An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation

Hugo Larochelle, **Dumitru Erhan**, Aaron Courville,
James Bergstra, and Yoshua Bengio
Université de Montréal

August 9, 2007 - CIAR Summer School

Introduction

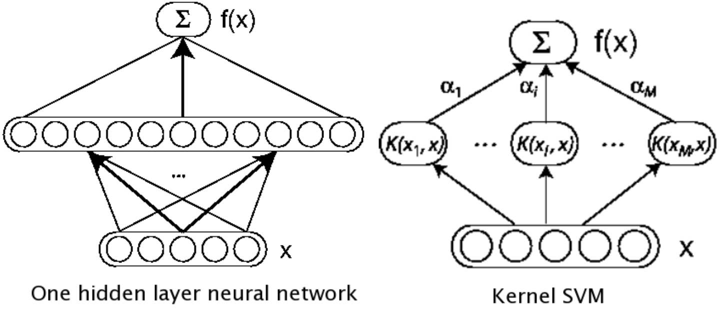
- Recently, models relying on deep architectures have been proposed (Deep Belief Networks)
- Their performance compare favorably to state of art models such as Support Vector Machines
- They have been tested on relatively few and simple problems
- We propose to evaluate them on problems with many factors of variation

Problems with Many Factors of Variation

- We focus here on problems were the input distribution has the following structure
$$p(x) = \sum_{\phi_1,\ldots,\phi_m} p(x|\phi_1,\ldots,\phi_m)p(\phi_1,\ldots,\phi_m)$$
where $p(\phi_1,\ldots,\phi_m)$ is high for (exponentially) many combinations of values of the factors of variation ϕ_i
- Problems with such a structure :
  - digit recognition (vision) : $\phi_i \in \{$rotation angle, scaling, background, etc.$\}$
  - document classification (NLP) : $\phi_i \in \{$topics, style, etc.$\}$
  - speech recognition (signal processing) : $\phi_i \in \{$speaker gender, background noise, environment echo, etc.$\}$
- Hard problems because the input space is densely populated, i.e. many regions of input space with potentially different desired output (target)
- We will focus on vision problems
- We want to avoid hand-engineered solutions to these problems

Shallow Architecture Models

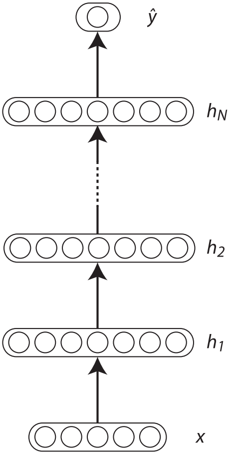
- A *shallow model* is a model with very few layers of computational units :



- To approximate a complex function, such models will need large (exponential) number of computational units (cf Yoshua's talk on Tuesday)

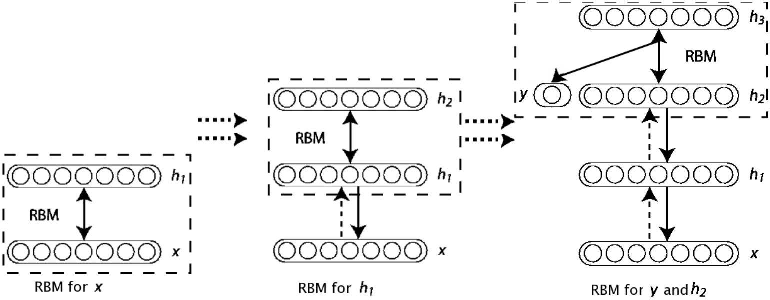
Deep Architecture Models

- A *deep architecture model* is such that its output is the result of many consecutive compositions of computational units
- Many layers potentially yield highly complex functions with a limited number of parameters
- The d dimensional parity function modeled with
  - $O(d2^d)$ parameters with Gaussian SVM
  - $O(d^2)$ parameters with a $O(\log_2 d)$ hidden layer neural network



Learning Deep Architecture Models (DBN)

- (Hinton et al., 2006) introduced the Deep Belief Network (DBN), a deep probabilistic neural network
- The training procedure is first **layer-wise greedy** and **unsupervised** (initialization)

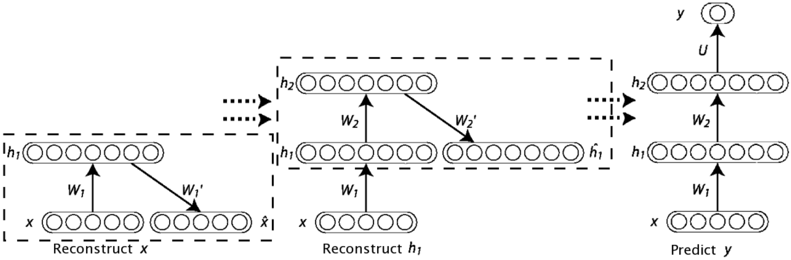


- Then the output of the model is **fine-tuned** on the supervised data

$$\arg\min_{\theta} -\frac{1}{n}\sum_{i=1}^{n} \log \hat{p}(y_i|x_i,\theta)$$

Learning Deep Architecture Models (SAA)

- Instead of stacking RBMs, we can have Stacked Autoassociators (SAA)
- The training procedure is first **layer-wise greedy** and **unsupervised** (initialization)



- Then the output of the model is **fine-tuned** on the supervised data

$$\arg\min_{\theta} -\frac{1}{n}\sum_{i=1}^{n} \log \hat{p}(y_i | x_i, \theta)$$

Greedy Module : Autoassociators

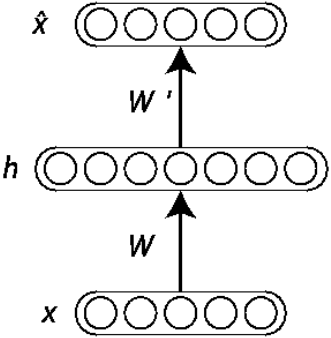
- Autoassociator is a neural network trained to reconstruct its input

$$\hat{x} = sigm(c + W' sigm(b + Wx))$$

- The reconstruction error is

$$R(x, \hat{x}) = -\sum_i x^i \log \hat{x}^i + (1 - x^i) \log(1 - \hat{x}^i)$$

- The neural network is trained using a gradient descent algorithm



Experimental setup

- We report results for the following models :
  - Support Vector Machine classifiers with polynomial (**SVM**$_{poly}$) and Gaussian (**SVM**$_{rbf}$) kernels
  - One hidden layer neural network (NNet)
  - Deep Belief Network (DBN-1 and DBN-3) and Stacked Autoassociator (SAA-3) with one or three hidden layers
- The validation set was used to do model selection and early stopping
- **SVM**$_{poly}$ and **SVM**$_{rbf}$ were retrained on the union of the training and validation set

Dataset Characteristics Summary

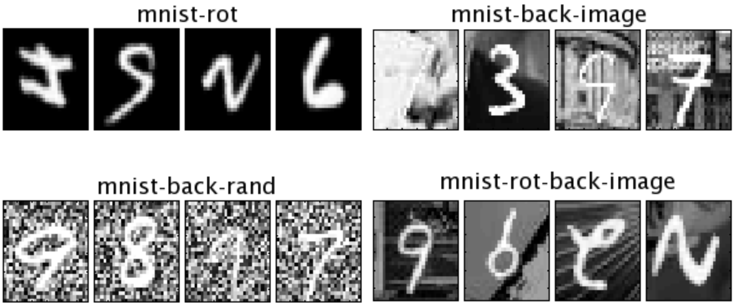
- Classification datasets on 28×28 pixel images
- All datasets have a training, validation and test split
- Training set size varies from 1000 to 10000 samples
- Validation set size varies from 200 to 2000 samples
- All datasets have a test set of size 50000.

Variations on Digit Recognition

- MNIST dataset with additional factors of variation
  1. Pick sample $(x, y) \in \mathcal{X}$ from the digit recognition dataset ;
  2. Create a perturbed version x^* of x according to some factors of variation ;
  3. Add (x^*, y) to a new dataset $\mathcal{X}^*$;
  4. Go back to 1 until enough samples are generated.
- We generated the following datasets :

Dataset	Additional factors of variation
mnist-rot	rotation angle between 0 and 2π radians
mnist-back-rand	random background pixels between 0 and 255
mnist-back-image	random patch from 20 black and white images
mnist-rot-back-image	factors of *mnist-rot* and *mnist-back-image*

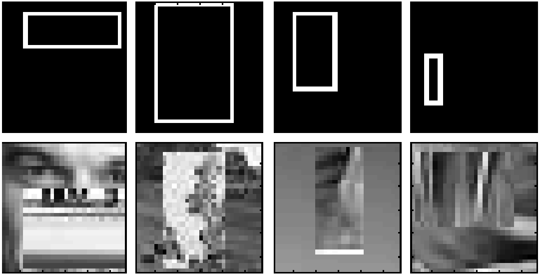
Variations on Digit Recognition (samples)



Dataset	SVM_{rbf}	SVM_{poly}	NNet	DBN-1	SAA-3	DBN-3
mnist-basic	**3.03**	3.69	4.69	3.94	3.46	**3.11**
mnist-rot	**10.38**	13.61	17.62	12.11	11.43	12.30
mnist-back-rand	14.58	16.62	20.04	9.80	11.28	**6.73**
mnist-back-image	22.61	24.01	27.41	**16.15**	23.00	**16.31**
mnist-rot-back-image	32.62	37.59	42.17	31.84	**24.09**	28.51

Discrimination between Tall and Wide Rectangles

- *rectangles* : the pixels corresponding to the border of the rectangle has a value of 255, 0 otherwise

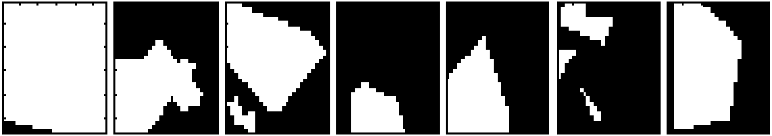


- *rectangles-image* : the border and inside of the rectangles correspond to an image patch. A background patch is also sampled

Dataset	**SVM$_{rbf}$**	**SVM$_{poly}$**	**NNet**	**DBN-1**	**SAA-3**	**DBN-3**
rectangles	**2.15**	**2.15**	7.16	4.71	**2.41**	2.60
rectangles-image	24.04	24.05	33.20	23.69	24.05	**22.50**

Recognition of Convex Sets

- *convex* contains images corresponding to convex and non convex sets of pixels



- The convex sets are intersections of random half-planes
- The non convex sets correspond to the union of a random number of convex sets, failing a convexity test

Dataset	SVM_{rbf}	SVM_{poly}	NNet	DBN-1	SAA-3	DBN-3
convex	19.13	19.82	32.25	19.92	**18.41**	**18.63**

Results Summary

Dataset	**SVM$_{rbf}$**	**SVM$_{poly}$**	**NNet**	**DBN-1**	**SAA-3**	**DBN-3**
mnist-basic	**3.03**	3.69	4.69	3.94	3.46	**3.11**
mnist-rot	**10.38**	13.61	17.62	12.11	11.43	12.30
mnist-back-rand	14.58	16.62	20.04	9.80	11.28	**6.73**
mnist-back-image	22.61	24.01	27.41	**16.15**	23.00	**16.31**
mnist-rot-back-image	32.62	37.59	42.17	31.84	**24.09**	28.51
rectangles	**2.15**	**2.15**	7.16	4.71	**2.41**	2.60
rectangles-image	24.04	24.05	33.20	23.69	24.05	**22.50**
convex	19.13	19.82	32.25	19.92	**18.41**	**18.63**

TAB.: Results on the benchmark for problems with factors of variation (in percentages). The best performance as well as those with overlapping confidence intervals are marked in bold.

Investigation of the “background effect”

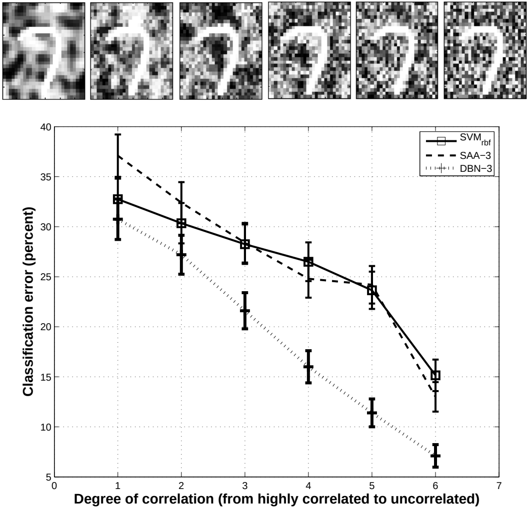


FIG.: Classification error of **SVM**$_{rbf}$, **SAA-3** and **DBN-3** on MNIST examples with progressively less pixel correlation in the background.

Conclusion

- In general, models with deep architectures either perform as well or outperform other models
- There are still challenges in scaling the current algorithms to problems with very complex input distribution
- Datasets and experimental details can be found on our public wiki page :

  `http://www.iro.umontreal.ca/~lisa/ptwiki/`

Future Work

- Address the “focus problem” of greedy layer-wise unsupervised training
- Develop learning algorithms that make better use of their capacity (NORB)
- Develop models and algorithms with fewer hyper-parameters
- Develop faster learning algorithms for these models

Kernel PCA

- Schoelkopf et al. suggested stacking up kPCAs (in ’98 !)
- They called that the “Multi-layered kernel machines”
- Why not ? Well, because. . .
  - Local kernels remain local, even if “stacked”
  - Big non-parametric “layers”
  - Biologically implausible (?)
  - No underlying generative model
- Why yes ?
  - Convex optimization of each layer
  - Few hyper-parameters
  - With simple—yet nonlinear—kernels, global (nonconvex) optimization
  - No underlying generative model :)

Multi-layered kernel machines framework

- The “algorithm” :
  - For each x, compute its projection $\hat{x}$ via kPCA
  - Use those projections as inputs to the next “layer” kPCA
  - Repeat until network is deep enough (3 layers, by our standards)
  - Put an SVM on top to do the supervised task
- Schoelkopf et al. did kPCA + linear SVM
- $N \times$ {polynomial, RBF} kernel PCA
- Followed by {linear, polynomial, RBF} SVM on top
- Number of layers = $N + 1$

Results

- Always better than vanilla SVM
- RBF kPCA followed by polynomial (d=2) SVM is the best model
- Optimal layer sizes are problem dependent : 512, 1024, 64, 128
- Adding another layer only hurts. . .

Dataset	2 − layer KM	SVM$_{rbf}$	DBN-1	SAA-3	DBN-3
mnist-basic	**2.48**	3.03	3.94	3.46	3.11
mnist-rot	**8.64**	10.38	12.11	11.43	12.30
mnist-back-rand	10.44	14.58	9.80	11.28	**6.73**
mnist-back-image	21.30	22.61	**16.15**	23.00	**16.31**

TAB.: Results on the benchmark for problems with factors of variation (in percentages). The best performance as well as those with overlapping confidence intervals are marked in bold.

Second round of conclusions and future work

- Deep architectures as a general concept
- Multi-layer kernel machines work pretty well ! Exercise 14.15 from "Learning with kernels" is solved !
- The "focus problem" is still there (kPCA is fully unsupervised)
- Stacking semi-semisupervised components (to "propagate" the target/label) seems sensible

Bye-bye

- Thank you!!

Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.