Unsupervised Deep Learning

Alberto Testolin and Marco Zorzi
Department of General Psychology, University of Padova (Italy)

Workshop on Contemporary Deep Neural Network Models
COGSCI 2016 - Philadelphia, 10 August 2016

http://ccnl.psy.unipd.it/research/deeplearning

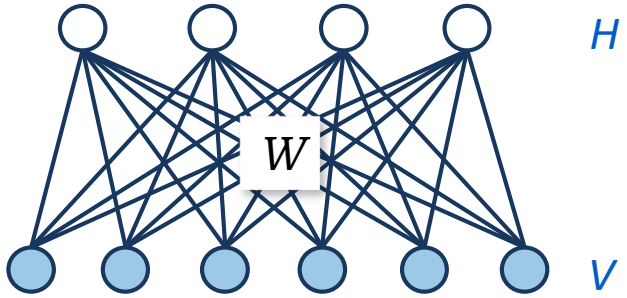
Aims of the breakout session

- Quick recap of unsupervised deep learning
  - Building blocks: Restricted Boltzmann Machines
  - Hierarchical generative models: Deep Belief Networks

- Hands-on tutorial
  - Learning a hierarchical generative model of handwritten digits (MNIST dataset): efficient implementation on GPUs
  - Analyzing the model:
    - Plotting receptive fields of the hidden neurons
    - Reading-out internal representations (supervised linear classifier)

Restricted Boltzmann Machines

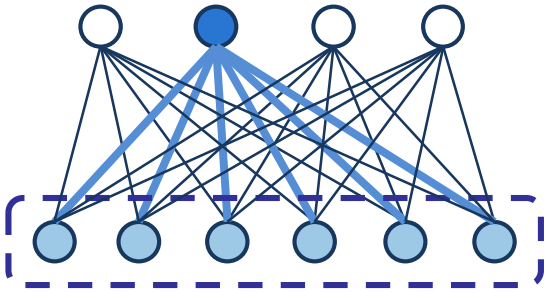
- Bipartite, fully-connected graphs (no intra-layer connections)
- Joint probability distribution defined by an ***energy function "E"***
- Energy depends on the strength of connections (model parameters):



$$P(v,h) = \frac{1}{Z}\prod_{i=1}^{N} \emptyset_i(v_k h_k) = \frac{e^{-E(v,h)}}{Z}$$

$$E(v,h) = -\,b^T v - c^T h - h^T\, W\, v$$

- The topology of the graph encodes conditional (in)dependences:

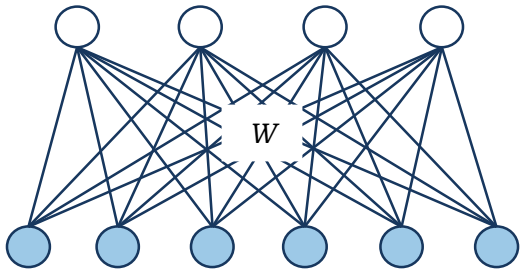


$$P(h|v) = \prod_i P(h_i|v)$$

$$P(v|h) = \prod_i P(v_i|h)$$

Contrastive Divergence

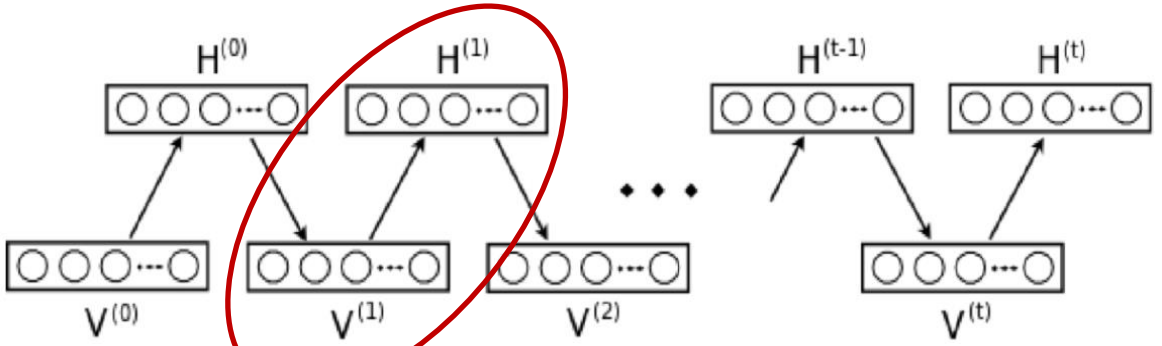
- Simple maximum-likelihood approach to learn RBMs
- Basic idea: minimize the discrepancy between the empirical data distribution (training set) and the model distribution (patterns generated by the network)
- Producing model's expectations is computationally too demanding, so we approximate this term by using "distortions" of the data produced by the model



$$\Delta W = \alpha(v^+h^+ - v^-h^-)$$

positive phase
(data driven)

negative phase
(model driven)



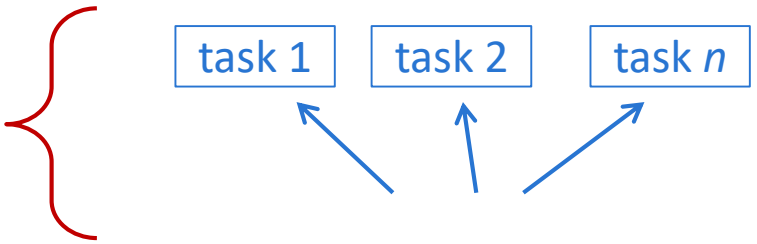
positive phase
(data driven)

In practice, CD-1 already
works extremely well!

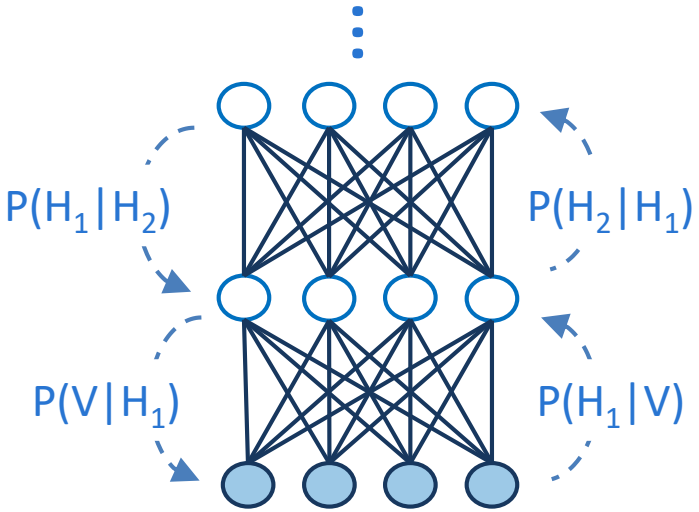
negative phase
(model driven, but
constrained by the data)

Deep Belief Networks

abstract representations can then be easily *read-out* by linear classifiers!



- We can learn more complex generative models by stacking together many RBMs
  - Greedy, layer-wise learning
  - Build multiple levels of representation (hypotheses over hypotheses)
  - Unsupervised learning of abstract features
  - Can simulate top-down effects

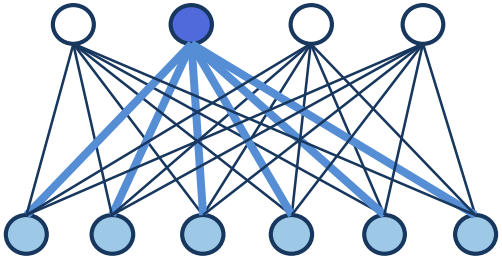


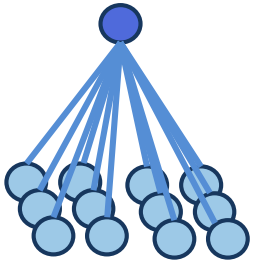
NB: each layer performs a ***non-linear*** projection of the input data!

Hinton & Salakhutdinov (2006), *Science*

Example: MNIST dataset

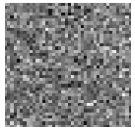
Receptive fields of hidden neurons:



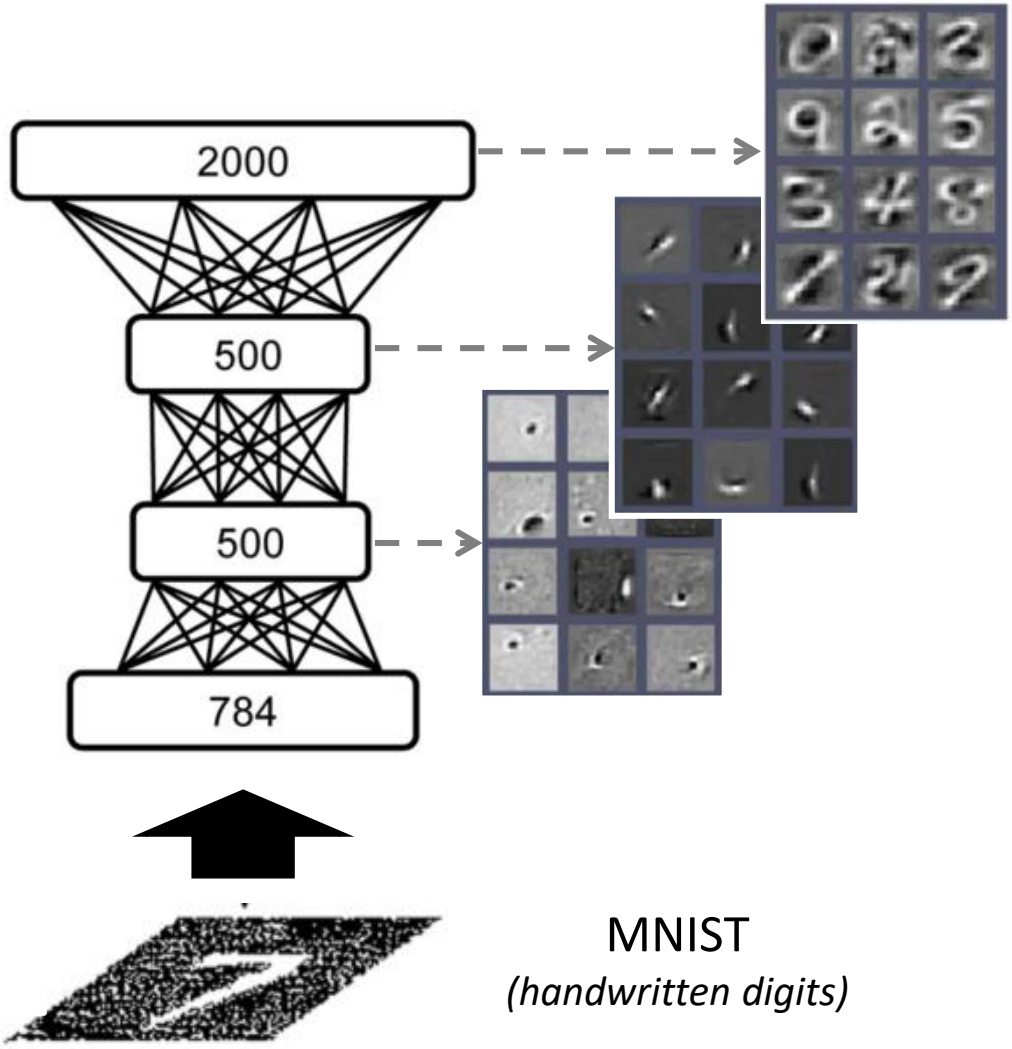


Connection strength:

Before learning (random)



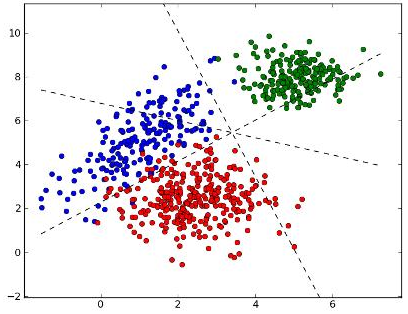
After learning (location specific)



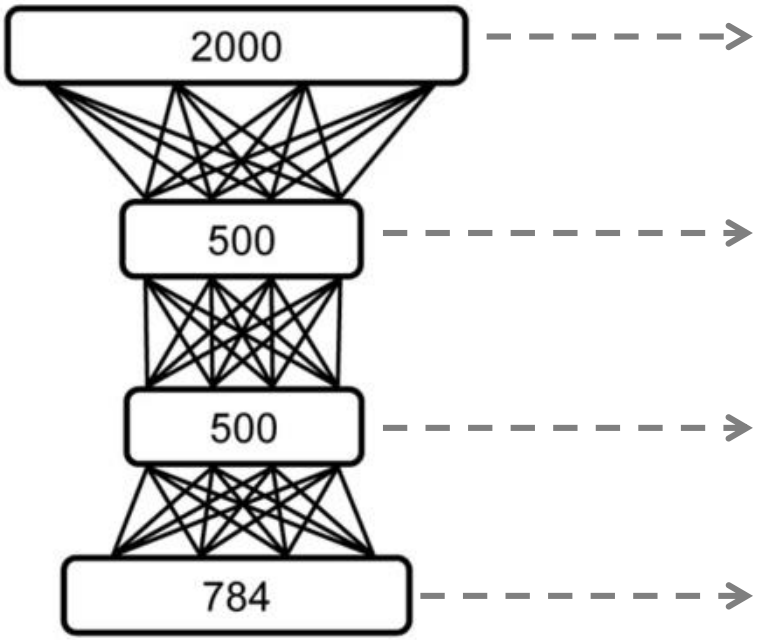
MNIST
(handwritten digits)

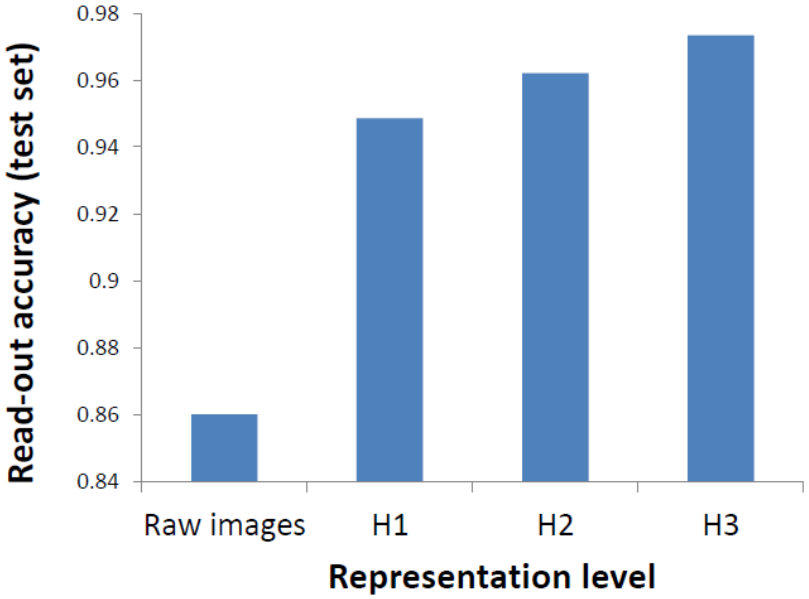
Reading-out internal representations

We can test how well each layer of representation supports a discriminative task



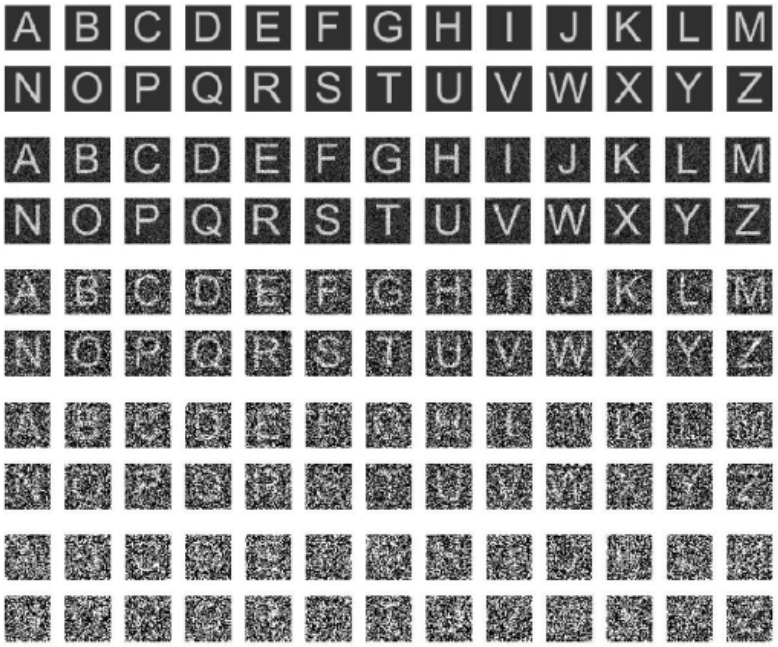
linear read-outs

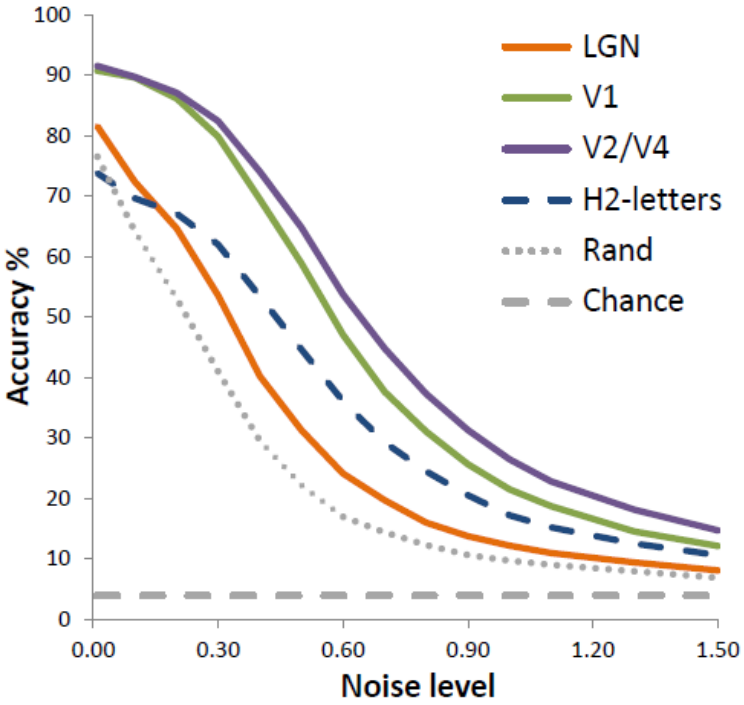




Reading-out internal representations

If accuracy is at ceiling, differences might be found by injecting noise and calculating psychometric functions:





Testolin et al. (2017), N*ature Human Behaviour*

Step 0: Download source code

- For fast learning on graphic processors (GPUs), download source code from our website: http://ccnl.psy.unipd.it/research/deeplearning
  - MATLAB (version > 2012) with Parallel Computing Toolbox
  - Python (version > 2.7) with PyCUDA libraries

- Additional MATLAB routines for useful analyses: http://ccnl.psy.unipd.it/research/dbn-analyses

Step 1: Data preparation

1. Download and unzip the following 4 files from http://yann.lecun.com/exdb/mnist
   * train-images-idx3-ubyte.gz
   * train-labels-idx1-ubyte.gz
   * t10k-images-idx3-ubyte.gz
   * t10k-labels-idx1-ubyte.gz

NB: Make sure file names are not changed during unzipping ('-' is often replaced with '.')

2. Convert raw images into MATLAB/Octave format by copying them into the source code directory and running the routine `'converter.m'`

3. Save the training set into a suitable 3D matrix by running `'makebatches.m'` (mini-batch size can be set inside that file before running it).
This will produce a file called `'MNIST_data_n.mat'`, where *n* is mini-batch size.

We can plot one pattern to make sure everything was correct:

Step 2: DBN training

Now we are ready to train a deep belief network using `'deeptrain_GPU.m'`

Make sure you set all the desired hyper-parameters before proceeding:

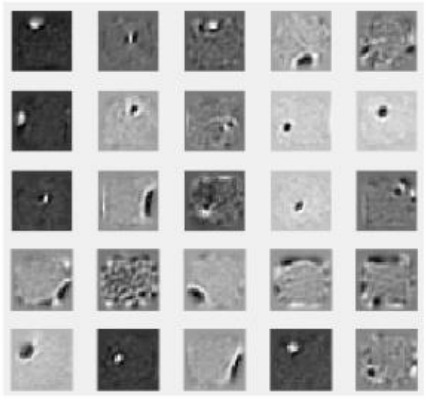
```
% DEEP NETWORK SETUP
% (parameters and final network weights will be saved in structure DN)
DN.layersize   = [500 500 800];        % network architecture
DN.nlayers     = length(DN.layersize);
DN.maxepochs   = 40;                    % unsupervised learning epochs
DN.batchsize   = 125;                   % mini-batch size
sparsity       = 1;                     % set to 1 to encourage sparsity
spars_factor   = 0.05;                  % how much sparsity?
epsilonw_GPU   = gpuArray(0.1);         % learning rate (weights)
epsilonvb_GPU  = gpuArray(0.1);         % learning rate (visible biases)
epsilonhb_GPU  = gpuArray(0.1);         % learning rate (hidden biases)
weightcost_GPU = gpuArray(0.0002);      % decay factor
init_momentum  = 0.5;                   % initial momentum coefficient
final_momentum = 0.9;                   % momentum coefficient
```

During training we should plot the reconstruction error to make sure it converges

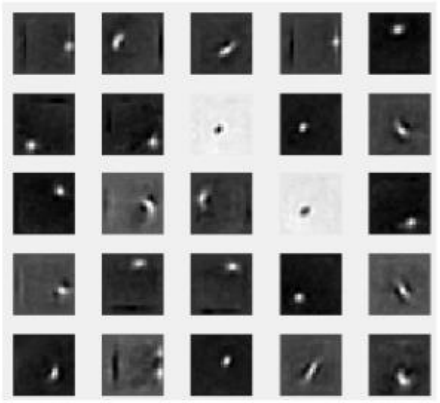
Step 3: DBN analysis

Plot receptive fields at different levels of the hierarchy, given as input a deep belief network and the desired number of hidden units:

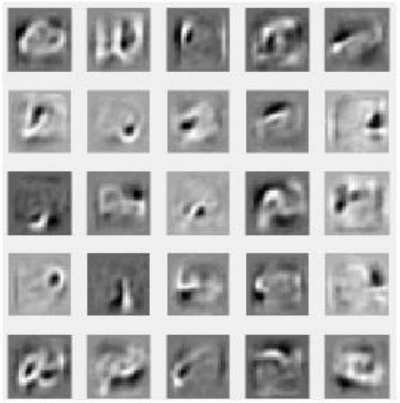
```
plot_L1(DN, 100)
```



```
plot_L2(DN, 100)
```



```
plot_L3(DN, 100)
```



Step 3: DBN analysis

Perform a read-out using a simple linear classifier, given as input a set of training and test patterns with corresponding labels. The function gives as output the weights of the classifier and the training and test accuracies:

```
[W, tr_acc, te_acc] = perceptron(tr_patt, tr_labels,
                                 te_patt, te_labels)
```

- **NB**: first we need to reshape the 3D matrices back to 2D! (and we can use single instead of double to save memory)

- We should start by giving as input to the classifier directly the raw images
- Then we should project the patterns on the internal representations by activating the hidden units (of each layer!) and repeat the read-out:

```
H1_tr = 1./(1 + exp(-tr_patt*DN.L{1}.vishid -
                    repmat(DN.L{1}.hidbiases,size(tr_patt, 1),1)));
H1_te = 1./(1 + exp(-te_patt*DN.L{1}.vishid -
                    repmat(DN.L{1}.hidbiases,size(te_patt, 1),1)));
[W, tr_acc, te_acc] = perceptron(H1_tr, tr_labels,H1_te, te_labels);
```

Useful references

- Testolin & Zorzi (2016). Probabilistic models and generative neural networks: towards an unified framework for modeling normal and impaired neurocognitive functions. *Frontiers in Computational Neuroscience.*
  Perspective paper about how to use unsupervised deep learning for neuropsychological modeling and how to relate it to other network-based approaches.
- Testolin, Stoianov, Sperduti, & Zorzi (2015). Learning orthographic structure with sequential generative neural networks. *Cognitive Science.*
  Cognitive model based on a powerful extension of RBMs that takes into account the temporal structure of the data (i.e., statistical sequence learning).
- Zorzi, Testolin, & Stoianov (2013). Modeling language and cognition with deep unsupervised learning: a tutorial overview. *Frontiers in Psychology*.
  Comprehensive tutorial review of unsupervised deep learning and related methods to build and test psychological models.
- Testolin, Stoianov, De Grazia, & Zorzi (2013). Deep unsupervised learning on a desktop PC : A primer for cognitive scientists. *Frontiers in Psychology*.
  Description and benchmarking of our efficient GPU implementation.
- Hinton (2007). Learning multiple layers of representation. *TICS*.
  Theoretical foundations for unsupervised deep learning.