y(t) = g(s(t))$$

  - The state at time t resumes the whole history of inputs

Recurrent NNs (RNNs)

RNNs in general

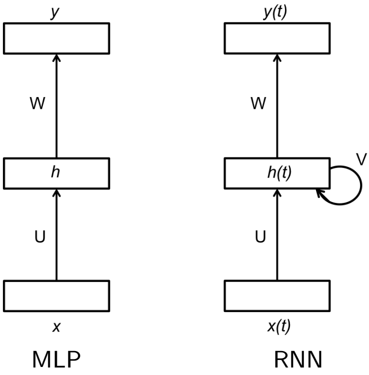
- May handle data of different dimension w.r.t. traditional FeedForward Models (Sequences, trees, ...)
- A recurrent neural network is a NN with cycles in its connections
- Much more powerful than acyclic models (FeedForward NNs such as MLPs)
- Not all architectures work well. Few popular ones.



Feedforward and Recurrent NNs

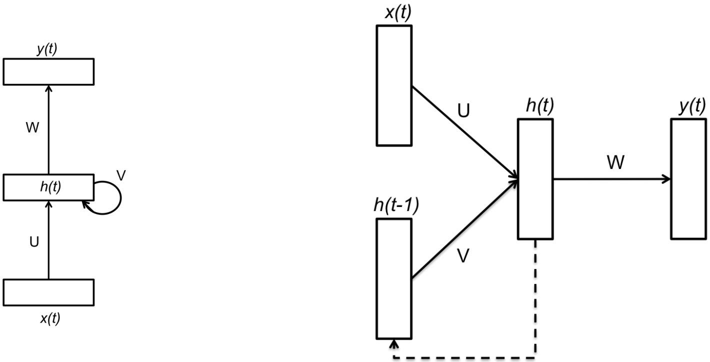
RNNs in general

- A recurrent neural network is a NN with cycles in its connections
- RNNs are dynamical systems
- Much more powerful than acyclic models (FeedForward NNs such as MLPs)
- Today RNNs are specific recurrent architectures. Not all architectures work well..



Popular RNNs

Two representations of the same model



Inference

Algorithm

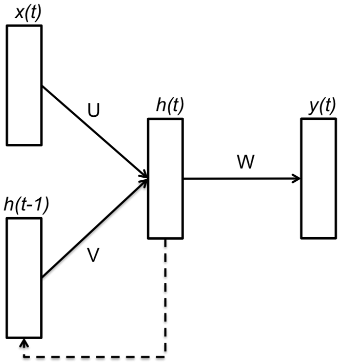
- Start with null state $h(0) = 0$
- Iterate

$$h(t) = tanh((Vh(t-1) + Ux(t))$$

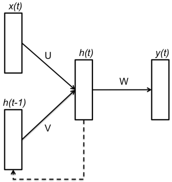
$$y(t) = tanh(Wh(t))$$

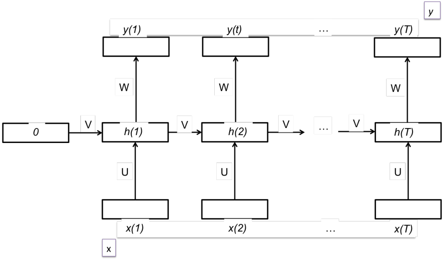
$\Rightarrow$ Inference is done as a forward propagation in a FeedForward NN

- This model computes an output sequence from an input sequence



Inference and learning through unfolding the RNN





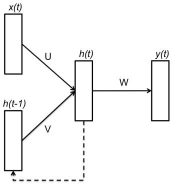
Inference: Forward propagation in the FeedForward unfolded RNN

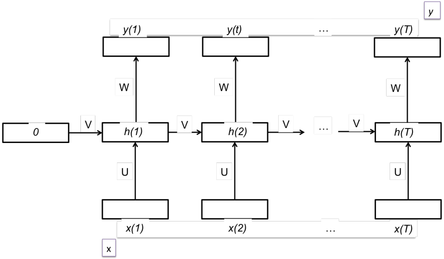
- Start with null state $h(0) = 0$
- Iterate

$$h(t) = g(V \times h(t-1) + U \times x(t)$$

$$y(t) = g(W \times h(t))$$

Inference and learning through unfolding the RNN

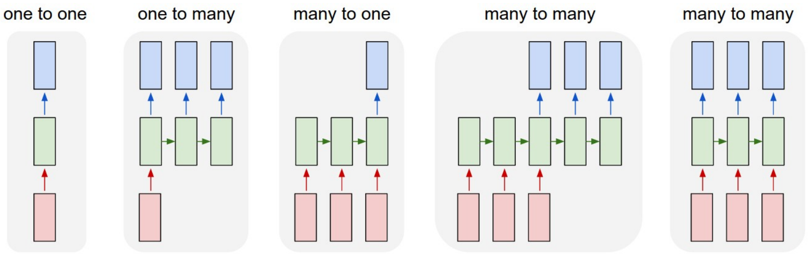




Learning: Back-propagation in the FeedForward unfolded RNN

- Unfold the model
- Backpropagate the gradient in the whole network
- Sum the gradient corresponding to all shared parameters and unshared parameters (possibly the last layer)
- Apply Gradient Optimization Update rule on all parameters

Various settings



- One to One : MLP, CNN ...
- One to Many : Generation of a sequential process (speech, handwriting ...)
- Many to one : Sequence classification (e.g. activity recognition)
- Asynchronous Many to many : Machine Translation
- Synchronous Many to Many : POS tagging, Speech recognition...

Example: Using RNNs for Language models (LM)

- A LM should allow computing the likelihood of sentences $p(w_1, ..., w_T)$ using a limited number of parameters
- Traditional n-gram language models use n-grams (e.g. bigams) assuming fixed and limited past dependencies...

$$p\left(w_t | w_{t-1}, w_{t-2}, ...w_{t-n+1}\right)$$

- ... to compute sentence likelihood. E.g. using bigrams:

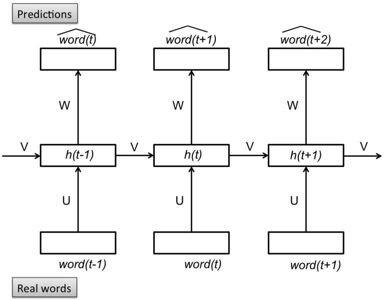
$$p(w_1, ..., w_T) = p(w_1) \times \prod_{t=2}^{T} p(w_t | w_{t-1})$$

Example: Using RNNs for Language models (LM)

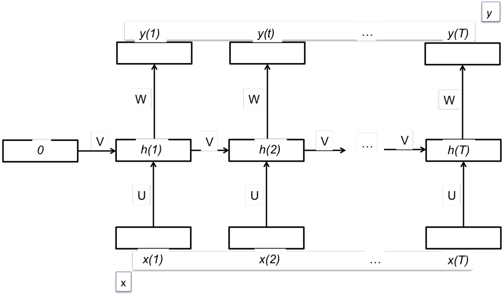
- RNN based Language Models use an, a priori, unlimited past by computing

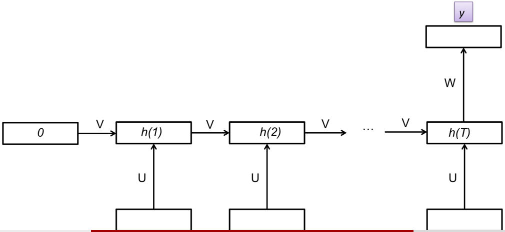
  $$p\left(w_t \middle| c\left(w_{t-1}, w_{t-2}, \ldots w_1\right)\right)$$

  where $c\left(w_{t-1}, w_{t-2}, \ldots w_1\right)$ stands for a fixed dimension representation of the context computed from the full past
- This corresponds to a recursive computation of a context information, s_t, and of the computation of an output $y_t\left(w_t\right)$ based on the full history.

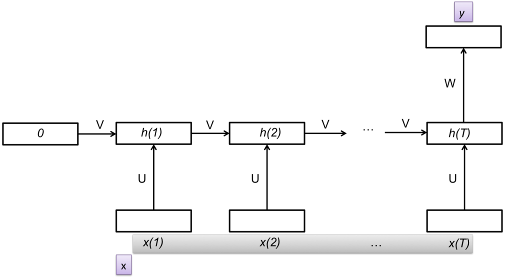


Sequence labeling vs Sequence Classification





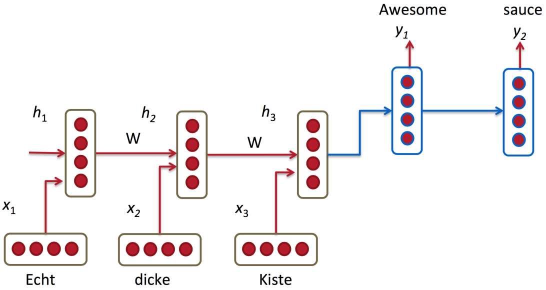
Unfolding the RNN: classification tasks



Inference

- Start : $h(0) = 0$
- For $t = 1$ to T DO : $h(t) = g(V \times h(t-1) + U \times x(t))$
- Predict : $y = g(W \times h(T))$
  $\Rightarrow$ The final state $h(T)$ resumes the whole input

Machine Translation



- Example of a translation model as a asynchronous Many to Many model
- The nature of language and of complex grammatical forms require to first "understand" the sentence, encoding it in a small dimensional hidden space, then to reconstruct the sentence in the target language.

Sequence autoencoders

Main idea

- Same architecture as the traduction model
- But where the first RNN that processes inputs is viewed as an encoder
- And the second RNN that successively produce all outputs is viewed as the decoder
- Same learning strategy as autoencoders: The desired output sequence is the input sequence
- This forces the model to learn to summarize the whole sequence in the last hidden state of the encoder

Depth in RNNs

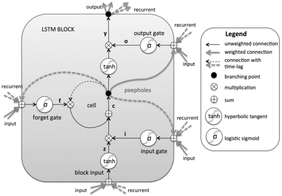
Two dimensions

- Stacked hidden layers as in traditional deep NNs : usual in many arhcitectures
- Long sequences $\rightarrow$ deep in time
- Both structural depths yield similar optimization problems (gradient flow)

New units for RNNs

- Motivation:
  - Optimization problems in Recurrent Neural Networks (gradient explosion / vanishing)
  - Difficulty to capture long term dependencies
- New types of hidden cells
  - Long Short Term Memory (LSTM) [Hochreichetr 98]
  - Gated Recurrent Unit (GRU) [Cho and al., 2014]

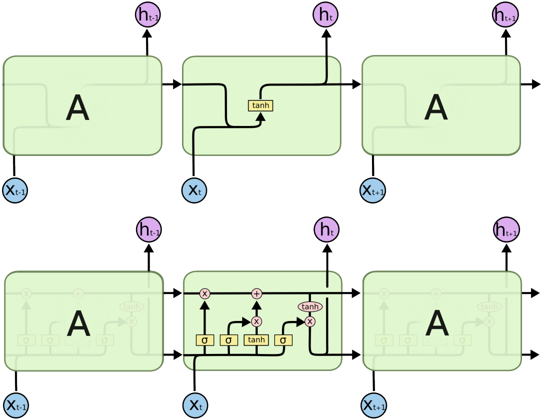
LSTM units



Motivation

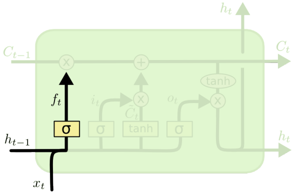
- Units that include few gates (*forget*, *input*, *output*) which allow to :
  - Stop capitilizing in the state the information about the past
  - Decide if it is worth using the information in the new input
- Depending on the input and on previous state
  - Reset the state, Update the state, Copy previous state
  - Ignore new input or fully use it to compute a new state

LSTM units

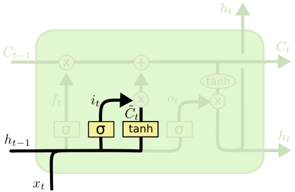


- LSTM layers may be stacked as well as standard RNN layers ($h_t = LSTM(x_t, h_{t-1}, c_{t-1})$ is input to the upper layer)

LSTM units



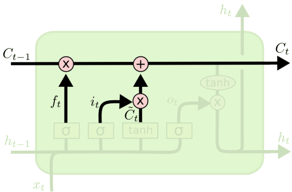
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



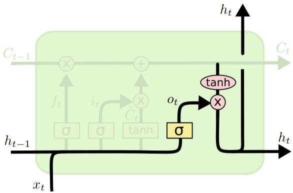
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- Forget gate f_t
- Input gate i_t
- Alternative cell state $\tilde{c}_t$

LSTM units



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- Cell state c_t
- Output o_t
- Hidden state to propagate h_t

LSTM units

Notations

- Cell state c_t
- Forget gate f_t
- Input gate i_t
- Output o_t
- Hidden state to propagate to upper layers h_t

How does it work ? in words...

- Recall formulas

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot tanh(c_t)$$

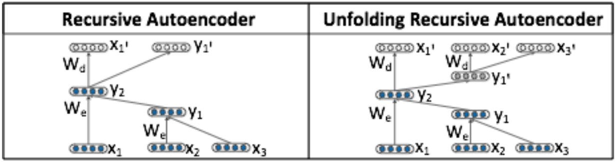
- Interpretation
  - If $f_t == 1$ and $i_t == 0$ use previous cell state
  - If $f_t == 0$ and $i_t == 1$ ignore previous cell sate
  - If $o_t == 1$ output ois set to cell sate
  - If $o_t == 0$ output is set to 0

Outline

Dealing with structured data

Principle

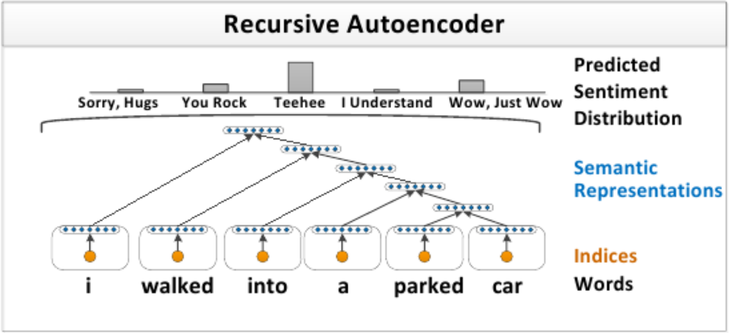
- Allows dealing with other structured data such as trees
- The model may still be unfolded and gradient may easily be computed



Dealing with structured data

Principle

- Allows dealing with other structured data such as trees
- The model may still be unfolded and gradient may easily be computed



Outline

1. RNNs
2. Recursive models
3. Embeddings
   - Embedding layer
   - Mikolov at google
   - Embeddings and transfer

Embedding layer

Motivation : Transformation layer for discrete/categorical inputs

- Example : a Word in a Dictionary (Natural Language Processing tasks)
- Embedding : distributed representation. Not a new idea (LSA, LDA)

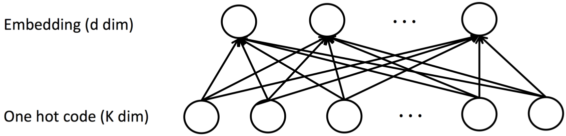
Main interests

- When the cardinality of the input is (very) large (e.g. NLP tasks) to allow accurate estimation from tractable corpus
- When one wants to infer some continuous representations of the input values to get insight on similarities between them

Embedding layer: Implementation

Look up table

- One entry for each of the possible values $\{v_1, \ldots, v_K\}$ (e.g.words in a dictionary)
- Each value is represented as a $d-$dimensional vector (d is the size of the embedding)
- Represented as a layer with a weight matrix $(K \times d)$



V_1 = [1 0 0 … 0 0]

V_2 = [0 1 0 … 0 0]

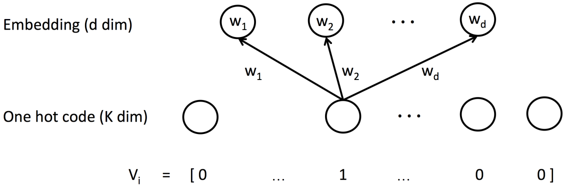
…

V_K = [0 0 0 … 0 1]

Embedding layer: Implementation

Look up table

- One entry for each of the possible values $\{v_1, ..., v_K\}$ (e.g.words in a dictionary)
- Each value is represented as a $d-$dimensional vector (d is the size of the embedding)
- Represented as a layer with a weight matrix ($K \times d$)

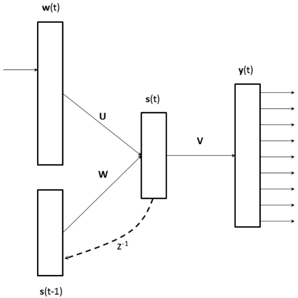


Word2vec [Mikolov, 2013]

Motivation pour les représentations continues (dans les modèles de langage)

- Modèle de langages type Ngrams : Pas de notion de similarité entre mots / nécessité de corpus de taille gigantesque pour une bonne estimation (qui n'existent pas)
- Représentation distribuée de mots → dépasse les limitations des Ngrams car partage de l'information entre mots :
  - Réponse similaire du système pour une entrée similaire
  - Taille de corpus nécessaire gigantesque mais suffisante pour apprendre ce que l'on veut

Modèle de langage type RN récurrent [Mikolov 2013]



- $W(t)$: Codage 1 parmi N
- $y(t)$: Distrib de proba sur Vocab

$$s(t) = f(Uw(t) + Ws(t-1)$$

$$y(t) = g(Vs(t-1))$$

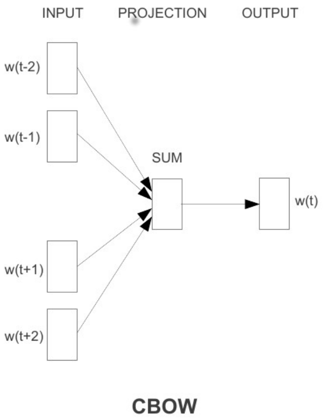
- f : sigmoide ; g : softmax

$\Rightarrow$ Les représentations des mots sont les colonnes de U

$\Rightarrow$ Capacité intéressante des représentations de mots

Une relation particulière entre deux mots (syntaxique pluriel, féminin, ou sémantique) correspond à un déplacement constant dans l'espace des représentations

Architecture CBOW (Continuous Bag Of Words) [Mikolov et al., 2013]



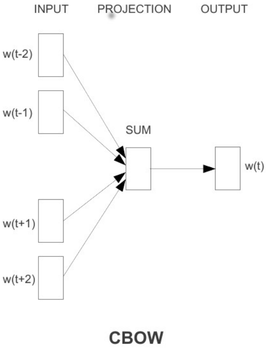
- Ultimate goal : estimate kind of *absolute* low dimensional distributed representations of words (including semantic, syntactic information...)
- Interests
  - Much smaller objects
  - Shared parameters between similar entries
- Maximum likekihood learning on very large corpus of texts

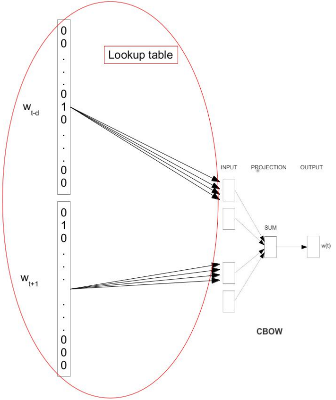
$$P(w_t|w_{t'\neq t}) \propto exp((\sum_{d\neq 0} w_{t+d})^T w_t))$$

- Learned embeddings may be used as feature vectors in many applications
- Recent alternatives, e.g. Gloves [Pennington and al., 2014]

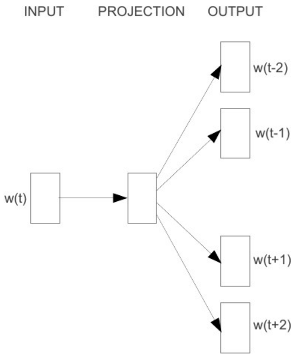
Architecture CBOW (Continuous Bag Of Words) [Mikolov et al., 2013]

- The overall architecture may be put in the shape of a deep NN





Architecture SkipGram ?



Skip-gram

- Utilise une representation pour chaque mot en entrée et idem en sortie
- Modèle

$$P(w_{t+d}|w_t) \propto exp((w_{t+d})^T w_t))$$

Word2vec

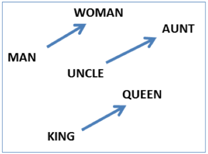
Caractéristiques de l'approche

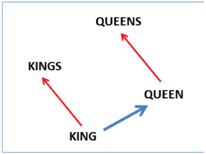
- Un modèle très simple appris avec **beaucoup** de données
- $\neq$ modèles continus de langage : pas de couche cachée
- Appris par MV sur textes standards indépendamment d'une tâche
  $\Rightarrow$ volonté de trouver une représentation "universelle"

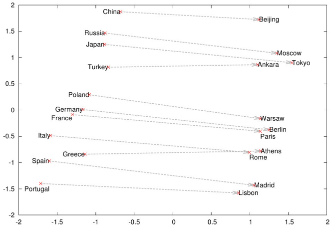
A particular interesting effect: compositionality

Idea

- $Emb('King') + Emb('Woman') - Emb('Man') \approx Emb('Queen')$
- It is an observed phenomenon which is not actually favored by the model design the learning criterion

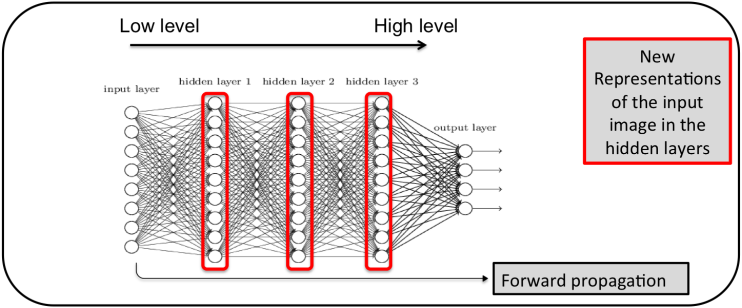






Extension of the embedding idea

More generally one call embedding a new representation space for any input data



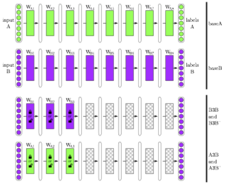
Genericity of representations [Yozinski and al., 2014]

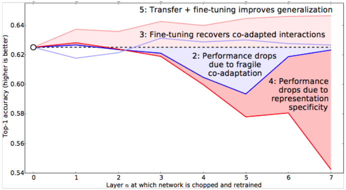
Experiments on two similar tasks

- Two DNN : Green one learned on Task A - Blue on Task B
- Reuse DNNA for Task B (and vice versa)
- Study the effect of reusing a DNN up to layer number i ...

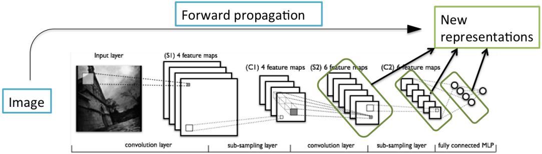
Main results

- Better to reuse DNNA and fine tune on Task B
- Lower layers learn transferable features while higher don't





Extension of the embedding idea for vision tasks



Main interest

- Many very deep architectures have been proposed by major actors (Google, Microsoft, Facebook...)
  - Using huge training corpora
  - Using huge computing resources
  - Architecture and Weights are often made publicly available
- It is better to use such models for computing high features from which one may design a classifier
  - With fine tuning (of upper layers) if enough training data are available on the target task
  - As a preprocessing if not