Machine Learning

Lecture 18: More on neural networks

Stefan Harmeling

13. December 2017

Was haben Sie nicht verstanden?

- Was sind Blöcke? Sind das Funktionen? Werden Daten, die durch ein neuronales Netz gehen durch mehrere 'Funktionen' ausgewertet? A: draw example of $f_3(W_3 \ldots)$ as block diagram.

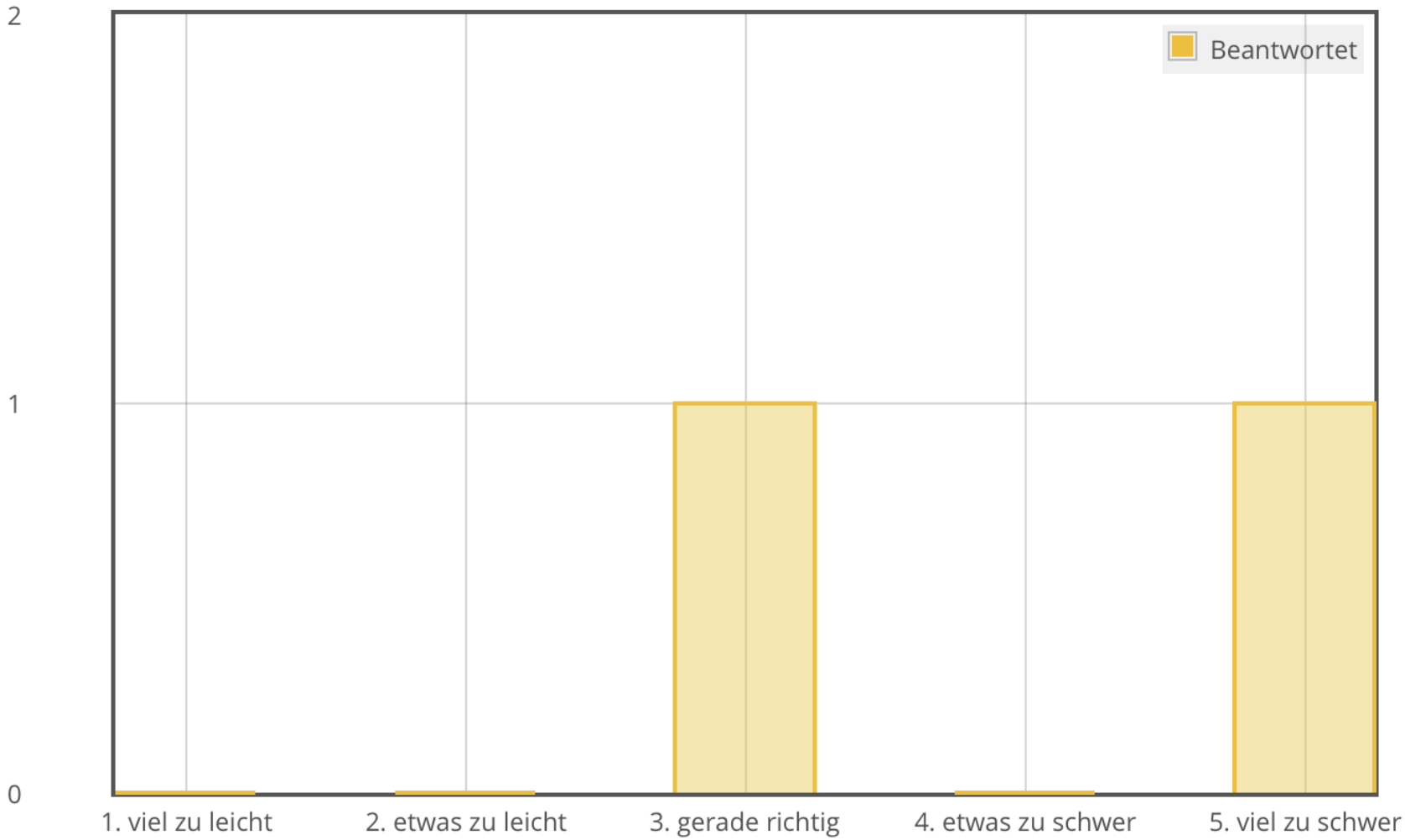
Was hat Ihnen gefallen?

- Sehr viele Erklärungen waren komplex! Es war sehr schwierig Ihnen zu folgen. Beispiele auf der Tafel waren jedoch sehr hilfreich. Mehr Tafelbeispiele, wenn Sie etwas erläutern!

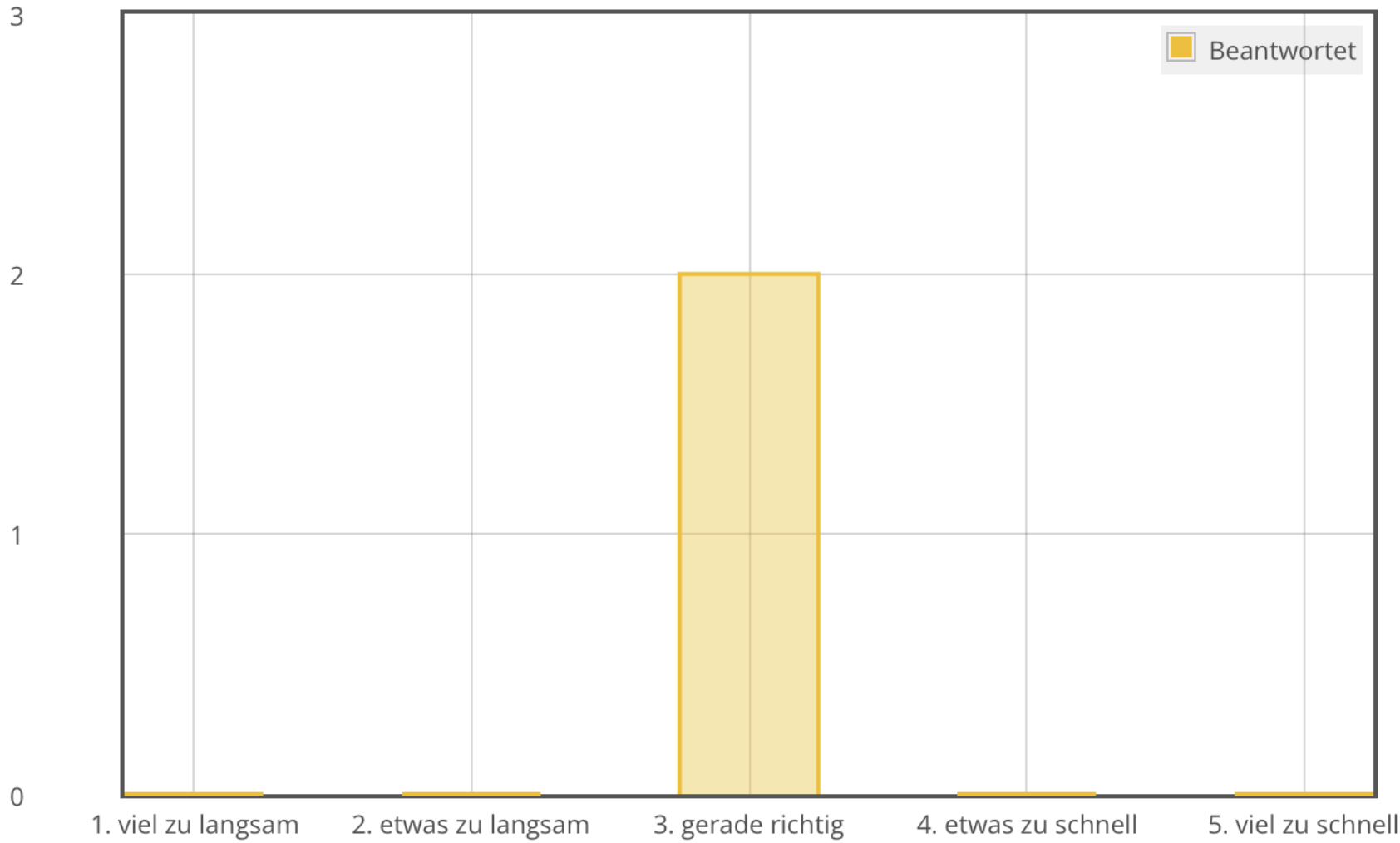
Was hat Ihnen nicht gefallen?

- Alle Folien vor ’Building Blocks for neural networks’. A: exactly those are important! show again

War die Vorlesung zu leicht oder zu schwer?



War die Vorlesung zu schnell oder zu langsam?



Haben Sie weitere Anmerkungen?

- —

Tricks of the trade

General tricks

copied from Burger, Schuler, Harmeling, *Image denoising: Can plain neural networks compete with BM3D*, CVPR 2012.

Goal:

Avoid flat regions of the loss function.
Try to start at the steep parts of the activation functions.

Tricks:

- data normalization: transform input points to have mean zero and variance one
- weight initialization for linear bricks: sample from a normal distribution with mean zero and std $\sigma = \sqrt{N}$ with N being the number of input nodes
- learning rate division: in each linear layer divide the learning rate by N, the number of input nodes

See also:

- LeCun, Bottou, Orr, Müller, “Efficient back-prop”, 1998.

Dropout

- very large network might overfit (i.e. be perfect on training data, but fail on test data)
- *dropout* is a trick during training to reduce overfitting
- during forward pass randomly set hidden nodes to zero (e.g. with probability 0.5)
- this leads to more robust features

Dropout block (component-wise):

Propagation (feed forward)

$$m = [rand(size(x)) <= p]$$
$$y = x \odot m$$

Back propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \odot m$$

Optimization strategies

- stochastic gradient descent with momentum (SGD)

$$v_{t+1} = \mu v_t - \eta \nabla_w E(w_t)$$
$$w_{t+1} = w_t + v_{t+1}$$

- Nesterov acceleration gradient (NAG)
  Nesterov, *A method of solving a convex programming problem with convergence rate (1/sqrt(k)), 1983.*

$$v_{t+1} = \mu v_t - \eta \nabla_w E(w_t + \mu v_t)$$
$$w_{t+1} = w_t + v_{t+1}$$

- AdaGrad
  see Duchi, Hazan, Singer, *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*, JMLR, 2011.

$$(w_{t+1})_i = (w_t)_i - \eta \frac{(\nabla_w E(w_t))_i}{\sqrt{\sum_{t'=1}^{t} (\nabla_w E(w_{t'}))_i^2}}$$

How to choose the architecture/hyperparameters?

Story: MNIST competition

- build your best handwritten digit classifier
- deadline 31.12.2017
- you are only given the training set (about 60,000 examples)
- on 01.01.2018 I will run your classifier on the test examples (about 10,000 digits) to calculate its test-error
- that test error will rank the submitted classifiers
- until then the test data is hidden in a safe
- you have only access to the training data

How can you choose the hyperparameter
w/o accessing the test data?

Given only the training data estimate the test error (loss) using:

Algorithm 18.1 (k-fold cross validation (CV))

Split your training data $\mathcal{D}$ with n examples into k disjoint subsets of approximately equal size:

$$\mathcal{D} = \mathcal{D}_1 \cup \ldots \cup \mathcal{D}_k$$

For $i = 1$ to k

1. *$\mathcal{D}_i$ is called validation set and plays the role of the test set*
2. *train the model on $\mathcal{D} - \mathcal{D}_i$*
3. *calculate validation error ε_i on $\mathcal{D}_i$*

The cross-validation estimate of the test error ε is the average validation error:

$$\varepsilon_{CV} = \frac{1}{k} \sum_{i=1}^{k} \varepsilon_i$$

For $k = n$ this is called leave-one-out cross validation (LOOCV).

How can you choose the hyperparameter w/o accessing the test data?

- estimate the test error via CV for different hyperparameter settings
- CV allows you also to test different network architectures

Note

- the CV estimator can be applied everywhere in machine learning where hyper parameters have to be chosen
- e.g.
  - for setting the regularization constant C in SVMs
  - for setting k in k nearest neighbors (what loss?)
  - for setting the regularization constant in ridge regression
  - etc...

Unsupervised neural networks

Auto-associative neural network (autoencoder)

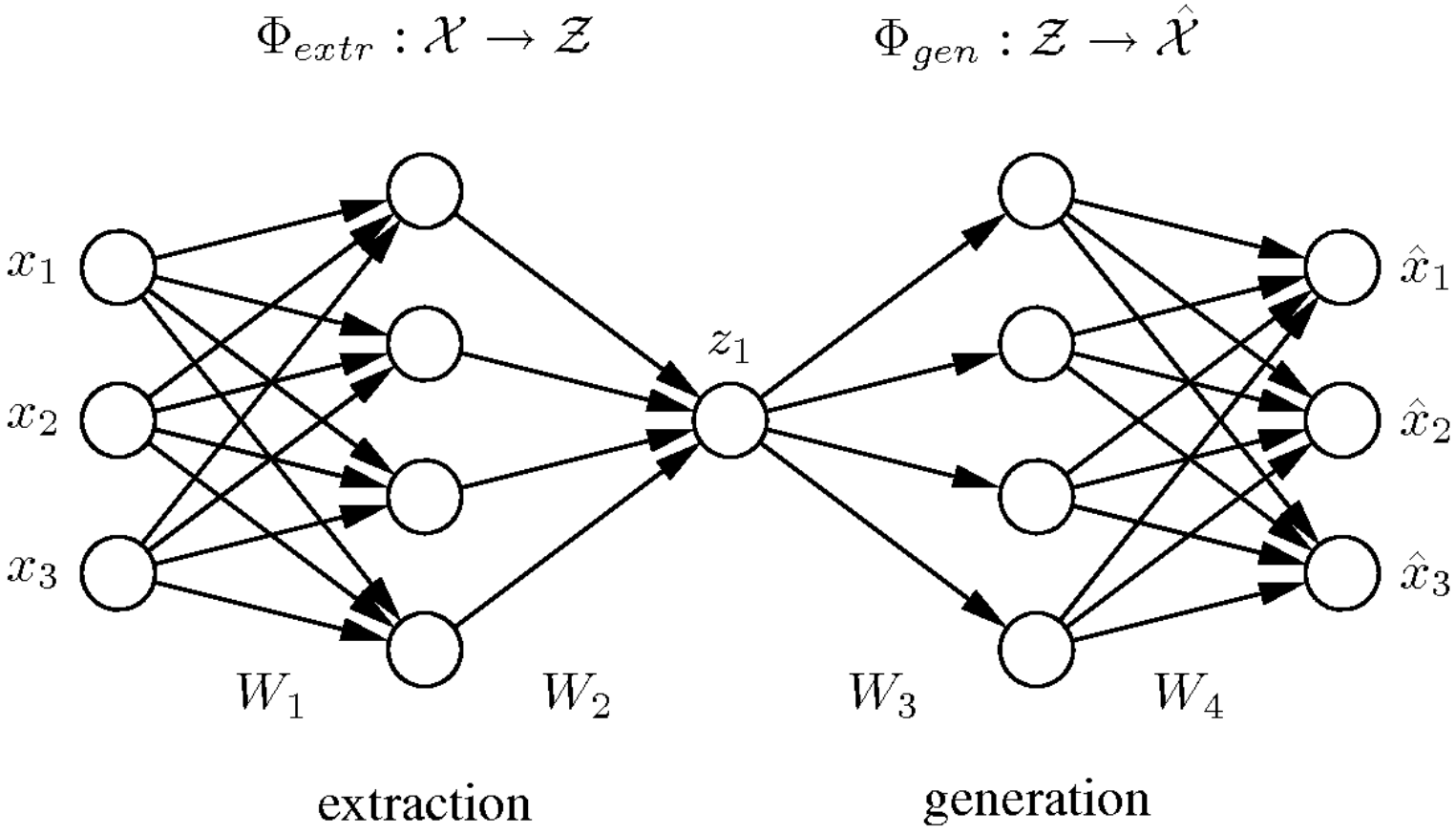
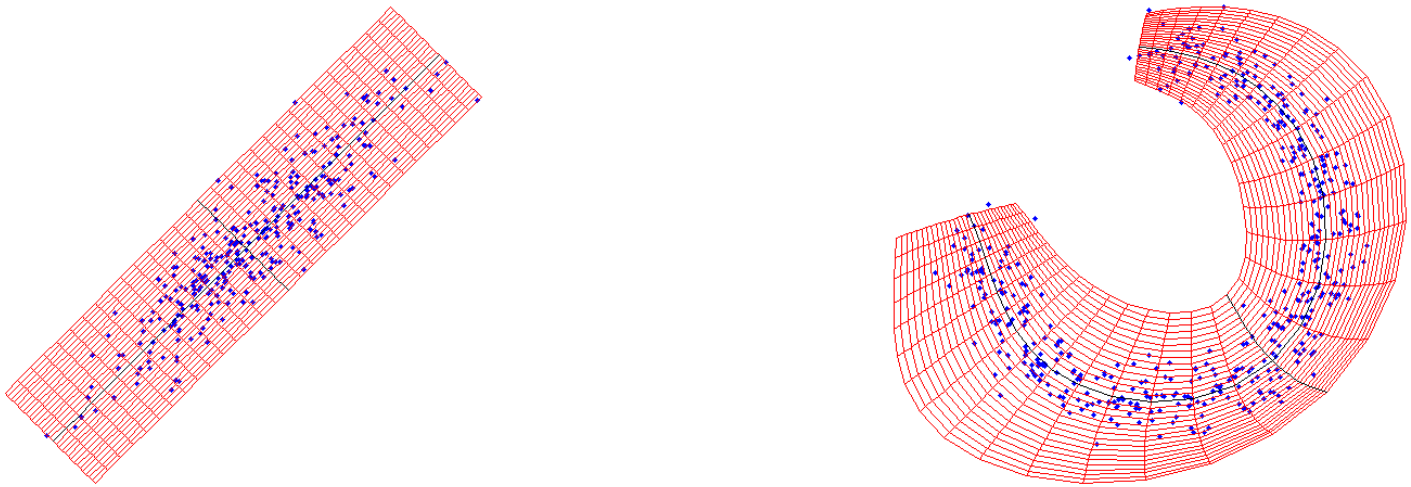


image from Matthias Scholz, `http://nlpca.org` and Scholz, *Nichtlinear Hauptkomponentenanalyse auf Basis neuronaler Netze*, Diplomarbeit, 2002.

see also, Kramer, *Nonlinear principal component analysis using auto- associative neural networks*, AIChE Journal, 1991.

Auto-associative neural network (autoencoder)



from Matthias Scholz, `http://nlpca.org`

Auto-associative neural network (autoencoder)

What is optimized?

- The neural network can be written as follows:

$$f(x) = b_4 + W_4 \tanh(b_3 + w_3 \tanh(b_2 + w_2^T \tanh(b_1 + W_1 x)))$$

  where W_4, W_1 are matrices, w_3, w_2, b_4, b_3, b_1, x are vectors, and b_2 is a scalar.
- the loss function to minimize is

$$E = \|x - f(x)\|^2$$

  i.e. we are trying to reconstruct the input with the output
- network reconstructs a single nonlinear directions (probably of largest variance)

Auto-associative neural network (autoencoder)

What if we want to reconstruct a two dimensional embedding?

- The neural network can be written as follows:

$$f(x) = b_4 + W_4 \tanh(b_3 + W_3 \tanh(b_2 + W_2^T \tanh(b_1 + W_1 x)))$$

  where W_4, W_3, W_2, W_1 are matrices, b_4, b_3, b_2, b_1, x are vectors.
- the loss function to minimize is

$$E = \|x - f(x)\|^2$$

- note that if there are several dimensions for the bottleneck there is no order on the nodes (unlike linear PCA), i.e. it is only about the reconstruction error (same problem as on slide 30 in lecture 11, where we found the two dimensional subspace of largest variance, but with an arbitrary rotation on the single directions),
- Scholz (2002) calls this s-NLPCA, “s” like symmetric.

Auto-associative neural network (autoencoder)

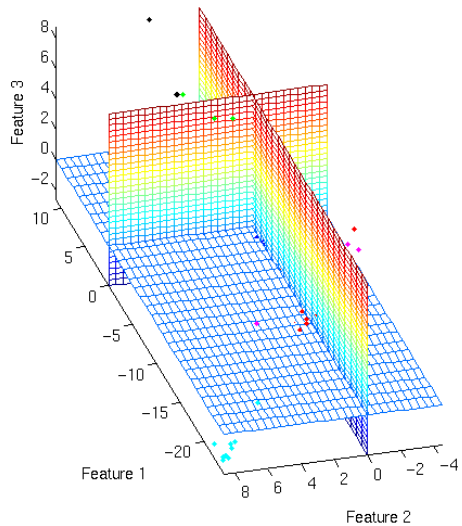
(Hierarchical) h-NLPCA, Scholz, 2002

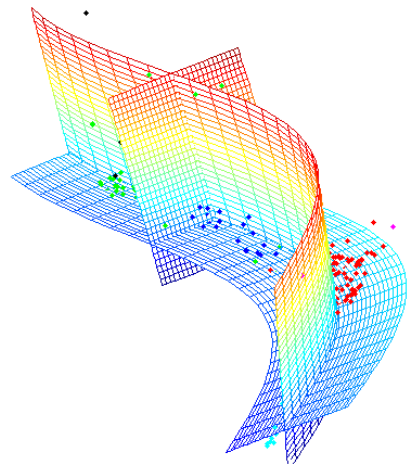
- Scholz suggests an *hierarchical* error function (called h-NLPCA, "h" like hierarchical, e.g. for two dimensions):

$$E = \alpha E_1 + E_{12} \qquad \text{for } 0 \leq \alpha < \infty$$

- For $\alpha = \infty$ we get one-dimensional NLPCA, for $\alpha = 0$ we get s-NLPCA, in between he calls it h-NLPCA.

Auto-associative neural network (autoencoder)

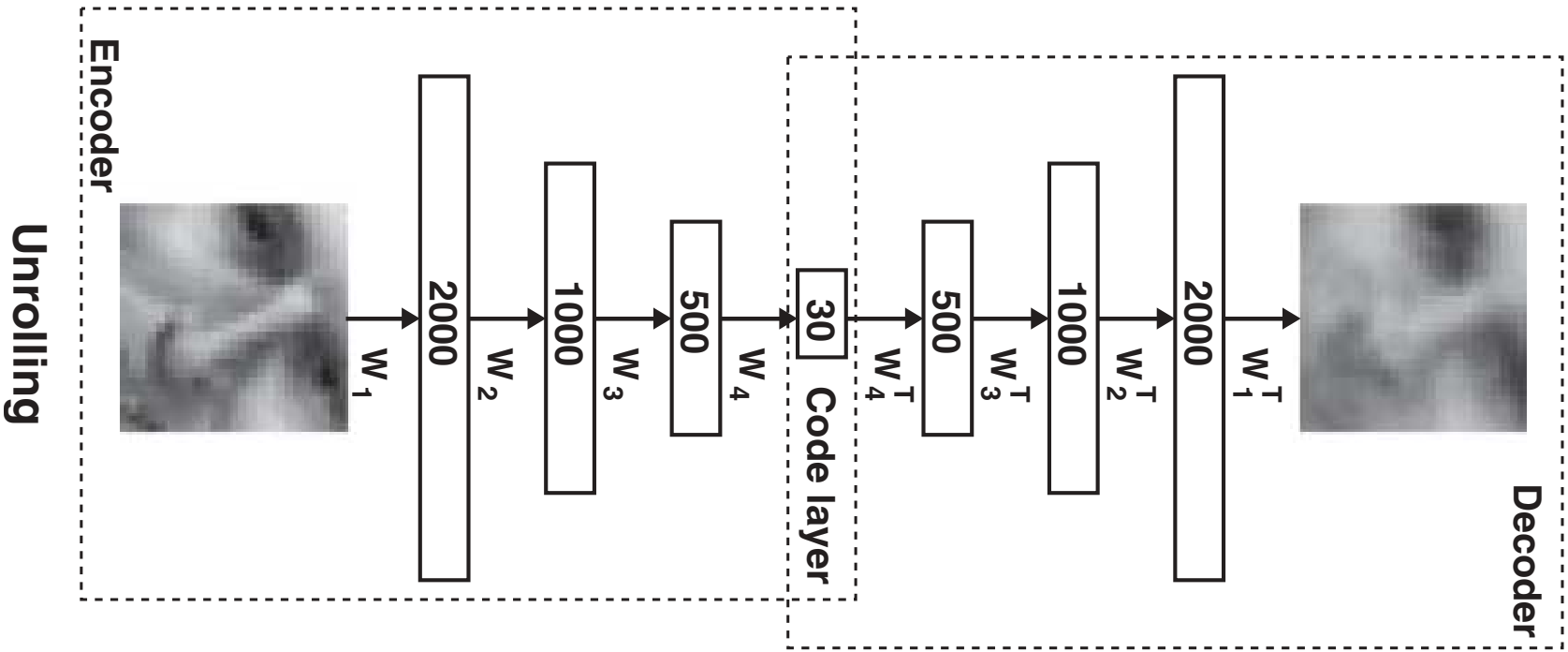




from Matthias Scholz, `http://nlpca.org`

Neural networks with pretraining

A recent unsupervised network, Hinton/Salahutdinov, Science, 2006



from Hinton and Salakhutdinov, *Reducing the Dimensionality of Data with Neural Networks*, Science, 2006.

Neural networks with pretraining

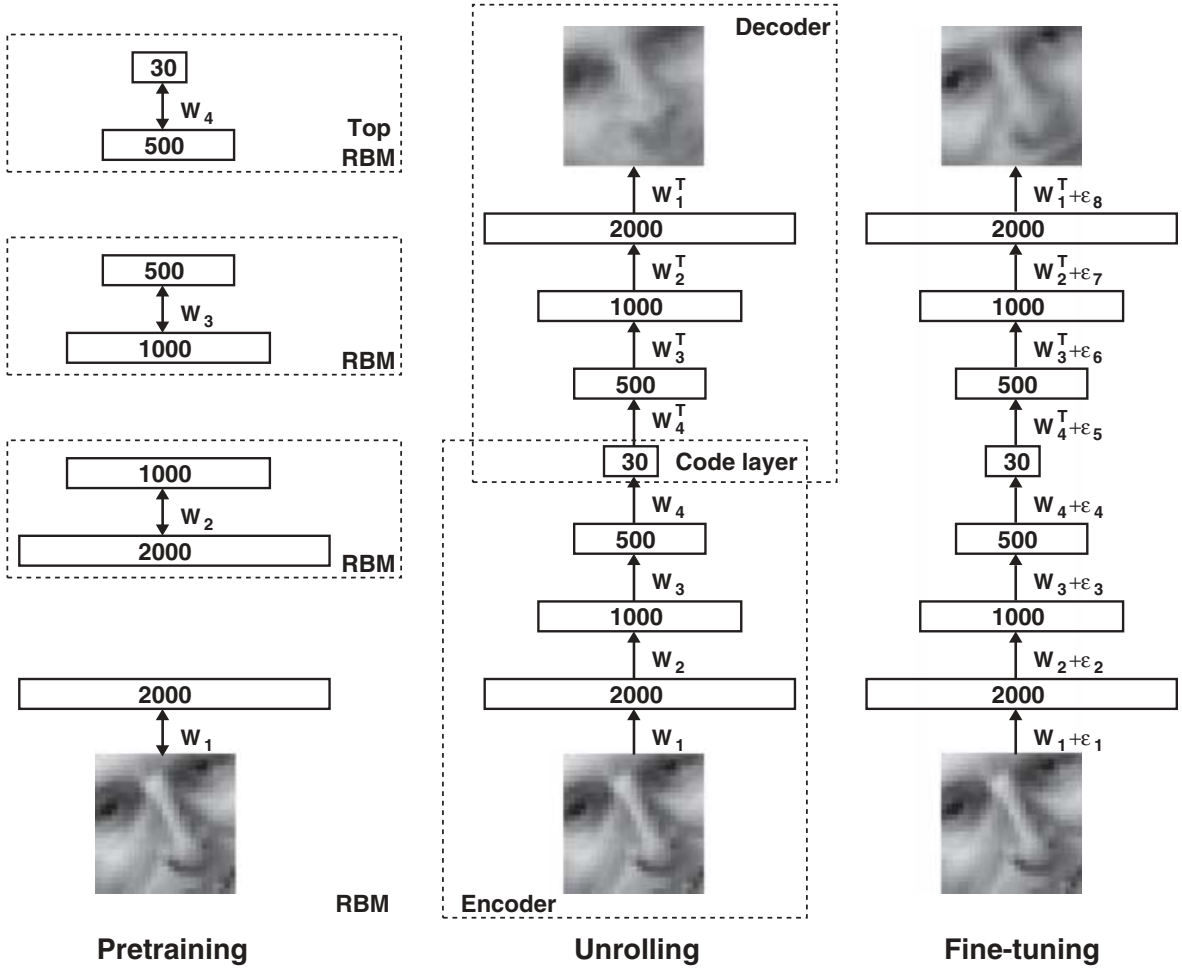


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

from Hinton and Salakhutdinov, *Reducing the Dimensionality of Data with Neural Networks*, Science, 2006.

Neural networks with pretraining

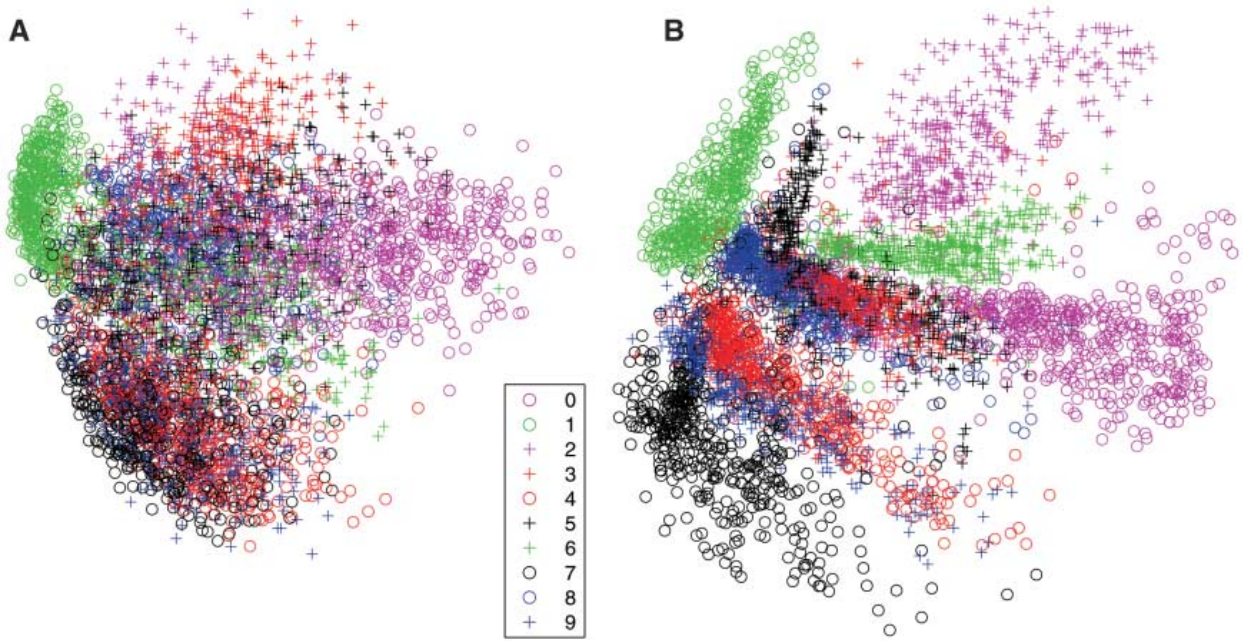


Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

from

Hinton and Salakhutdinov, *Reducing the Dimensionality of Data with Neural Networks*, Science, 2006.