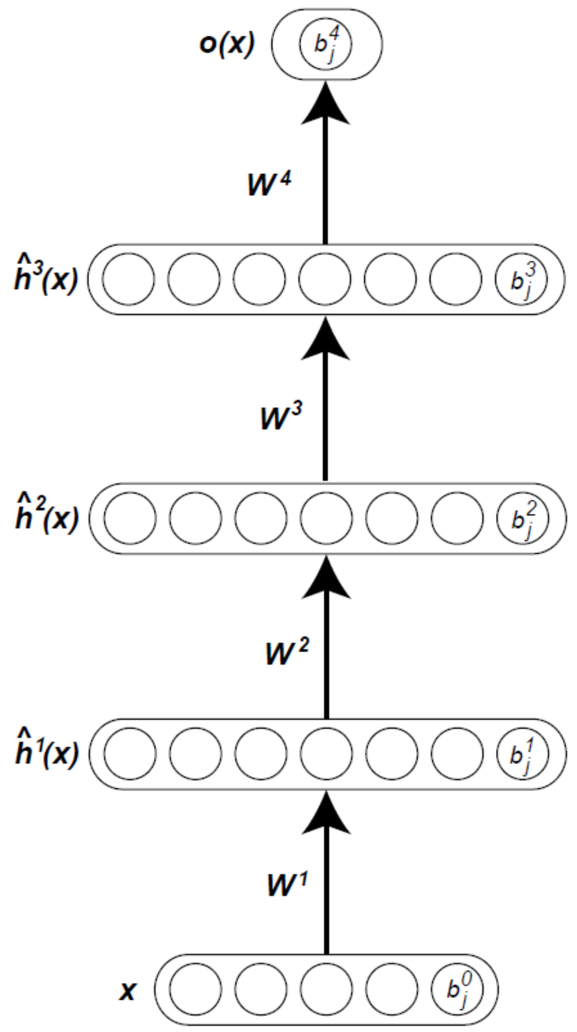
Exploring Strategies for Training Deep Neural Networks

Presenter: Hossein Pourmodheji

EECS 6412

Fall 20184



Motivation

- It is suggested that deep multi-layer neural networks architectures can be much more efficient than shallow architectures, in terms of computational elements and parameters required to represent some functions.
- Not on every problem, but whenever the task is complex enough, and there is enough data to capture that complexity.
- *Finding better learning algorithms for such deep networks could be beneficial.*

Training Challenges

- Training deep multi-layered neural networks is known to be hard.
- The standard learning strategy consists of randomly initializing the weights of the network and applying gradient descent using Backpropagation.
- It is known empirically to find poor solutions for networks with 3 or more hidden layers.

Challenge (I)

- Gradient descent can easily get stuck in poor local minima or plateaus of the non-convex training criterion.
  - With more layers, the number local minima or the width of plateaus increase.

Challenge (II)

- Gradient descent introduce low training error but very different generalization errors
- When gradient descent is able to find a (possibly local) good minimum in terms of training error, there are no guarantees that the associated parameter configuration will provide good generalization.
- The type of unsupervised initialization discussed here can help to select basins of attraction (for the supervised fine-tuning optimization phase) from which learning good solutions is easier both from the point of view of the training set and of a test set.

Exploring Two Training Algorithms

- Stacked Restricted Boltzmann Machine Network (SRBM)
  - It is proposed a greedy *layer-wise unsupervised learning* procedure <u>**relying on**</u> the training algorithm of *restricted Boltzmann machines* (RBM) to initialize the parameters of a *deep belief network* (DBN).

- Stacked Autoassociators Network (SAA)
  - SRBM was followed by the proposal of another greedy *layer-wise* procedure, <u>**relying on**</u> the usage of *autoassociator networks*.

Paper Main Objectives

- Studying these two algorithms empirically to better understand their success, in the context of the optimization problem.
- Showing that the greedy *layer-wise unsupervised training* strategy helps the optimization by *initializing weights* in a region **near a good local minimum**.
- Showing that this strategy implicitly acts as a sort of *regularization* that brings better generalization and encourages internal distributed representations that are high-level abstractions of the input.

Idea Behind these two Algorithm

- An approach that has been explored with some success in the past is based on constructively adding layers. (*layer-wise*)

- Each layer in a multi-layer neural network can be seen as a representation of the input.

- Good internal representation of the data should disentangle the factors of variation that inherently explain the structure of the distribution.

Supervised vs. Unsupervised

- When such a representation is going to be used for unsupervised learning, we would like it to preserve information about the input.

- When a representation is going to be used in a supervised prediction or classification task, we would like it to be such that there exists a “simple” (i.e., somehow easy to learn) mapping from the representation to a good prediction.

Supervised Representation Issues

- To constructively build such a representation, it has been proposed to use a *supervised* criterion at each stage.
- However, the use of a supervised criterion at each stage may be too greedy and does not yield as good generalization as using an unsupervised criterion.
- Aspects of the input may be ignored in a representation tuned to be immediately useful (with a linear classifier) but these aspects might turn out to be important when more layers are available.

Combining Supervised & Unsupervised

- Combining unsupervised (e.g., learning about $p(x)$) and supervised components (e.g., learning about $p(y \mid x)$) can be helpful when both functions $p(x)$ and $p(y \mid x)$ share some structure.

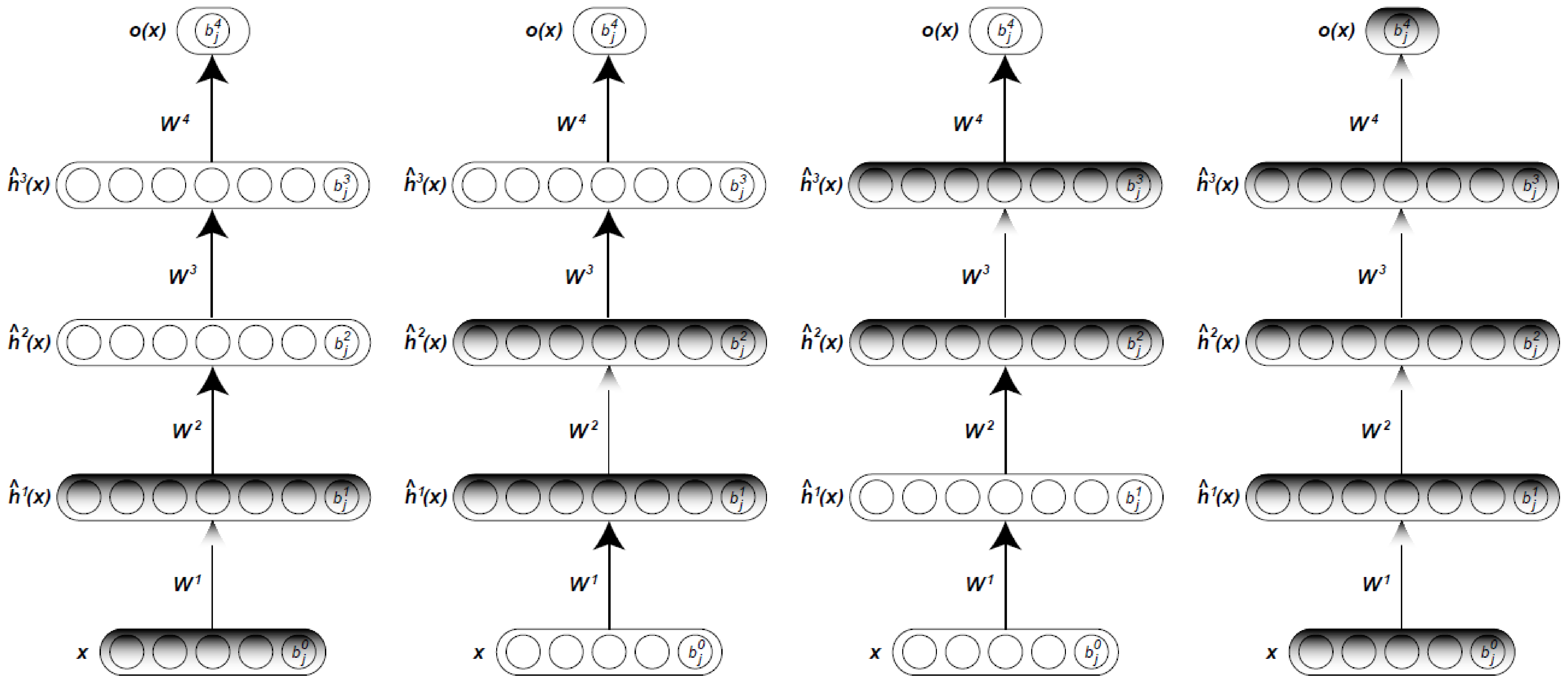
Three Principles for Training Deep NN

1) Pre-training one layer at a time in a greedy way.

2) Using unsupervised learning at each layer in a way that preserves information from the input and disentangles factors of variation.

3) Fine-tuning the whole network with respect to the ultimate criterion of interest.

Unsupervised greedy layer-wise training procedure

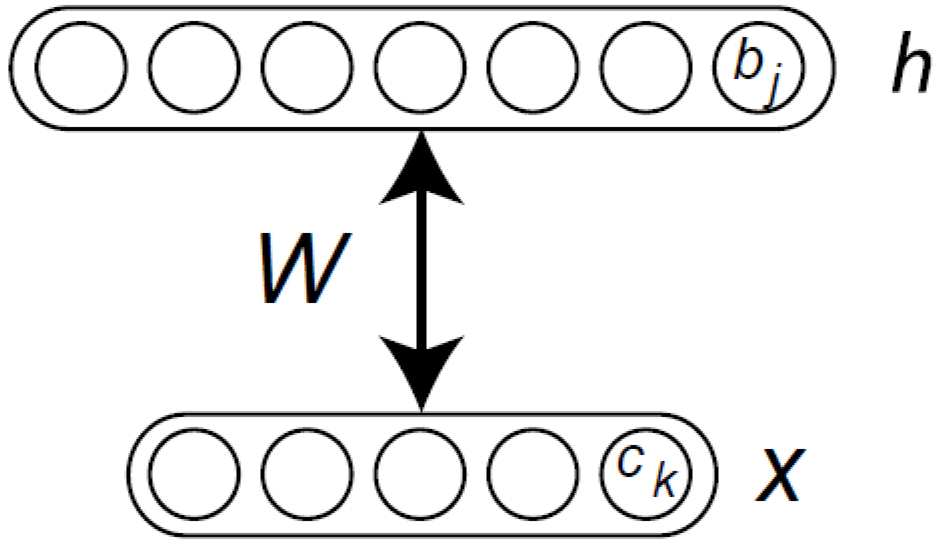


SRBM

- Stacking Restricted Boltzmann Machine (SRBM) provides a good initialization strategy for the weights of a deep artificial neural network.
- SRBM: training upper RBMs on the distribution of activities computed by lower RBMs
- This approach is extended to
  - stacked autoassociators network (SAA)
  - non-linear autoencoders
  - deep convolutional neural network

RBM

- It is a generative model that uses a layer of binary variables to explain its input data.
- Hinton (2006) argues that this representation can be improved by giving it as input to another RBM, whose posterior over its hidden layer will then provide a more complex representation of the input.
- Contrastive Divergence algorithm (CD-k) is used to train an RBM



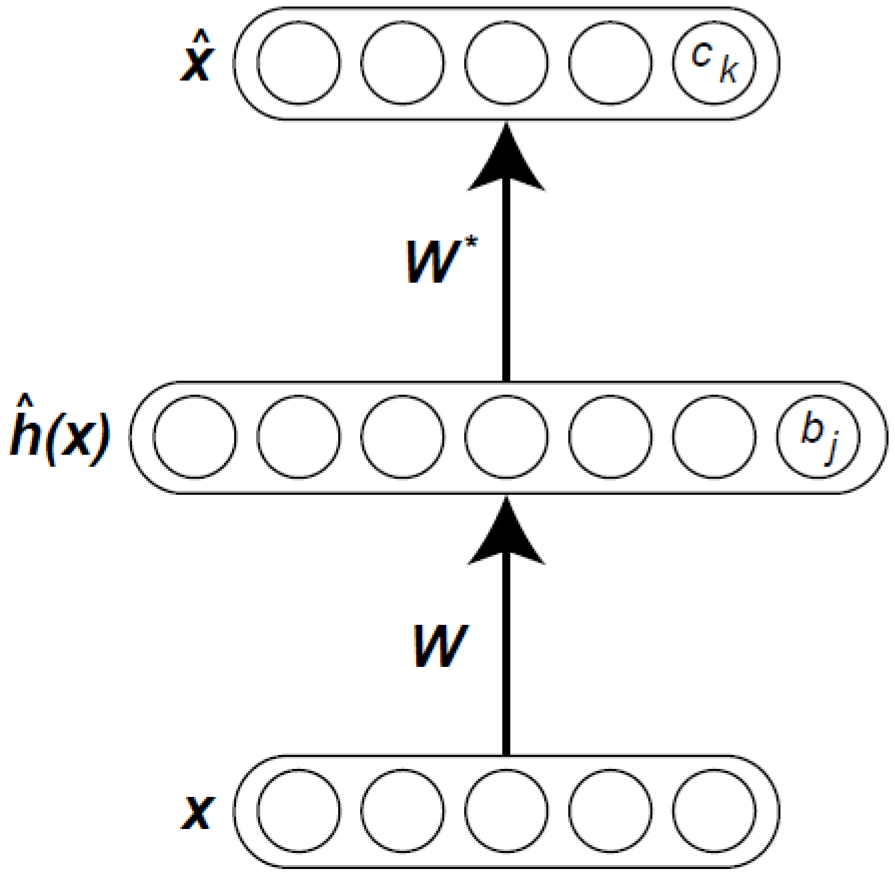
SAA

- There are other non-linear, unsupervised learning models than RBM that, when stacked, are able to improve the learned representation at the last layer added.
- An example of such a non-linear unsupervised learning model is the autoassociator or autoencoder network.
- Autoassociators are neural networks that are trained to compute a representation of the input from which it can be reconstructed with as much accuracy as possible.

SAA (Cont’s)

$$\widehat{x}_k = g(\widehat{a}_k) \text{ where } \widehat{a}_k = c_k + \sum_j W^*_{jk} \widehat{h}_j(\mathbf{x})$$

$$\widehat{h}_j(\mathbf{x}) = f(a_j) \text{ where } a_j(\mathbf{x}) = b_j + \sum_k W_{jk} x_k$$



SAA Challenge

- Some care must be taken so that the network does not learn a trivial identity function, that is, finds weights that simply “copy” the whole input vector in the hidden layer and then copy it again at the output.
- For example, a network with small weights W_{jk} between the input and hidden layers and large weights W_{jk}^{*} between the hidden and output layers could encode such an uninteresting identity function.
- An easy way to avoid such a pathological behavior in the case of continuous inputs is to set the weight matrices $\mathbf{W}^{T}$ and $\mathbf{W}$ to be the same.

Experiments

- SGD for both layer-wise unsupervised learning and global supervised fine-tuning.
- Data sets were separated in disjoint training, validation and testing subsets.
- All experiments correspond to classification problems.
- The experiments are based on the MNIST data set and variants of this problem where the input distribution has been made more complex by inserting additional factors of variations, such as rotations and background images

Validating the Unsupervised Layer-Wise Strategy

- Evaluating the advantages brought by the *unsupervised layer-wise* strategy by answering the following two questions:

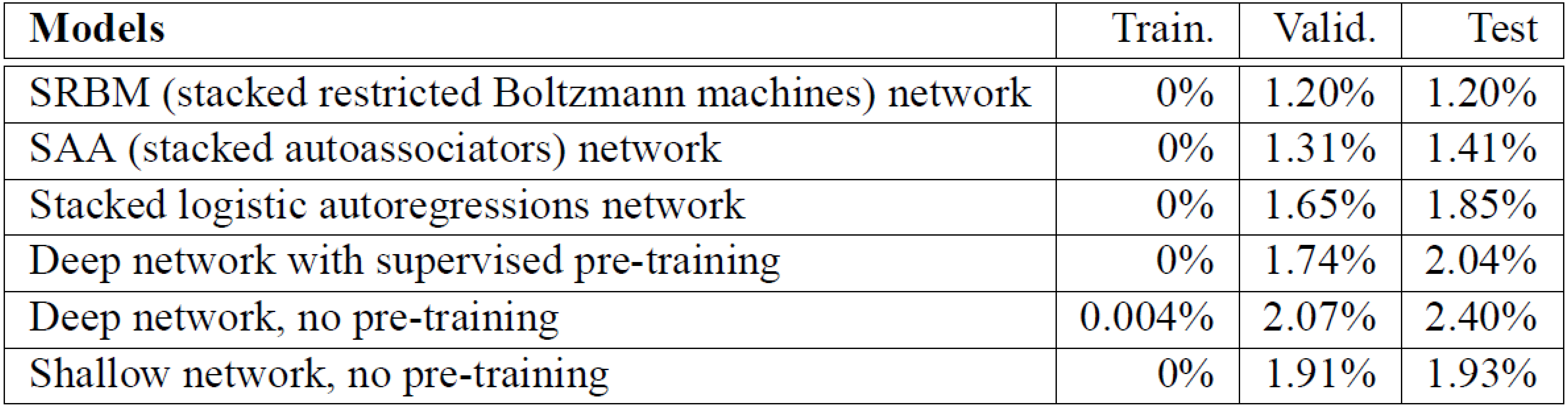
1. To what extent does initializing greedily the parameters of the different layers help?
2. How important is unsupervised learning for this procedure?

Comparison

- The two learning algorithms for deep networks are compared with the following algorithms:
  - Deep network without pre-training (1st question)
  - Deep network with supervised pre-training (2nd question)
  - Stacked logistic Autoregression network (2nd question)

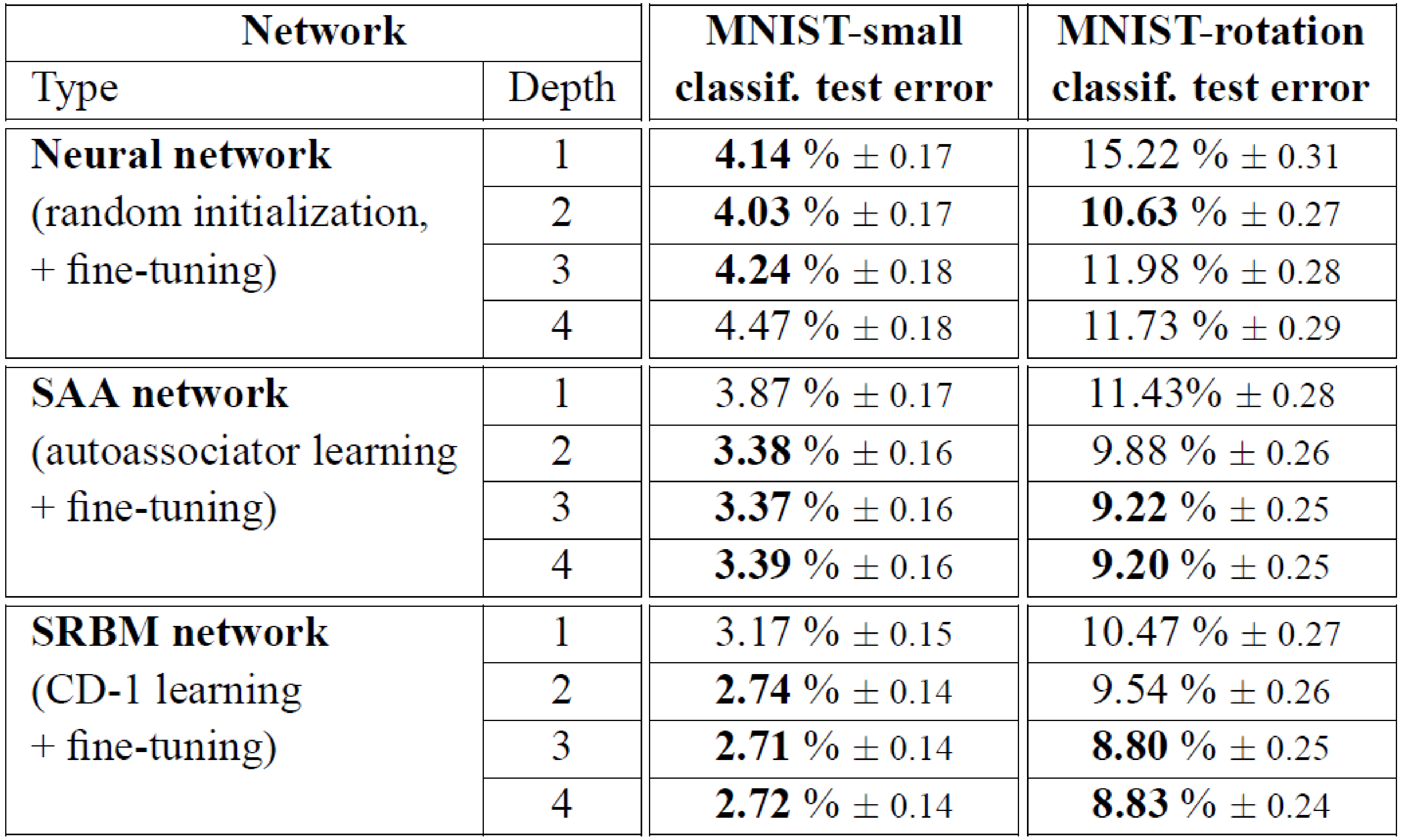
Results

- The *unsupervised layer-wise pre-training* improves generalization.



Models	Train.	Valid.	Test
SRBM (stacked restricted Boltzmann machines) network	0%	1.20%	1.20%
SAA (stacked autoassociators) network	0%	1.31%	1.41%
Stacked logistic autoregressions network	0%	1.65%	1.85%
Deep network with supervised pre-training	0%	1.74%	2.04%
Deep network, no pre-training	0.004%	2.07%	2.40%
Shallow network, no pre-training	0%	1.91%	1.93%

Network Depth



Network		MNIST-small	MNIST-rotation
Type	Depth	classif. test error	classif. test error
Neural network (random initialization, + fine-tuning)	1	**4.14** % ± 0.17	15.22 % ± 0.31
	2	**4.03** % ± 0.17	**10.63** % ± 0.27
	3	**4.24** % ± 0.18	11.98 % ± 0.28
	4	4.47 % ± 0.18	11.73 % ± 0.29
SAA network (autoassociator learning + fine-tuning)	1	3.87 % ± 0.17	11.43% ± 0.28
	2	**3.38** % ± 0.16	9.88 % ± 0.26
	3	**3.37** % ± 0.16	**9.22** % ± 0.25
	4	**3.39** % ± 0.16	**9.20** % ± 0.25
SRBM network (CD-1 learning + fine-tuning)	1	3.17 % ± 0.15	10.47 % ± 0.27
	2	**2.74** % ± 0.14	9.54 % ± 0.26
	3	**2.71** % ± 0.14	**8.80** % ± 0.25
	4	**2.72** % ± 0.14	**8.83** % ± 0.24

Network Depth Effects on SRBM

- ➢ Consider a hypothetical deep network where the top-level stacked RBM has learned a representation made of units that are mostly independent.
- ➢ An additional RBM stacked on this representation would have no statistical structure to learn.
- ➢ This would initialize the weights of that new RBM to zero, which is particularly troublesome as the representation at this level would then contain no information about the input.
- ➢ There is indeed an optimal number of hidden layers for the deep networks, and that this optimum tends to be larger when unsupervised greedy layer-wise learning is used.

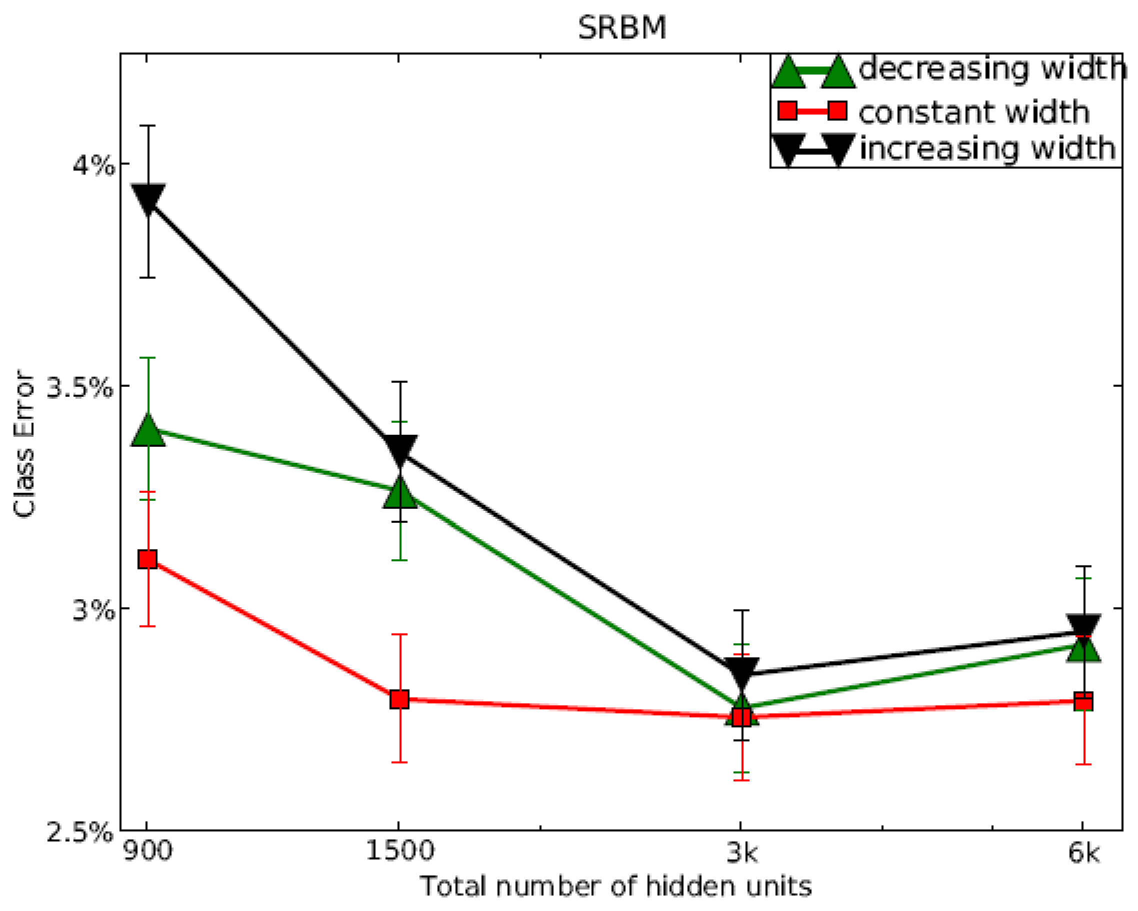
Type of Network Architecture

- The number of hidden layers of a deep network has already been chosen and good sizes of the different layers must be found.
- The space of such possible choices is exponential in the number of layers

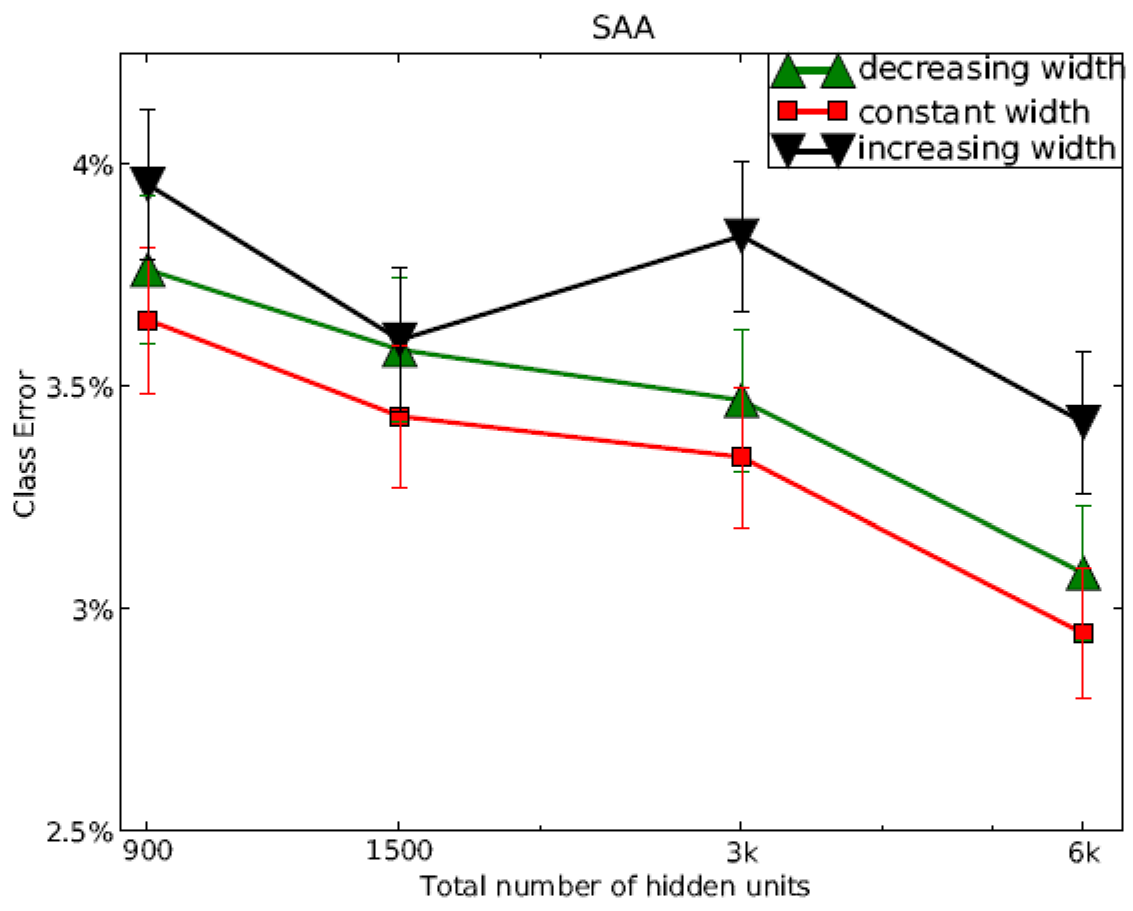
Type of Network Architecture (Cont'd)

- Three general cases, as the layer index increases, their sizes either
  - increases (doubles)
  - decreases (halves)
  - does not change
- The architecture that most often is among the best performing ones across the different sizes of network is the one with equal sizes of hidden layers.

MNIST-small

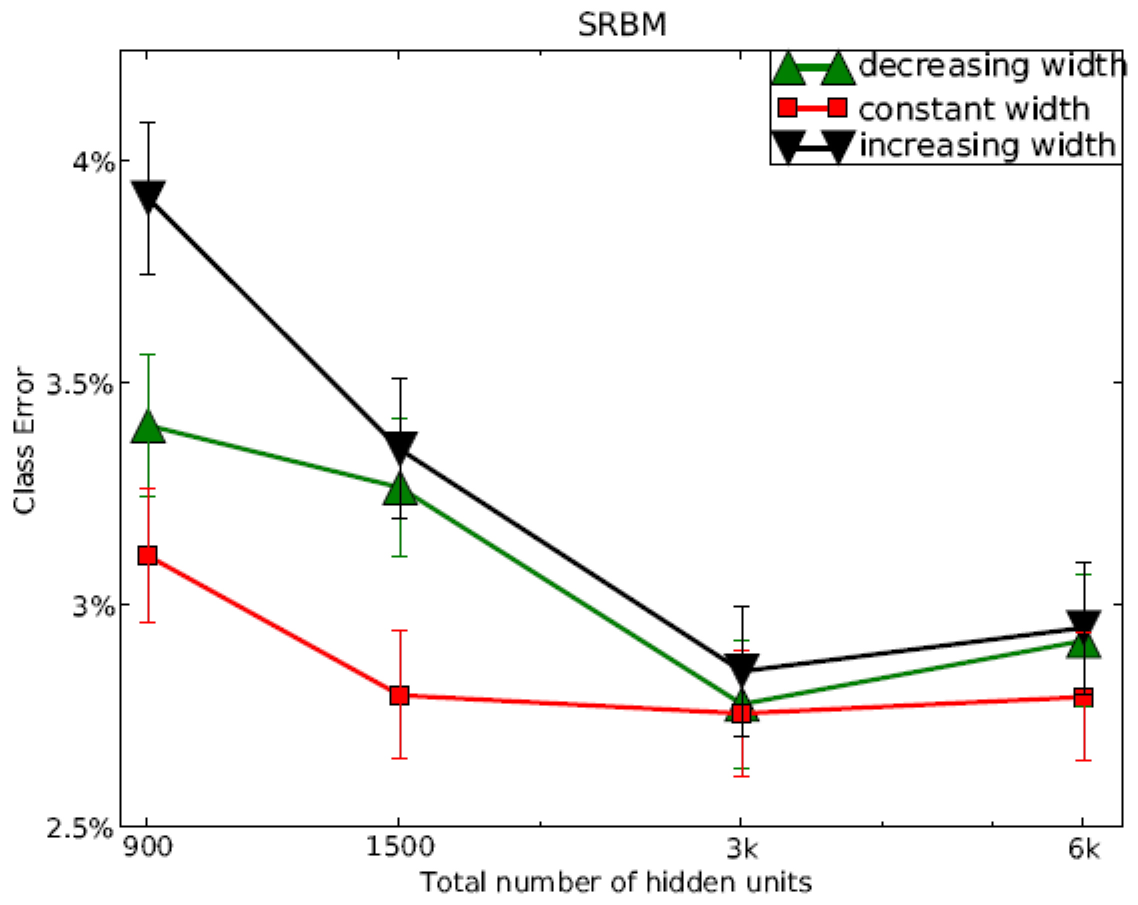


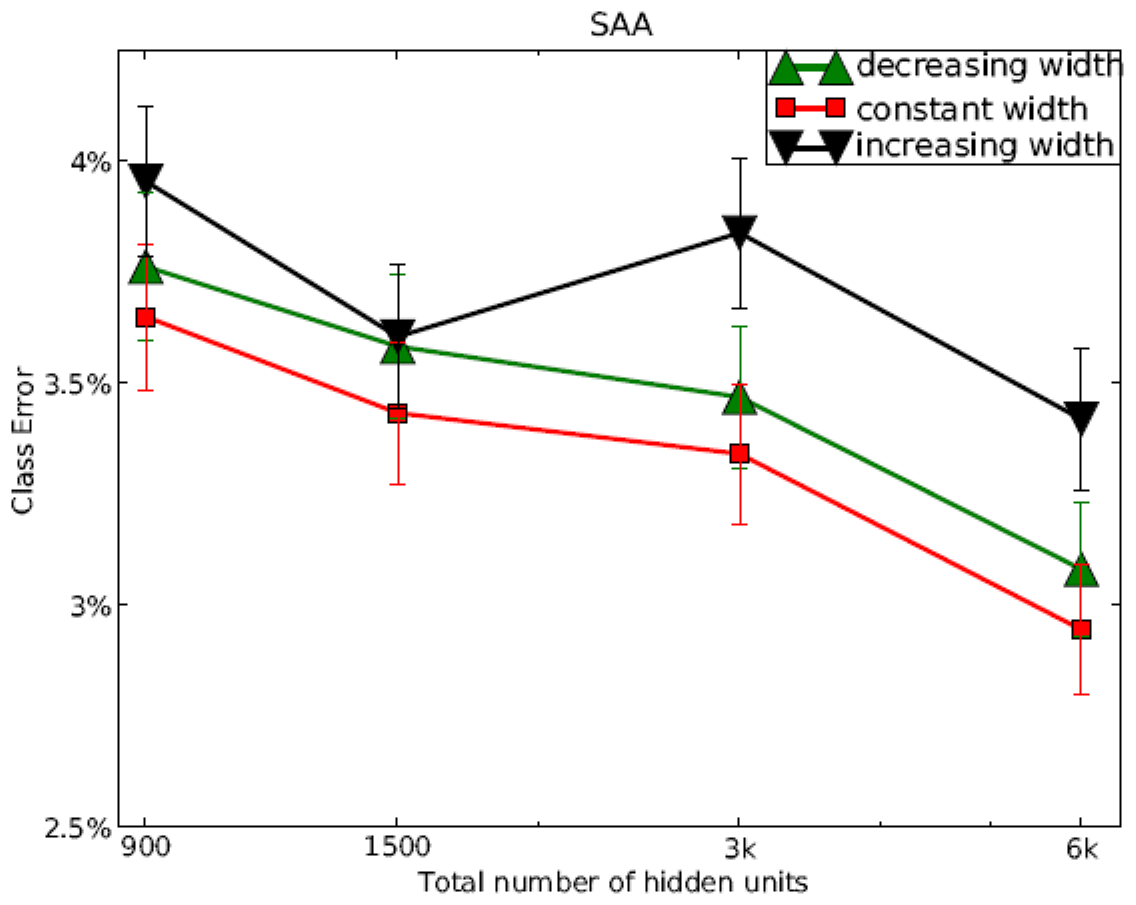
(a) SRBM network.



(b) SAA network.

MNIST-rotation





Generating vs Encoding

- The RBM is based on the learning algorithm of a *generative model*, which is trained to be able to generate data similar to those found in the training set.
- The autoassociator is based on the learning algorithm of an *encoding model* which tries to learn a new representation or code from which the input can be reconstructed without too much loss of information.
- *It is not clear which of the two approaches (generating or encoding) is the most appropriate.*

Generating

- It is possible that the problem it is trying to solve is harder than it needs to be, since ultimately we are only interested in coming up with good representations or features of the input.

- For instance, if one is interested in finding appropriate clusters in a very high dimensional space, using a mixture of Gaussians with full covariance matrix can quickly become too computationally intensive, whereas using the simple k-means algorithm might do a good enough job.

Encoding

- As for encoding models, they do not require to be interpretable as a generative model and they can be more flexible because any parametric or non-parametric form can be chosen for the encoder and decoder, as long as they are differentiable.

Conclusion

- Discussed three principles for training deep neural networks:
  - Pre-training one layer at a time in a greedy way
  - Using unsupervised learning at each layer in a way that preserves information from the input and disentangles factors of variation
  - Fine-tuning the whole network with respect to the ultimate criterion of interest
- Experimental evidence supports the claim that they are key ingredients for reaching good results

Conclusion (Cont’d)

- Unsupervised procedure helps the optimization of the deep architecture, while initializing the parameters in a region near which a good solution of the supervised task can be found.
- There are cases where greater depth clearly helps, but too much depth could be slightly detrimental.