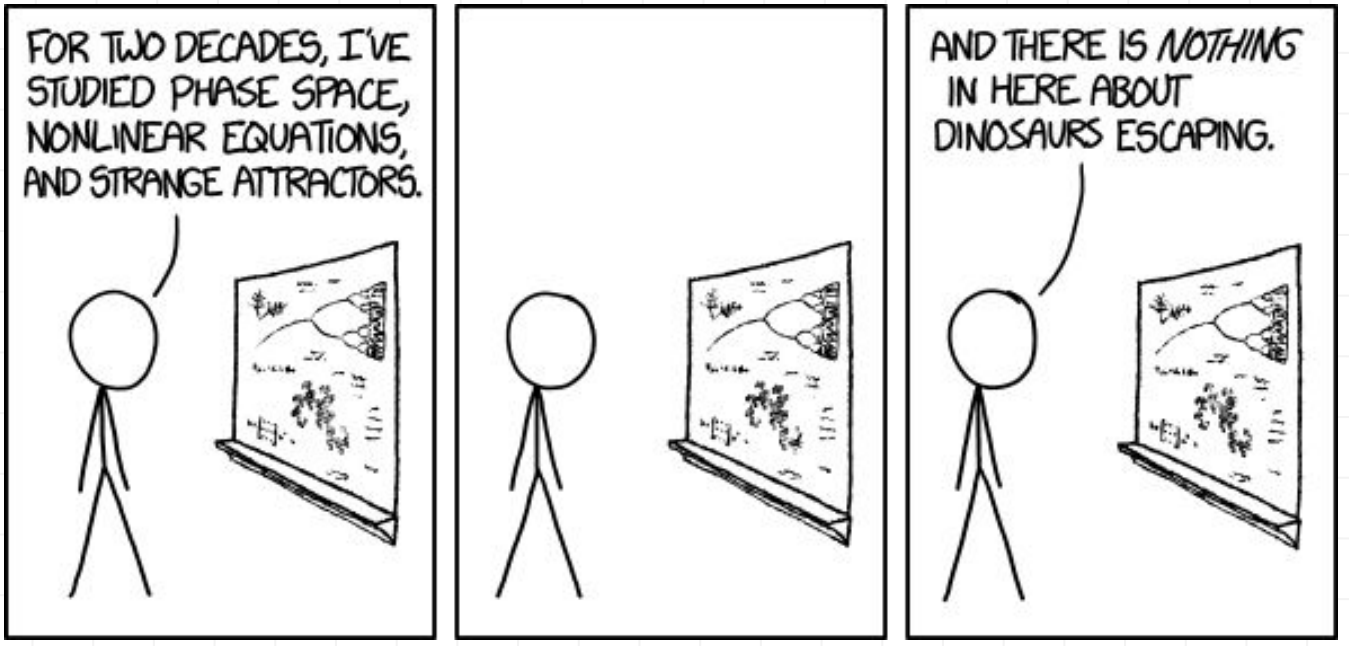
https://xkcd.com/1399/



Linear and Nonlinear Component Analysis

Probabilistic PCA

Autoencoders

Autoencoders for image processing

Recommender systems

[PRML book]
Pre-read PCA - 12.1
This lecture:
12.2, 12.2.1, 12.4.2

Kernel PCA 12.3 (read after week6 lectures)

Recsys:
22.1 of Murphy's PML book

Assignment 1 Q&A Thu 1-3 pm
Fri 3-5 pm

PCA: recap

data $\{x_1, \ldots, x_n\}$

$x_i \in \mathbb{R}^D$ $\quad D \to M$

Covariance matrix of (centered) data

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^{\mathrm{T}}$$

Maximum variance projection

$$\mathbf{x}_n = \sum_{i=1}^{D}\left(\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i\right)\mathbf{u}_i. \qquad (12.9)$$

assignment to diff components.

$$\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

$u_i^T u_i = 1$

$u_i^T u_j = 0$

Minimum error formulation

min $$J = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\left(\mathbf{x}_n^{\mathrm{T}}\mathbf{u}_i - \bar{\mathbf{x}}^{\mathrm{T}}\mathbf{u}_i\right)^2 = \sum_{i=M+1}^{D}\mathbf{u}_i^{\mathrm{T}}\mathbf{S}\mathbf{u}_i. \qquad (12.15) \qquad = \sum_{i=M+1}^{D}\lambda_i$$

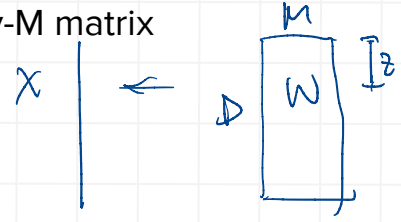
Probabilistic PCA

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}). \qquad (12.31)$$

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I}) \qquad (12.32)$$

x : D-dimensional vector
z : M-dimensional Gaussian latent variable
D >= M

W: D-by-M matrix



Marginal and Conditional Gaussians

Given a marginal Gaussian distribution for $\mathbf{x}$ and a conditional Gaussian distribution for $\mathbf{y}$ given $\mathbf{x}$ in the form

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}) \qquad (2.113)$$
$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}) \qquad (2.114)$$

the marginal distribution of $\mathbf{y}$ and the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ are given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^{\mathrm{T}}) \qquad (2.115)$$
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\Sigma}\{\mathbf{A}^{\mathrm{T}}\mathbf{L}(\mathbf{y} - \mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}\}, \boldsymbol{\Sigma}) \qquad (2.116)$$

where

$$\boldsymbol{\Sigma} = (\boldsymbol{\Lambda} + \mathbf{A}^{\mathrm{T}}\mathbf{L}\mathbf{A})^{-1}. \qquad (2.117)$$

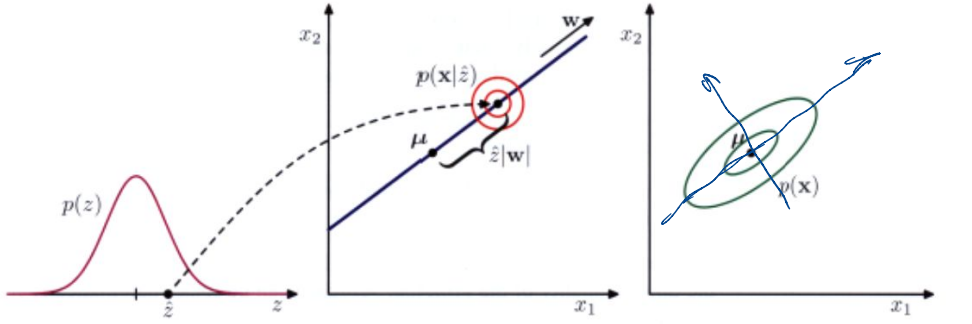


Figure 12.9 An illustration of the generative view of the probabilistic PCA model for a two-dimensional data space and a one-dimensional latent space. An observed data point $\mathbf{x}$ is generated by first drawing a value $\hat{z}$ for the latent variable from its prior distribution $p(z)$ and then drawing a value for $\mathbf{x}$ from an isotropic Gaussian distribution (illustrated by the red circles) having mean $\mathbf{w}\hat{z} + \boldsymbol{\mu}$ and covariance $\sigma^2\mathbf{I}$. The green ellipses show the density contours for the marginal distribution $p(\mathbf{x})$.

Likelihood and posterior for x and z

C^{-1} : needs OCD^3) computation.

also (2.113)-(2.117)

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$$
$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$$

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z}.$$

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{C}) \tag{12.35}$$

$$\mathbf{C} = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I}. \tag{12.36}$$

D-by-D

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$$

$$\begin{aligned}
\mathbb{E}[\mathbf{x}] &= \mathbb{E}[\mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}] = \boldsymbol{\mu} && (12.37)\\
\mathrm{cov}[\mathbf{x}] &= \mathbb{E}\left[(\mathbf{W}\mathbf{z} + \boldsymbol{\epsilon})(\mathbf{W}\mathbf{z} + \boldsymbol{\epsilon})^{\mathrm{T}}\right] \\
&= \mathbb{E}\left[\mathbf{W}\mathbf{z}\mathbf{z}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\right] + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^{\mathrm{T}}] = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I} && (12.38)
\end{aligned}$$

Woodbury / matrix inversion identity

$$(\mathbf{A} + \mathbf{B}\mathbf{D}^{-1}\mathbf{C})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} + \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} \tag{C.7}$$

$$\mathbf{C}^{-1} = \sigma^{-1}\mathbf{I} - \sigma^{-2}\mathbf{W}\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}$$
$$\mathbf{M} = \mathbf{W}^{\mathrm{T}}\mathbf{W} + \sigma^2\mathbf{I}.$$

M-by-M

$O(M^3)$

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}|\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}(\mathbf{x} - \boldsymbol{\mu}), \sigma^{-2}\mathbf{M}\right). \tag{12.42}$$

rotation matrix U

$$\mathbf{C} = \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I}.$$

Maximum likelihood for W, $\boldsymbol{\mu}$, σ^2 → noise,

$P(x) = N(\mu, C)$

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|\mathbf{W}, \boldsymbol{\mu}, \sigma^2)$$

$$= -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{C}| - \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}). \quad (12.43)$$

↳ linear Gaussian latent var.

$$\mathbf{W}_{\mathrm{ML}} = \mathbf{U}_M(\mathbf{L}_M - \sigma^2\mathbf{I})^{1/2}\mathbf{R} \quad (12.45)$$

$$\sigma^2_{\mathrm{ML}} = \frac{1}{D-M}\sum_{i=M+1}^{D}\lambda_i \quad (12.46)$$

L_M: diag with first M eigen vectors.

U_M: first M eigen vectors

R: arbitrary rotation

EM for probabilistic PCA

$$\mathbb{E}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \mathbf{W}, \sigma^2)] = \ldots \ldots$$

p(x|z)

E-step.

$$\mathbb{E}[\mathbf{z}_n] = \mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}}) \tag{12.54}$$

$$\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}] = \sigma^2\mathbf{M}^{-1} + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}} \tag{12.55}$$

M-step.

$$\mathbf{W}_{\text{new}} = \left[\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}\right]\left[\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}]\right]^{-1} \tag{12.56}$$

$$\sigma^2_{\text{new}} = \frac{1}{ND}\sum_{n=1}^{N}\left\{\|\mathbf{x}_n - \overline{\mathbf{x}}\|^2 - 2\mathbb{E}[\mathbf{z}_n]^{\mathrm{T}}\mathbf{W}_{\text{new}}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}}) + \mathrm{Tr}\left(\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^{\mathrm{T}}]\mathbf{W}_{\text{new}}^{\mathrm{T}}\mathbf{W}_{\text{new}}\right)\right\}. \tag{12.57}$$

why EM?

MLE for W, σ^2 requires S

EM: only need M

Linear and Nonlinear Component Analysis

Probabilistic PCA

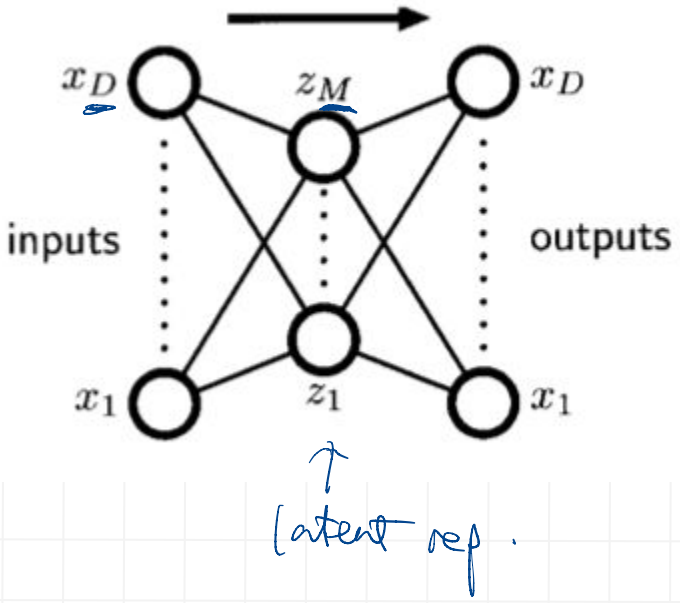
Autoencoders

Autoencoders for image processing

Recommender systems

Two-layer associative neural nets

Figure 12.18 An autoassociative multilayer perceptron having two layers of weights. Such a network is trained to map input vectors onto themselves by minimization of a sum-of-squares error. Even with nonlinear units in the hidden layer, such a network is equivalent to linear principal component analysis. Links representing bias parameters have been omitted for clarity.

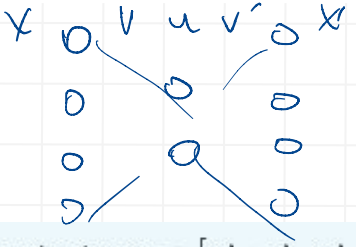


Linear autoencoder ➔ PCA solution

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{x}_n\|^2. \tag{12.91}$$

If the hidden units have linear activations functions, then it can be shown that the error function has a unique global minimum, and that at this minimum the network performs a projection onto the M-dimensional subspace which is spanned by the first M principal components of the data (Bourlard and Kamp, 1988; Baldi and Hornik, 1989). Thus, the vectors of weights which lead into the hidden units in Figure 12.18 form a basis set which spans the principal subspace. Note, however, that these vectors need not be orthogonal or normalized. This result is unsurprising, since both principal component analysis and the neural network are using linear dimensionality reduction and are minimizing the same sum-of-squares error function.

Linear autoencoder ➔ PCA solution



Consider an autoencoder with four inputs $\mathbf{x} = [x_1, x_2, x_3, x_4]$, two hidden units $\mathbf{u} = [u_1, u_2]$, and four outputs $\mathbf{x'} = [x'_1, x'_2, x'_3, x'_4]$. Both the input and output layers have linear activation. Given a training data set $\mathbf{X} \in \mathbf{R}^{10000 \times 4}$ with zero mean, denote the covariance matrix of $\mathbf{X}$ as $\mathbf{S}$, with corresponding eigen vectors $\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_4}]$ satisfying $\mathbf{S}\mathbf{v}_i = \lambda_i \mathbf{v}_i, i = 1, 2, 3, 4$, and $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4$. We also denote the row vectors of $\mathbf{V}$ as $\mathbf{V} = [\mathbf{v}_{:,1}; \mathbf{v}_{:,2}; \mathbf{v}_{:,3}; \mathbf{v}_{:,4}]$, where ";" denote vertical concatenation of row vectors.

Now we set the weights of the input layer to $\mathbf{v}_1$ and $\mathbf{v}_2$ for each hidden unit, respectively. We also set the weight for the output layer to be $\mathbf{v}_{:,1}$ and $\mathbf{v}_{:,2}$, respectively. Which of the following statements are correct:

Select one or more:

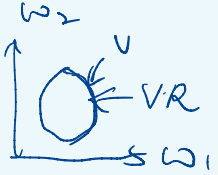
A. If we aim to minimize the square error between X and X' with a commonly used L2 regulariser, the weights will remain unchanged.

B. None of the other options.

C. X' is the minimum error reconstruction of X given this network structure.

D. If we perform gradient descent on the square difference between $\mathbf{x}$ and $\mathbf{x}'$, the weight will remain unchanged.

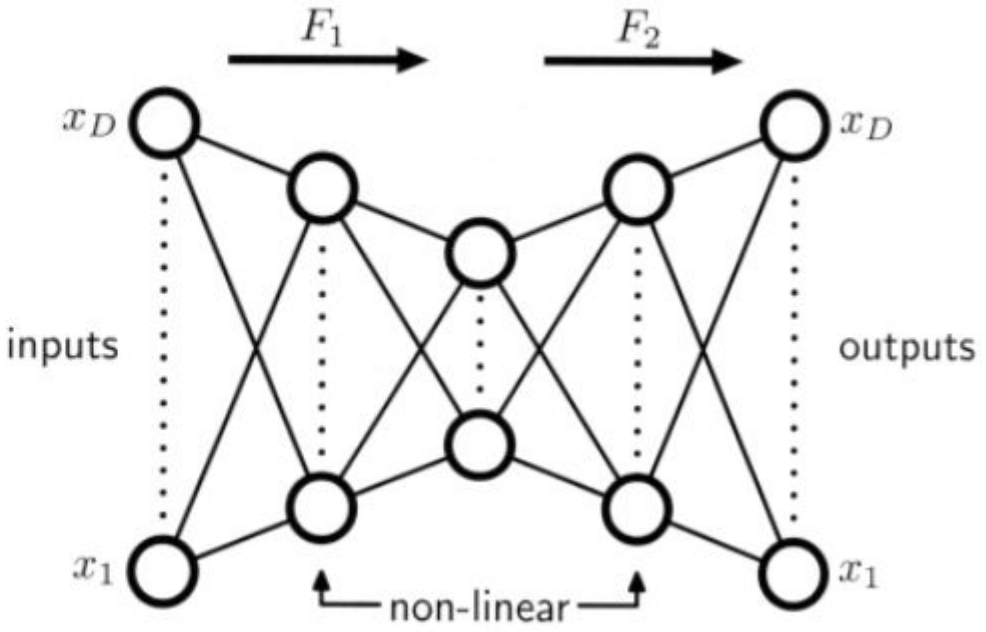
E. The variance of X' is maximised given this network structure.



U.A. of NN
"any" function $z = f(x)$

$x \xrightarrow{\text{two layer}} z \rightarrow x'$

Figure 12.19 Addition of extra hidden layers of nonlinear units gives an autoassociative network which can perform a nonlinear dimensionality reduction.



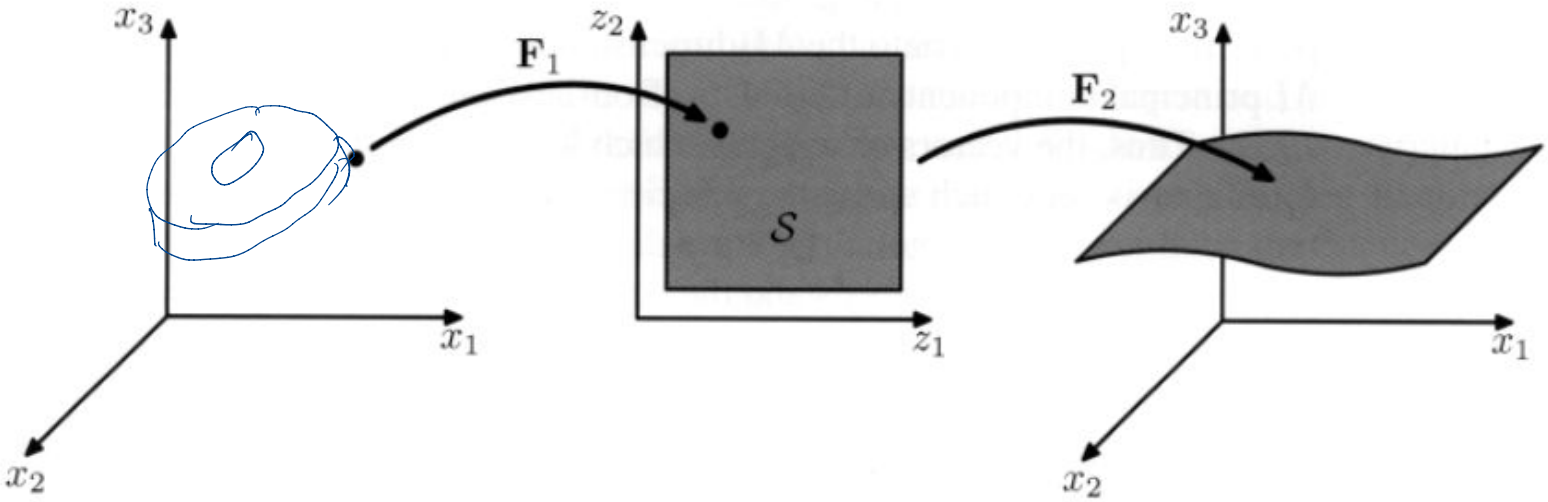


Figure 12.20 Geometrical interpretation of the mappings performed by the network in Figure 12.19 for the case of $D = 3$ inputs and $M = 2$ units in the middle hidden layer. The function $\mathbf{F}_2$ maps from an M-dimensional space $\mathcal{S}$ into a D-dimensional space and therefore defines the way in which the space $\mathcal{S}$ is embedded within the original $\mathbf{x}$-space. Since the mapping $\mathbf{F}_2$ can be nonlinear, the embedding of $\mathcal{S}$ can be nonplanar, as indicated in the figure. The mapping $\mathbf{F}_1$ then defines a projection of points in the original D-dimensional space into the M-dimensional subspace $\mathcal{S}$.

Easier to represent with more layers

An old result:

- functions that can be compactly represented by a depth k architecture might require an exponential number of computational elements to be represented by a depth $k-1$ architecture
- Example, the d bit parity function

$$\text{parity} : (b_1, \ldots, b_d) \in \{0,1\}^d \mapsto \begin{cases} 1 & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\ 0 & \text{otherwise} \end{cases}$$

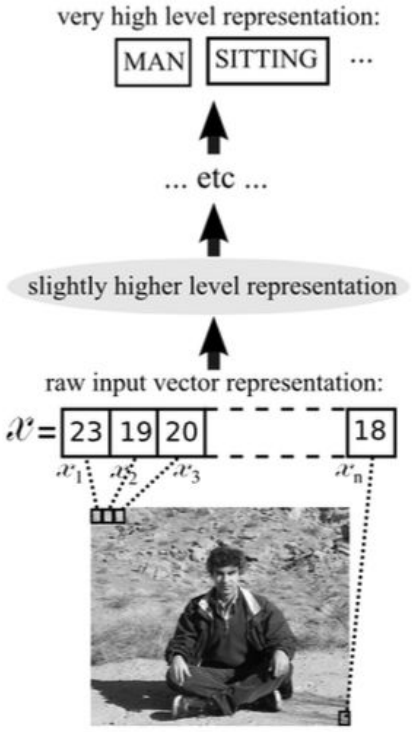
- Theorem: d-bit parity circuits of depth 2 have exponential size

"On the Expressive Power of Deep Architectures", Bengio and Delalleau, 2011

Yao, A. (1985). *Separating the polynomial-time hierarchy by oracles*. In Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, pages 1–10. FOCS

Håstad, J. (1986). *Almost optimal lower bounds for small depth circuits*. In Proceedings of the 18th annual ACM Symposium on Theory of Computing, pages 6–20, Berkeley, California.

Deep representation - intuition

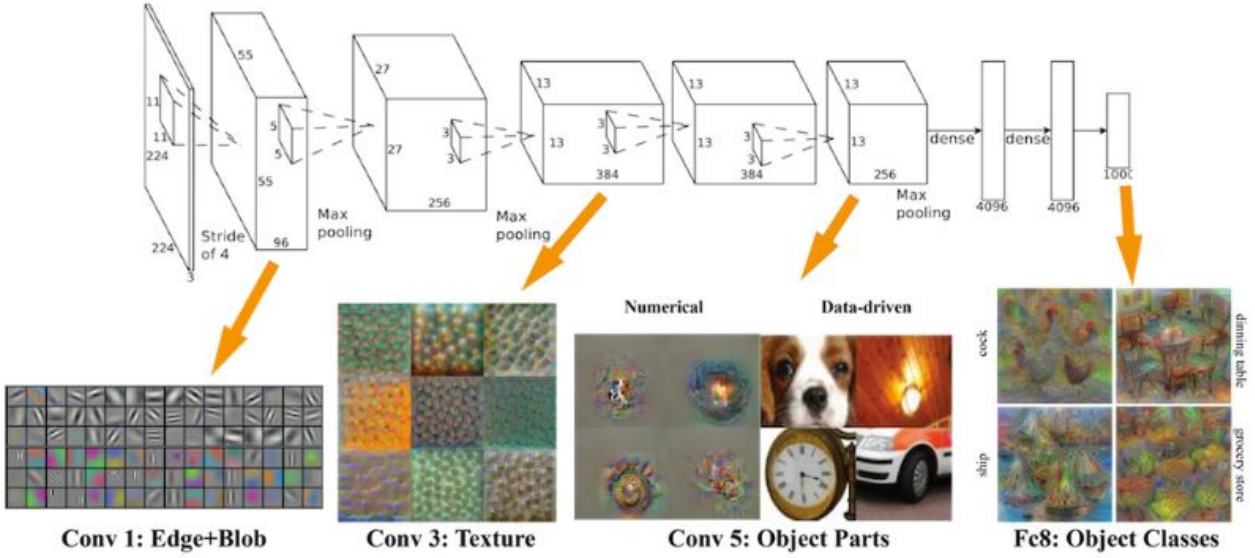


Bengio, "Learning Deep Architectures for AI", 2009

Challenges of “just add more layers”

- Deep architectures get stuck in local minima or plateaus
- As architecture gets deeper, more difficult to obtain good generalisation
- Hard to initialise random weights well
- 1 or 2 hidden layers seem to perform better
- 2006: Unsupervised pre-training, find distributed representation

Deep representation - practice



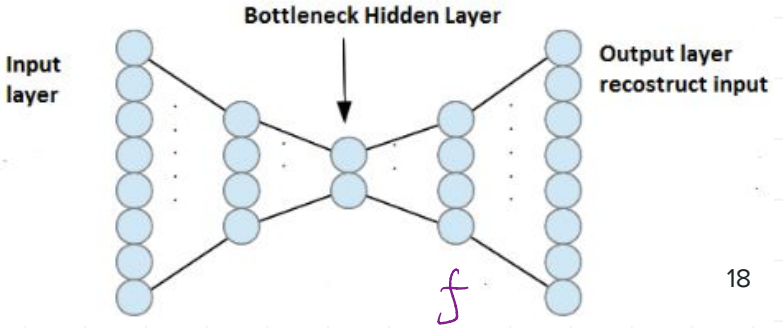
AlexNet / VGG-F network visualized by mNeuron.

Autoencoder

- An autoencoder is trained to encode the input x into some representation $c(x)$ so that the input can be reconstructed from that representation
- the target output of the autoencoder is the autoencoder input itself
- With one linear hidden layer and the mean squared error criterion, the k hidden units learn to project the input in the span of the first k principal components of the data
- If the hidden layer is nonlinear, the autoencoder behaves differently from PCA, with the ability to capture multimodal aspects of the input distribution
- Let f be the decoder. We want to minimise the reconstruction error

$$\sum_{n=1}^{N} \ell\left(x_n, f(c(x_n))\right)$$

c(x)



Cost function of an autoencoder

- Recall: $f(c(x))$ is the reconstruction produced by the network
- Minimisation of the negative log likelihood of the reconstruction, given the encoding $c(x)$

$$\text{RE} = -\log P(x|c(x))$$

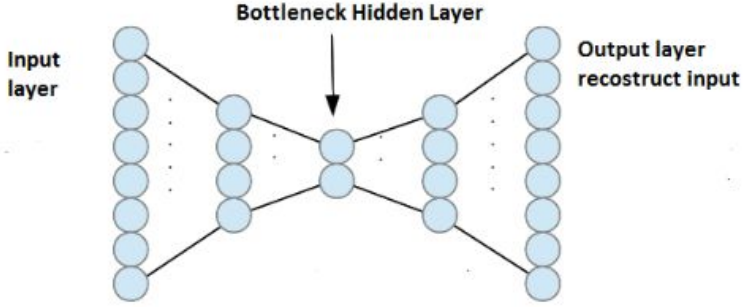
- If $x|c(x)$ is Gaussian, we recover the familiar squared error
- If the inputs x_i are either binary or considered to be binomial probabilities, then the loss function would be the cross entropy

$$-\log P(x|c(x)) = -x_i \log f_i(c(x)) + (1 - x_i) \log(1 - f_i(c(x)))$$

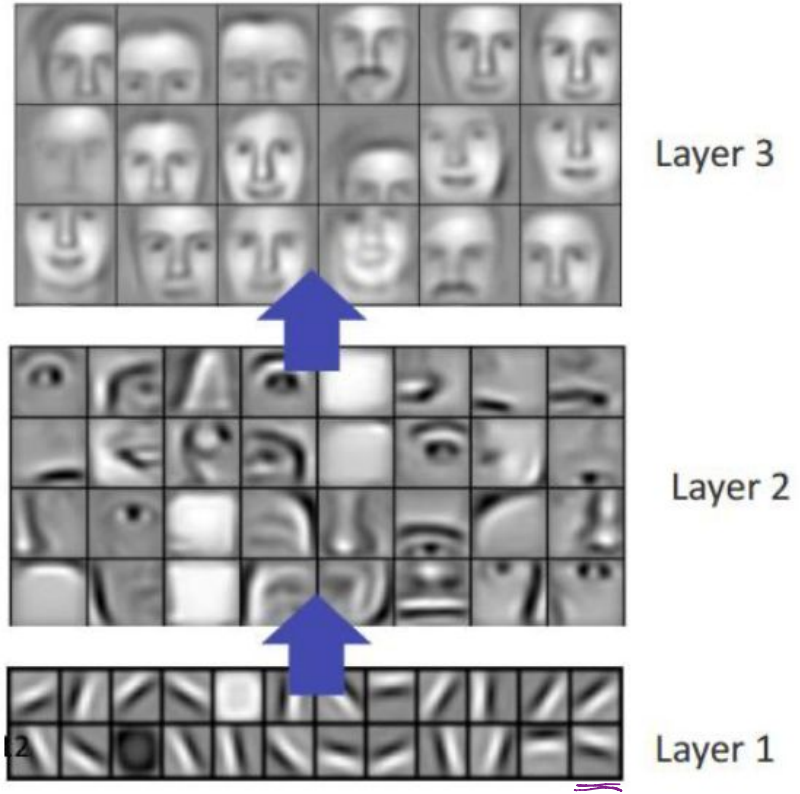
where $f_i(\cdot)$ is the i^{th} component of the decoder

Stacking autoencoders

- Let c_j and f_j be the encoder and corresponding decoder of the j^{th} layer
- Let z_j be the representation after the encoder c_j
- We can define multiple layers of autoencoders recursively.
- For example, let $z_1 = c_1(x)$, and $z_2 = c_2(z_1)$, the corresponding decoding is given by $f_1(f_2(z_2))$
- Because of non-linear activation functions, the latent feature z_2 can capture more complex patterns than z_1.



Higher level image features - faces



codingplayground.blogspot.com

Pre-training supervised neural networks

- Latent features z_j in layer j can capture high level patterns

$$z_j = c_j(c_{j-1}(\cdots c_2(c_1(x))\cdots))$$

- These features may also be useful for supervised learning tasks.
- In contrast to the feed forward network, the features z_j are constructed in an unsupervised fashion.
- Discard the decoding layers, and directly use z_j with a supervised training method, such as logistic regression.
- Various such pre-trained networks are available on-line, e.g VGG-19.

- task transfer.
  - CV
  - NLP
  - ...
- self-supervision.

Xavier Initialisation / ReLU

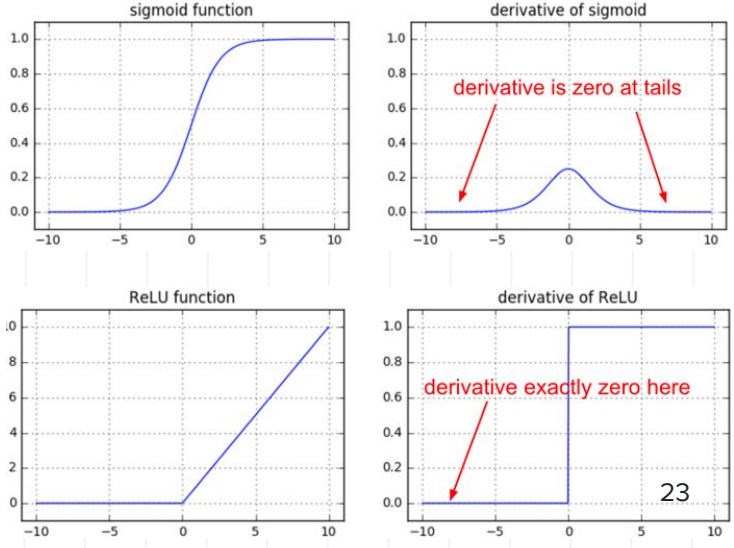
Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In AISTATS 2010 pp. 249-256.

- Layer-wise unsupervised pre-training helps by extracting useful features for subsequent supervised backprop.
- Pre-training also avoids saturation (large magnitude arguments to, say, sigmoidal units).
- Simpler Xavier initialization can also avoid saturation.
- Let the inputs $x_i \sim \mathcal{N}(0,1)$, weights $w_i \sim \mathcal{N}(0,\sigma^2)$ and activation $z = \sum_{i=1}^{m} x_i w_i$. Then:

$$\text{VAR}[z] = \mathbb{E}[(z - \mathbb{E}[z])^2] = \mathbb{E}[z^2] = \mathbb{E}[(\sum_{i=1}^{m} x_i w_i)^2]$$

$$= \sum_{i=1}^{m} \mathbb{E}[(x_i w_i)^2] = \sum_{i=1}^{m} \mathbb{E}[x_i^2]\mathbb{E}[w_i^2] = m\sigma^2.$$

- So we set $\sigma = 1/\sqrt{m}$ to have "nice" activations.
- Glorot initialization takes care to have nice back-propagated signals — see the auto-encoder lab.
- ReLU activations $h(x) = \max(x, 0)$ also help in practice.



Linear and Nonlinear Component Analysis

Probabilistic PCA

Autoencoders

Autoencoders for image processing

Recommender systems

Denoising autoencoder

- Add noise to input, keeping perfect example as output
- Autoencoder tries to:
  1. preserve information of input
  2. undo stochastic corruption process
- Reconstruction log likelihood

$$-\log P(x|c(\hat{x}))$$

where x noise free, $\hat{x}$ corrupted

noisy x’ c(x) clean x

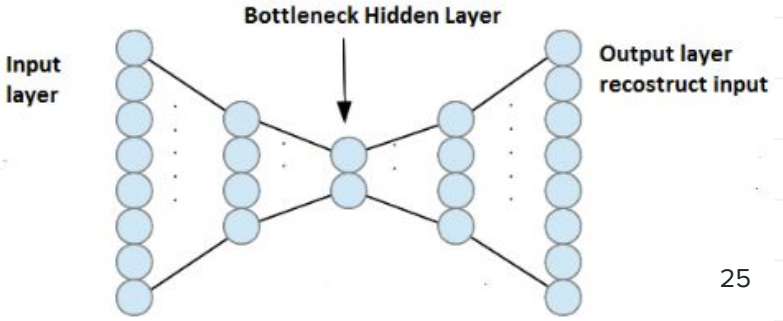
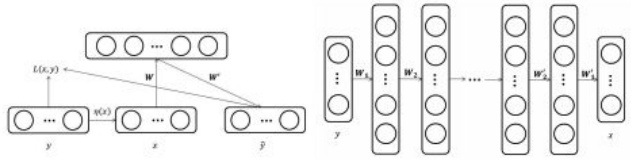


Image denoising

- Images with Gaussian noise added.

- Denoised using Stacked Sparse Denoising Autoencoder

Xie, Junyuan, Linli Xu, and Enhong Chen. "Image denoising and inpainting with deep neural networks." *Advances in neural information processing systems* 25 (2012): 341-349.



(a) Denoising auto-encoder (DA) architecture (b) Stacked sparse denoising auto-encoder architecture

Figure 1: Model architectures.

2.1 Problem Formulation

Assuming $\mathbf{x}$ is the observed noisy image and $\mathbf{y}$ is the original noise free image, we can formulate the image corruption process as:

$$\mathbf{x} = \eta(\mathbf{y}). \tag{1}$$

where $\eta : \mathbb{R}^n \to \mathbb{R}^n$ is an arbitrary stochastic corrupting process that corrupts the input. Then, the denoising task's learning objective becomes:

$$f = \underset{f}{\operatorname{argmin}} \mathbf{E}_{\mathbf{y}} \|f(\mathbf{x}) - \mathbf{y}\|_2^2 \tag{2}$$

From this formulation, we can see that the task here is to find a function f that best approximates η^{-1}. We can now treat the image denoising and inpainting problems in a unified framework by choosing appropriate η in different situations.

2.2 Denoising Auto-encoder

Let $\mathbf{y_i}$ be the original data for $i = 1, 2, ..., N$ and $\mathbf{x_i}$ be the corrupted version of corresponding $\mathbf{y_i}$. DA is defined as shown in Fig.1a:

$$\mathbf{h}(\mathbf{x_i}) = \sigma(\mathbf{W}\mathbf{x_i} + \mathbf{b}) \tag{3}$$

$$\hat{\mathbf{y}}(\mathbf{x_i}) = \sigma(\mathbf{W'}\mathbf{h}(\mathbf{x_i}) + \mathbf{b'}) \tag{4}$$

where $\sigma(x) = (1 + \exp(-x))^{-1}$ is the sigmoid activation function which is applied element-wise to vectors, $\mathbf{h_i}$ is the hidden layer activation, $\hat{\mathbf{y}}(\mathbf{x_i})$ is an approximation of $\mathbf{y_i}$ and $\mathbf{\Theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{W'}, \mathbf{b'}\}$ represents the weights and biases. DA can be trained with various optimization methods to minimize the reconstruction loss:

$$\theta = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \|\mathbf{y_i} - \hat{\mathbf{y}}(\mathbf{x_i})\|. \tag{5}$$

After finish training a DA, we can move on to training the next layer by using the hidden layer activation of the first layer as the input of the next layer. This is called Stacked denoising auto-encoder (SDA) [21].

2.3 Stacked Sparse Denoising Auto-encoders

In this section, we will describe the structure and optimization objective of the proposed model Stacked Sparse Denoising Auto-encoders (SSDA). Due to the fact that directly processing the entire image is intractable, we instead draw overlapping patches from the image as our data objects. In the training phase, the model is supplied with both the corrupted noisy image patches $\mathbf{x_i}$, for $i = 1, 2, ..., N$, and the original patches $\mathbf{y_i}$. After training, SSDA will be able to reconstruct the corresponding clean image given any noisy observation.

To combine the virtues of sparse coding and neural networks and avoid over-fitting, we train a DA to minimize the reconstruction loss regularized by a sparsity-inducing term:

$$L_1(\mathbf{X}, \mathbf{Y}; \theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \|\mathbf{y_i} - \hat{\mathbf{y}}(\mathbf{x_i})\|_2^2 + \beta \, \mathrm{KL}(\hat{\rho} \| \rho) + \frac{\lambda}{2} (\|\mathbf{W}\|_F^2 + \|\mathbf{W'}\|_F^2) \tag{6}$$

Method	Standard deviation σ		
	25/PSNR=20.17	50/PSNR=14.16	100/PSNR=8.13
SSDA	30.52 ± 1.02	27.37 ± 1.10	24.18 ± 1.39
BLS-GSM	30.49 ± 1.17	27.28 ± 1.44	24.37 ± 1.36
KSVD	30.96 ± 0.77	27.34 ± 1.11	23.50 ± 1.15

Table 1: Comparison of the denoising performance. Performance is measured by Peak Signal to Noise Ratio (PSNR). Results are averaged over testing set.

where

$$\mathrm{KL}(\hat{\rho} \| \rho) = \sum_{j=1}^{|\hat{\rho}|} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{(1 - \rho)}{1 - \hat{\rho}_j}, \qquad \hat{\rho} = \frac{1}{N} \sum_{i}^{N} \mathbf{h}(\mathbf{x_i}).$$

and $\mathbf{h}(\cdot)$ and $\hat{\mathbf{y}}(\cdot)$ are defined in (3), (4) respectively. Here $\hat{\rho}$ is the average activation of the hidden layer. We regularize the hidden layer representation to be sparse by choosing small ρ so that the KL-divergence term will encourage the mean activation of hidden units to be small. Hence the hidden units will be zero most of the time and achieve sparsity.

After training of the first DA, we use $\mathbf{h}(\mathbf{y_i})$ and $\mathbf{h}(\mathbf{x_i})$ as the clean and noisy input respectively for the second DA. This is different from the approach described in [21], where $\mathbf{x_i}$ is discarded and $\eta(\mathbf{h}(\mathbf{y_i}))$ is used as the noisy input. We point out that our method is more natural in that, since $\mathbf{h}(\mathbf{y_i})$ lies in a different space from $\mathbf{y_i}$, the meaning of applying $\eta(\cdot)$ to $\mathbf{h}(\mathbf{y_i})$ is not clear.

We then initialize a deep network with the weights obtained from K stacked DAs. The network has one input layer, one output and $2K - 1$ hidden layers as shown in Fig.1b. The entire network is then trained using the standard back-propagation algorithm to minimize the following objective:

$$L_2(\mathbf{X}, \mathbf{Y}; \theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \|\mathbf{y_i} - \mathbf{y}(\mathbf{x_i})\|_2^2 + \frac{\lambda}{2} \sum_{j=1}^{2K} (\|\mathbf{W_j}\|_F^2). \tag{7}$$

Here we removed the sparsity regularization because the pre-trained weights will serve as regularization to the network [18].

In both of the pre-training and fine-tuning stages, the loss functions are optimized with L-BFGS algorithm (a Quasi-Newton method) which, according to [22], can achieve fastest convergence in our settings.

3 Experiments

We narrow our focus down to denoising and inpainting of grey-scale images, but there is no difficulty in generalizing to colored images. We use a set of natural images collected from the web[1] as our training set and standard testing images[2] as the testing set. We create noisy images from clean training and testing images by applying the function (1) to them. Image patches are then extracted from both clean and noisy images to train SSDAs. We employ Peak Signal to Noise Ratio (PSNR) to quantify denoising results: $10 \log_{10}(255^2/\sigma_e^2)$, where σ_e^2 is the mean squared error. PSNR is one of the standard indicators used for evaluating image denoising results.

3.1 Denoising White Gaussian Noise

We first corrupt images with additive white Gaussian noise of various standard deviations. For the proposed method, one SSDA model is trained for each noise level. We evaluate different hyper-parameter combinations and report the best result. We set K to 2 for all cases because adding more layers may slightly improve the performance but require much more training time. In the meantime, we try different patch sizes and find that higher noise level generally requires larger patch size. The

[1] http://decsai.ugr.es/cvg/dbimagenes/

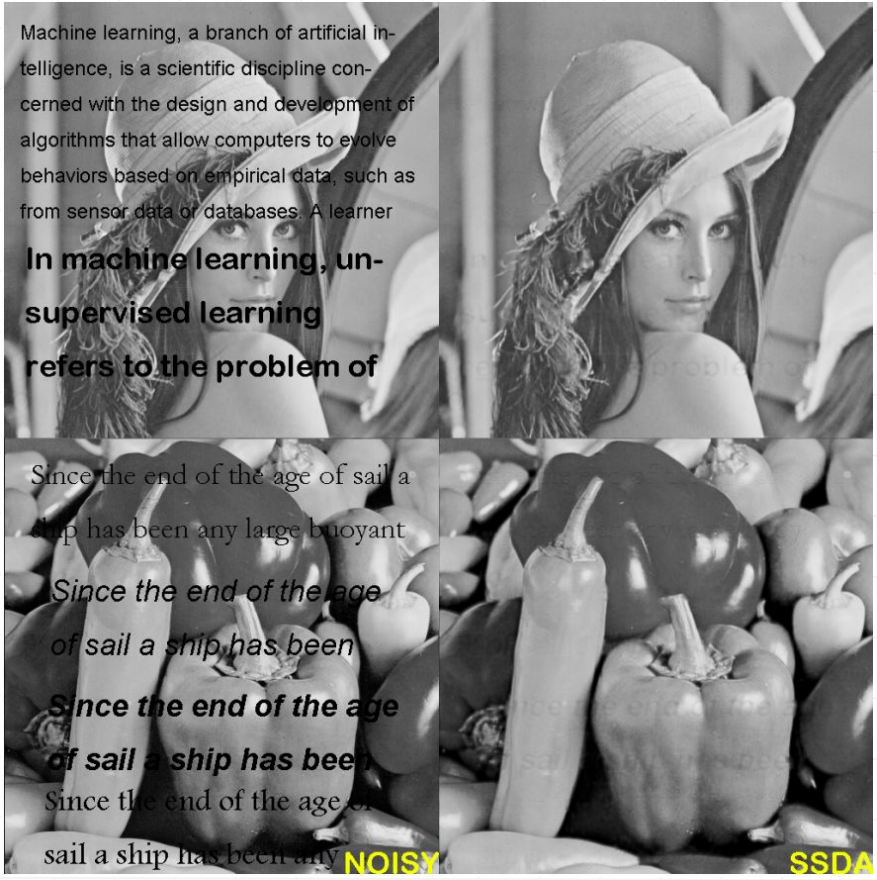
[2] Widely used images commonly referred to as Lena, Barbara, Boat, Pepper, etc. in the image processing community.

Xie, Junyuan, Linli Xu, and Enhong Chen. "Image denoising and inpainting with deep neural networks." *Advances in neural information processing systems* 25 (2012): 341-349.

Resizing an image

https://cimg.eu/greycstoration/demonstration.shtml

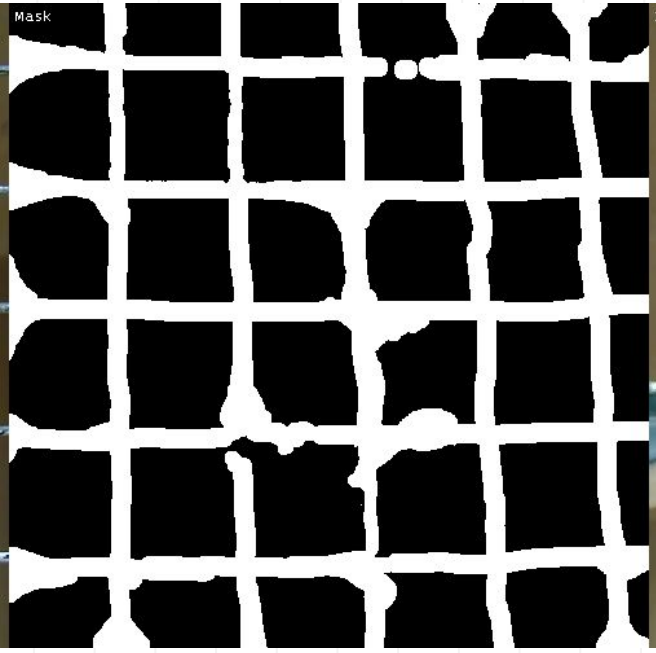
Image inpainting - undo text over image



Xie, Junyuan, Linli Xu, and Enhong Chen. "Image denoising and inpainting with deep neural networks." *Advances in neural information processing systems* 25 (2012): 341-349.

Image inpainting - free a bird

https://cimg.eu/greycstoration/demonstration.shtml



Linear and nonlinear component analysis

Probabilistic PCA

Autoencoders

Applications in Image processing -- denoising, upscaling, impainting

Recommender systems

Relational data and recommender systems

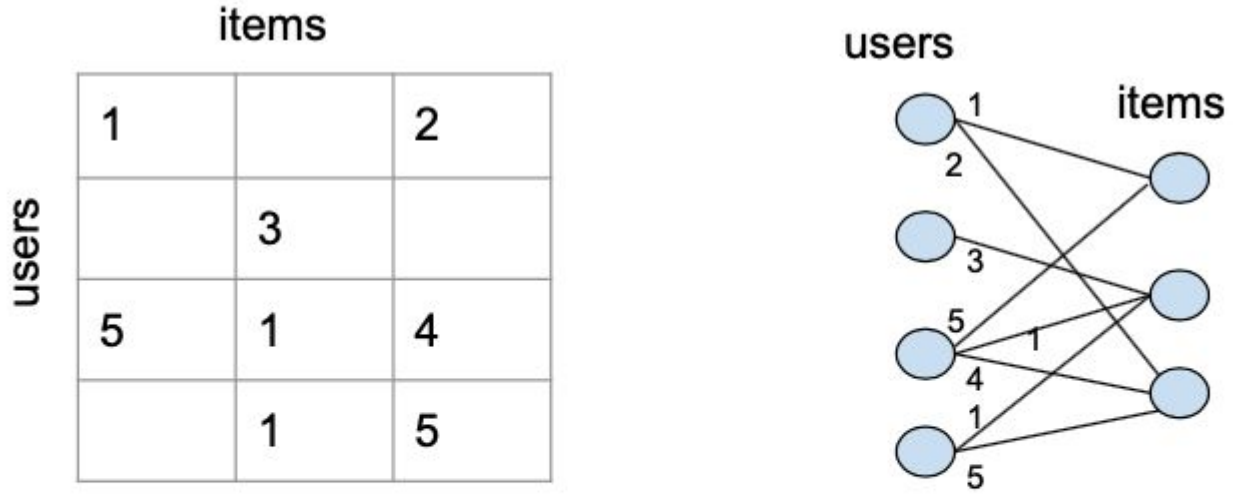


Figure 22.1: Example of a relational dataset represented as a sparse matrix (left) or a sparse bipartite graph (right). Values corresponding to empty cells (missing edges) are unknown. Rows 3 and 4 are similar to each other, indicating that users 3 and 4 might have similar preferences, so we can use the data from user 3 to predict user 4's preferences. However, user 1 seems quite different in their preferences, and seems to give low ratings to all items. For user 2, we have very little observed data, so it is hard to make reliable predictions.

	MovieID	Title	Genres
36	858	Godfather, The (1972)	Action|Crime|Drama
35	1387	Jaws (1975)	Action|Horror
65	2028	Saving Private Ryan (1998)	Action|Drama|War
63	1221	Godfather: Part II, The (1974)	Action|Crime|Drama
11	913	Maltese Falcon, The (1941)	Film-Noir|Mystery
20	3417	Crimson Pirate, The (1952)	Adventure|Comedy|Sci-Fi
34	2186	Strangers on a Train (1951)	Film-Noir|Thriller
55	2791	Airplane! (1980)	Comedy
31	1188	Strictly Ballroom (1992)	Comedy|Romance
28	1304	Butch Cassidy and the Sundance Kid (1969)	Action|Comedy|Western

(a)

	MovieID	Title	Genres
516	527	Schindler's List (1993)	Drama|War
1848	1953	French Connection, The (1971)	Action|Crime|Drama|Thriller
596	608	Fargo (1996)	Crime|Drama|Thriller
1235	1284	Big Sleep, The (1946)	Film-Noir|Mystery
2085	2194	Untouchables, The (1987)	Action|Crime|Drama
1188	1230	Annie Hall (1977)	Comedy|Romance
1198	1242	Glory (1989)	Action|Drama|War
897	922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	Film-Noir
1849	1954	Rocky (1976)	Action|Drama
581	593	Silence of the Lambs, The (1991)	Drama|Thriller

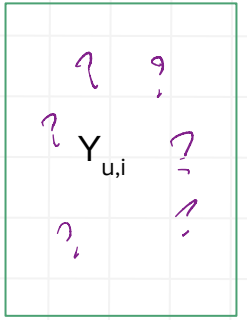
(b)

Figure 22.4: (a) Top 10 movies (from a list of 69) that user "837" has already highly rated. (b) Top 10 predictions (from a list of 3637) from the algorithm. Generated by code at figures.probml.ai/book1/22.4.

Collaborative filtering

Netflix prize dataset: 480K+ users, rating 17.8K movies, 100M ratings total
Newer public datasets:
MovieLens 1 M - 6040 users, 3760 movies, 1M ratings

MovieLens 10 M - ~70k users, 10K movies, ~10M ratings



user u
item i

observed rating $y_{u,i}$
estimate unknown rating $\hat{y}_{u,i}$

$$\hat{Y}_{ui} = \sum_{u':Y_{u',i} \neq ?} \text{sim}(u, u') \, Y_{u',i} \qquad (22.1)$$

can come from y
can have other use profile info.

Matrix completion / matrix factorisation

Loss function

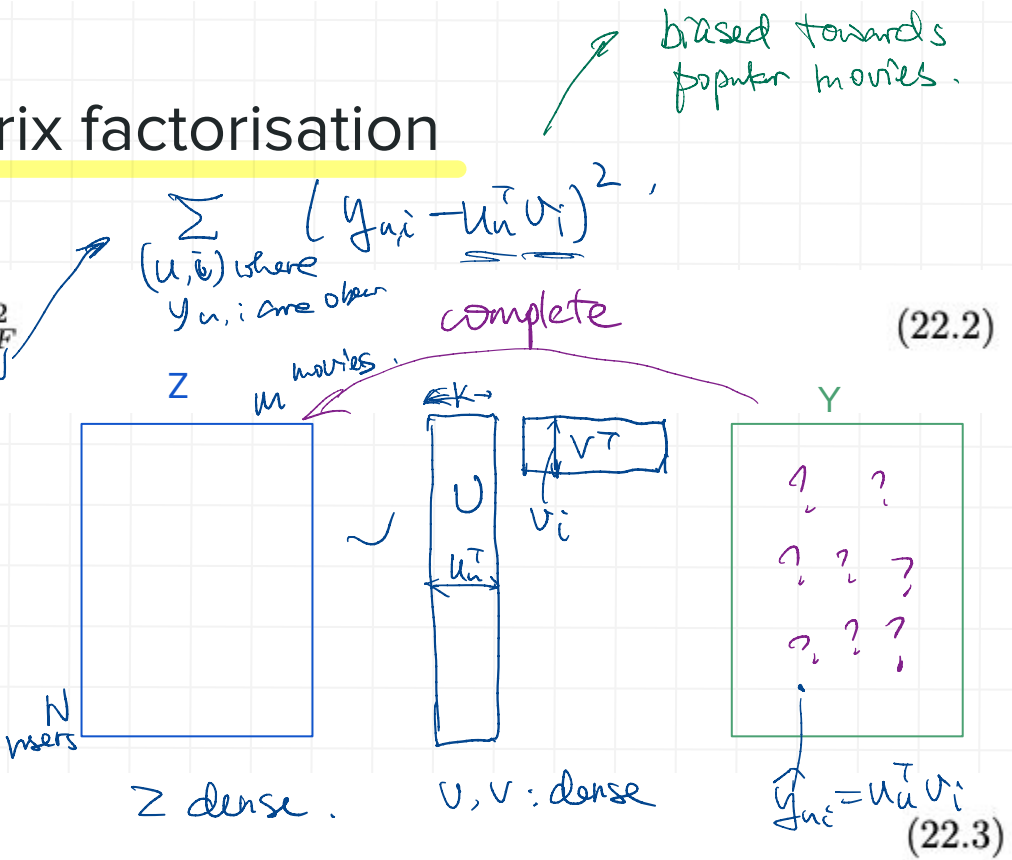
$$\mathcal{L}(\mathbf{Z}) = \sum_{ij:Y_{ij}\neq ?} (Z_{ij} - Y_{ij})^2 = ||\mathbf{Z} - \mathbf{Y}||_F^2 \quad (22.2)$$

"model"

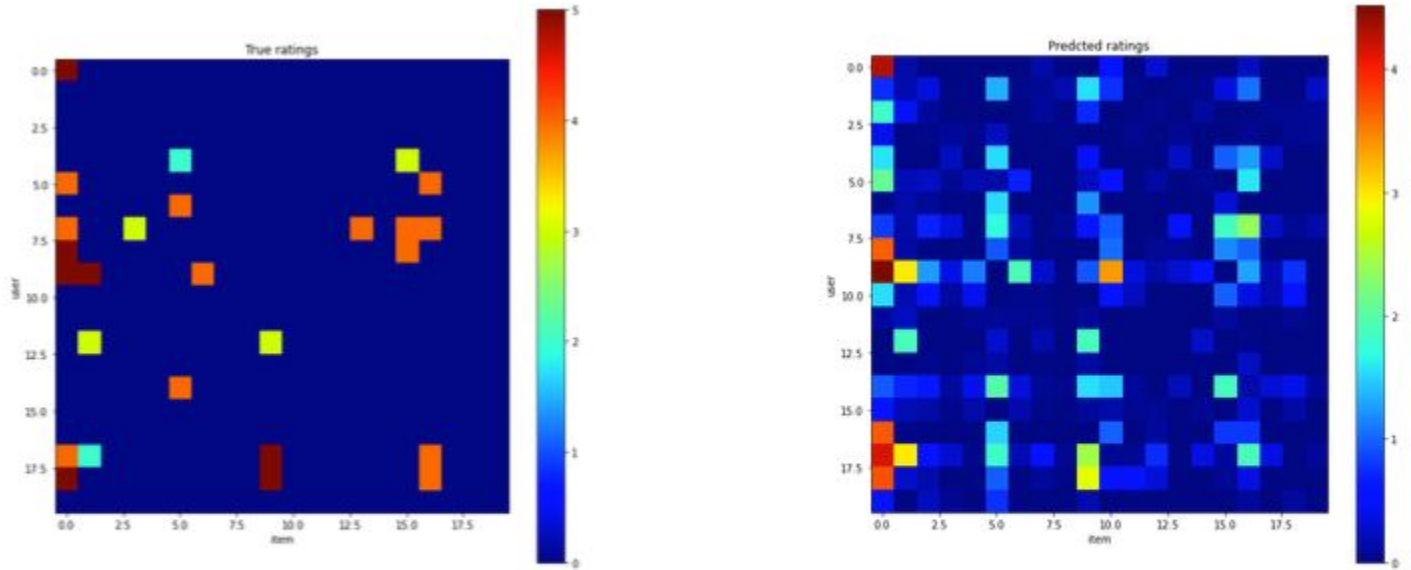
$$\mathbf{Z} = \mathbf{U}\mathbf{V}^\top \approx \mathbf{Y}$$

prediction

$$\hat{y}_{ui} = \mathbf{u}_u^\top \mathbf{v}_i \quad (22.3)$$

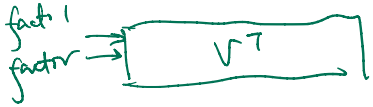


Optimisation: alternating least squares, or stochastic gradient descent (SGD)



(a) (b)

Figure 22.3: (a) A fragment of the observed ratings matrix from the MovieLens-1M dataset. (b) Predictions using SVD with 50 latent components. Generated by code at figures.probml.ai/book1/22.3.



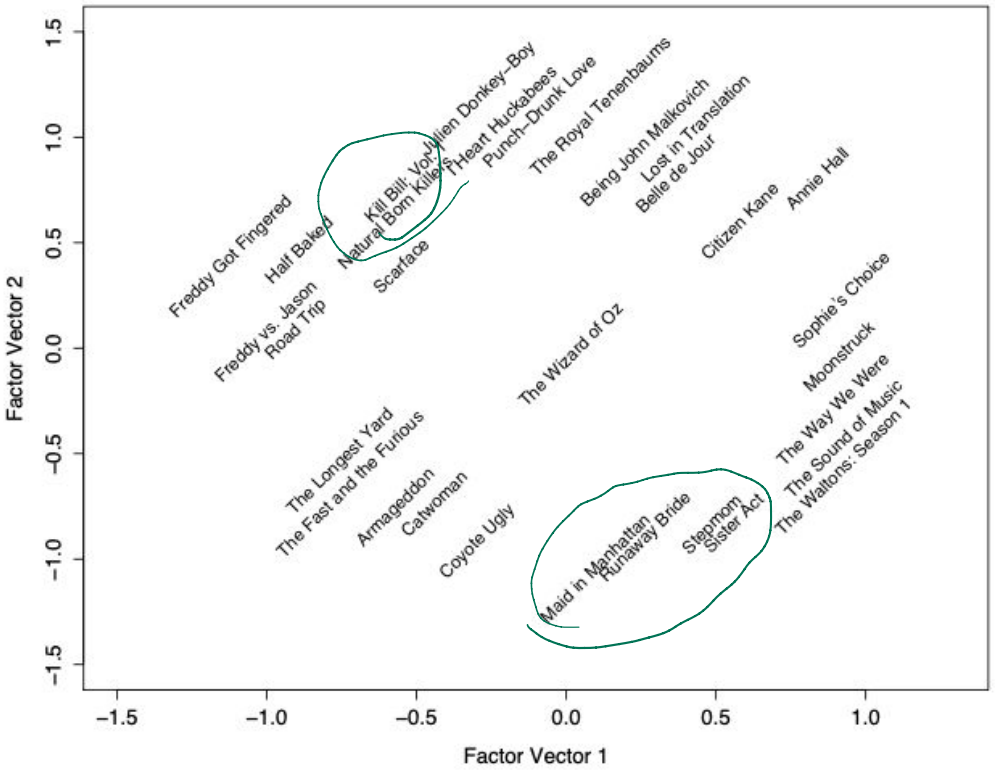


Figure 22.2: Visualization of the first two latent movie factors estimated from the Netflix challenge data. Each movie j is plotted at the location specified by $\mathbf{v}_j$. See text for details. From Figure 3 of [KBV09]. Used with kind permission of Yehuda Koren.

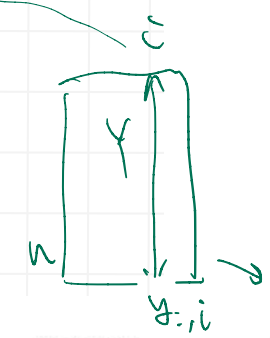
Autoencoders for recommendation

[Sedhain et al, 2015]

"model"

prediction

$$f(\mathbf{y}_{:,i};\boldsymbol{\theta}) = \mathbf{W}^\top \varphi(\mathbf{V}\mathbf{y}_{:,i} + \boldsymbol{\mu}) + \mathbf{b}$$



Loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \sum_{u:y_{ui}\neq ?} (y_{u,i} - f(\mathbf{y}_{:,i};\boldsymbol{\theta})_u)^2 + \frac{\lambda}{2}(||\mathbf{W}||_F^2 + ||\mathbf{V}||_F^2)$$

square loss

L2 regularizer

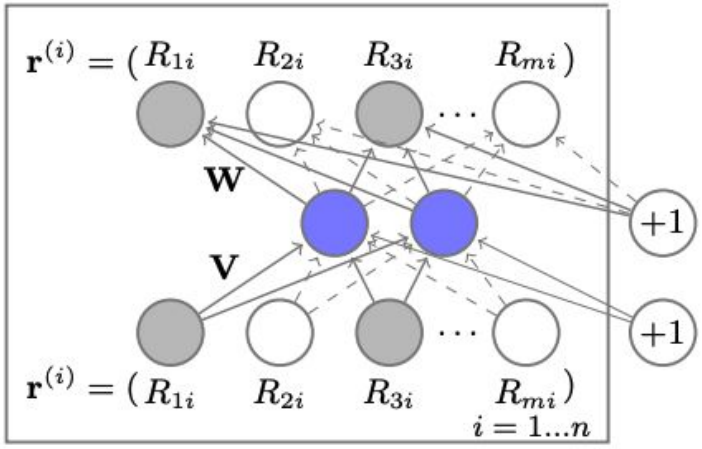


Figure 1: Item-based AutoRec model. We use plate notation to indicate that there are n copies of the neural network (one for each item), where $\mathbf{W}$ and $\mathbf{V}$ are tied across all copies.

AutoRec: Autoencoders Meet Collaborative Filtering

Suvash Sedhain[†*], Aditya Krishna Menon[†*], Scott Sanner[†*], Lexing Xie[*†]
[†] NICTA, [*] Australian National University
suvash.sedhain@anu.edu.au, { aditya.menon, scott.sanner }@nicta.com.au,
lexing.xie@anu.edu.au

ABSTRACT

This paper proposes AutoRec, a novel autoencoder framework for collaborative filtering (CF). Empirically, AutoRec's compact and efficiently trainable model outperforms state-of-the-art CF techniques (biased matrix factorization, RBM-CF and LLORMA) on the Movielens and Netflix datasets.

Categories and Subject Descriptors D.2.8 [**Information Storage and Retrieval**]Information Filtering

Keywords Recommender Systems; Collaborative Filtering; Autoencoders

1. INTRODUCTION

Collaborative filtering (CF) models aim to exploit information about users' preferences for items (e.g. star ratings) to provide personalised recommendations. Owing to the Netflix challenge, a panoply of different CF models have been proposed, with popular choices being matrix factorisation [1, 2] and neighbourhood models [5]. This paper proposes *AutoRec*, a new CF model based on the autoencoder paradigm; our interest in this paradigm stems from the recent successes of (deep) neural network models for vision and speech tasks. We argue that AutoRec has representational and computational advantages over existing neural approaches to CF [4], and demonstrate empirically that it outperforms the current state-of-the-art methods.

2. THE AUTOREC MODEL

In rating-based collaborative filtering, we have m users, n items, and a partially observed user-item rating matrix $R \in \mathbb{R}^{m \times n}$. Each user $u \in U = \{1 \dots m\}$ can be represented by a partially observed vector $\mathbf{r}^{(u)} = (R_{u1}, \dots R_{un}) \in \mathbb{R}^n$. Similarly, each item $i \in I = \{1 \dots n\}$ can be represented by a partially observed vector $\mathbf{r}^{(i)} = (R_{1i}, \dots R_{mi}) \in \mathbb{R}^m$. Our aim in this work is to design an item-based (user-based) autoencoder which can take as input each partially observed $\mathbf{r}^{(i)}$ ($\mathbf{r}^{(u)}$), project it into a low-dimensional latent (hidden) space, and then reconstruct $\mathbf{r}^{(i)}$ ($\mathbf{r}^{(u)}$) in the output space to predict missing ratings for purposes of recommendation.

Formally, given a set $\mathbf{S}$ of vectors in $\mathbb{R}^d$, and some $k \in \mathbb{N}_+$, an autoencoder solves

$$\min_{\theta} \sum_{\mathbf{r} \in \mathbf{S}} ||\mathbf{r} - h(\mathbf{r}; \theta)||_2^2, \qquad (1)$$

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742726 .

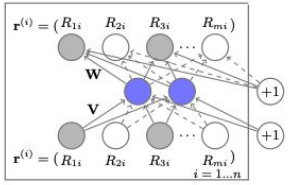


Figure 1: Item-based AutoRec model. We use plate notation to indicate that there are n copies of the neural network (one for each item), where $\mathbf{W}$ and $\mathbf{V}$ are tied across all copies.

where $h(\mathbf{r}; \theta)$ is the *reconstruction* of input $\mathbf{r} \in \mathbb{R}^d$,

$$h(\mathbf{r}; \theta) = f\left(\mathbf{W} \cdot g(\mathbf{V}\mathbf{r} + \boldsymbol{\mu}) + \mathbf{b}\right)$$

for *activation functions* $f(\cdot), g(\cdot)$. Here, $\theta = \{\mathbf{W}, \mathbf{V}, \boldsymbol{\mu}, b\}$ for transformations $\mathbf{W} \in \mathbb{R}^{d \times k}$, $\mathbf{V} \in \mathbb{R}^{k \times d}$, and biases $\mu \in \mathbb{R}^k, \mathbf{b} \in \mathbb{R}^d$. This objective corresponds to an auto-associative neural network with a single, k-dimensional hidden layer. The parameters θ are learned using backpropagation.

The item-based AutoRec model, shown in Figure 1, applies an autoencoder as per Equation 1 to the set of vectors $\{\mathbf{r}^{(i)}\}_{i=1}^n$, with two important changes. First, we account for the fact that each $\mathbf{r}^{(i)}$ is partially observed by only updating during backpropagation those weights that are associated with observed inputs, as is common in matrix factorisation and RBM approaches. Second, we regularise the learned parameters so as to prevent overfitting on the observed ratings. Formally, the objective function for the Item-based AutoRec (I-AutoRec) model is, for regularisation strength $\lambda > 0$,

$$\min_{\theta} \sum_{i=1}^{n} ||\mathbf{r}^{(i)} - h(\mathbf{r}^{(i)}; \theta))||_{\mathcal{O}}^2 + \frac{\lambda}{2} \cdot (||\mathbf{W}||_F^2 + ||\mathbf{V}||_F^2), \qquad (2)$$

where $||\cdot||_{\mathcal{O}}^2$ means that we only consider the contribution of observed ratings. User-based AutoRec (U-AutoRec) is derived by working with $\{\mathbf{r}^{(u)}\}_{u=1}^m$. In total, I-AutoRec requires the estimation of $2mk + m + k$ parameters. Given learned parameters $\hat{\theta}$, I-AutoRec's predicted rating for user u and item i is

$$\hat{R}_{ui} = (h(\mathbf{r}^{(i)}; \hat{\theta}))_u. \qquad (3)$$

Figure 1 illustrates the model, with shaded nodes corresponding to observed ratings, and solid connections corresponding to weights that are updated for the input $\mathbf{r}^{(i)}$.

	ML-1M	ML-10M
U-RBM	0.881	0.823
I-RBM	0.854	0.825
U-AutoRec	0.874	0.867
I-AutoRec	**0.831**	**0.782**

(a)

$f(\cdot)$	$g(\cdot)$	RMSE
Identity	Identity	0.872
Sigmoid	Identity	0.852
Identity	Sigmoid	**0.831**
Sigmoid	Sigmoid	0.836

(b)

	ML-1M	ML-10M	Netflix
BiasedMF	0.845	0.803	0.844
I-RBM	0.854	0.825	-
U-RBM	0.881	0.823	0.845
LLORMA	0.833	**0.782**	0.834
I-AutoRec	**0.831**	**0.782**	**0.823**

(c)

Table 1: (a) Comparison of the RMSE of I/U-AutoRec and RBM models. (b) RMSE for I-AutoRec with choices of linear and nonlinear activation functions, Movielens 1M dataset. (c) Comparison of I-AutoRec with baselines on MovieLens and Netflix datasets. We remark that I-RBM did not converge after one week of training. LLORMA's performance is taken from [2].

AutoRec is distinct to existing CF approaches. Compared to the RBM-based CF model (RBM-CF) [4], there are several differences. First, RBM-CF proposes a generative, probabilistic model based on restricted Boltzmann machines, while AutoRec is a discriminative model based on autoencoders. Second, RBM-CF estimates parameters by maximising log likelihood, while AutoRec directly minimises RMSE, the canonical performance in rating prediction tasks. Third, training RBM-CF requires the use of contrastive divergence, whereas training AutoRec requires the comparatively faster gradient-based backpropagation. Finally, RBM-CF is only applicable for discrete ratings, and estimates a separate set of parameters for each rating value. For r possible ratings, this implies nkr or (mkr) parameters for user- (item-) based RBM. AutoRec is agnostic to r and hence requires fewer parameters. Fewer parameters enables AutoRec to have less memory footprint and less prone to overfitting. Compared to matrix factorisation (MF) approaches, which embed both users and items into a shared latent space, the item-based AutoRec model only embeds items into latent space. Further, while MF learns a linear latent representation, AutoRec can learn a *nonlinear* latent representation through activation function $g(\cdot)$.

3. EXPERIMENTAL EVALUATION

In this section, we evaluate and compare AutoRec with RBM-CF [4], Biased Matrix Factorisation [1] (BiasedMF), and Local Low-Rank Matrix Factorisation (LLORMA) [2] on the Movielens 1M, 10M and Netflix datasets. Following [2], we use a default rating of 3 for test users or items without training observations. We split the data into random 90%–10% train-test sets, and hold out 10% of the training set for hyperparamater tuning. We repeat this splitting procedure 5 times and report average RMSE. 95% confidence intervals on RMSE were ± 0.003 or less in each experiment. For all baselines, we tuned the regularisation strength $\lambda \in \{0.001, 0.01, 0.1, 1, 100, 1000\}$ and the appropriate latent dimension $k \in \{10, 20, 40, 80, 100, 200, 300, 400, 500\}$.

A challenge training autoencoders is non-convexity of the objective. We found resilient propagation (RProp) [3] to give comparable performance to L-BFGS, while being much faster. Thus, we use RProp for all subsequent experiments:

Which is better, item- or user-based autoencoding with RBMs or AutoRec? Table 1a shows item-based (I-) methods for RBM and AutoRec generally perform better; this is likely since the average number of ratings per item is much more than those per user; high variance in the number of user ratings leads to less reliable prediction for user-based methods. I-AutoRec outperforms all RBM variants.

How does AutoRec performance vary with linear and nonlinear activation functions $f(\cdot), g(\cdot)$? Table 1b indicates that nonlinearity in the *hidden* layer (via $g(\cdot)$) is critical for good performance of I-AutoRec, indicating its potential advantage over MF methods. Replacing sigmoids with Rectified Linear Units (ReLU) performed worse. All other AutoRec experiments use identity $f(\cdot)$ and sigmoid $g(\cdot)$ functions.

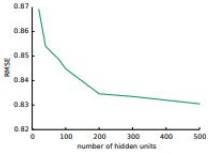


Figure 2: RMSE of I-AutoRec on Movielens 1M as the number of hidden units k varies.

How does performance of AutoRec vary with the number of hidden units? In Figure 2, we evaluate the performance of AutoRec model as the number of hidden units varies.We note that performance steadily increases with the number of hidden units, but with diminishing returns. All other AutoRec experiments use $k = 500$.

How does AutoRec perform against all baselines? Table 1c shows that AutoRec consistently outperforms all baselines, except for comparable results with LLORMA on Movielens 10M. Competitive performance with LLORMA is of interest, as the latter involves *weighting 50 different local matrix factorization models*, whereas AutoRec only uses a single latent representation via a neural net autoencoder.

Do deep extensions of AutoRec help? We developed a deep version of I-AutoRec with three hidden layers of (500, 250, 500) units, each with a sigmoid activation. We used greedy pretraining and then fine-tuned by gradient descent. On Movielens 1M, RMSE reduces from 0.831 to 0.827 indicating potential for further improvement via deep AutoRec.

Acknowledgments NICTA is funded by the Australian Government as represented by the Dept. of Communications and the ARC through the ICT Centre of Excellence program. This research was supported in part by ARC DP140102185.

References

[1] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42, 2009.
[2] J. Lee, S. Kim, G. Lebanon, and Y. Singer. Local low-rank matrix approximation. In *ICML*, 2013.
[3] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, 1993.
[4] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In *ICML*, 2007.
[5] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

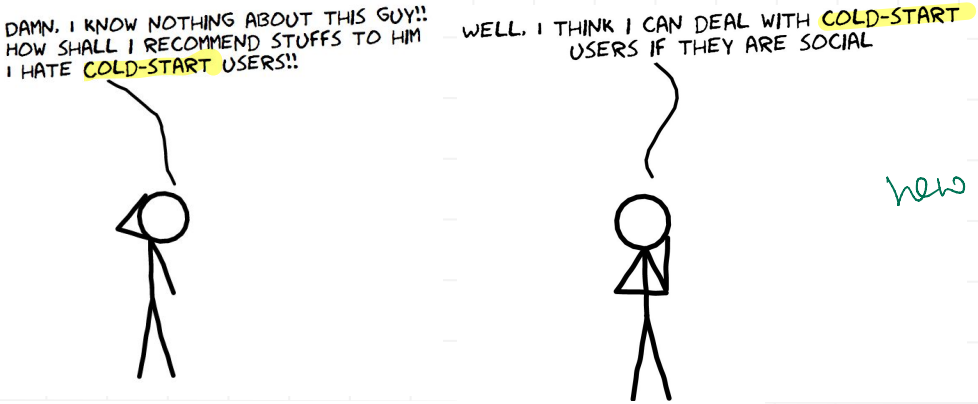
A fireside chat with Dr Suvash Sedhain and Dr Xi Yang from Twitter Engineering

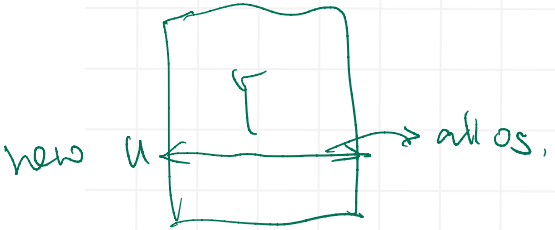
https://cecs.anu.edu.au/events/event-series/anu-computing-leadership-seminar-series

Social recommendation

Social Collaborative Filtering for Cold-start Recommendations. Sedhain, Suvash, Sanner, Scott, Braziunas, Darius, and Xie, Lexing, *Recsys* 2014

[credit: Suvash Sedhain for poster comic]





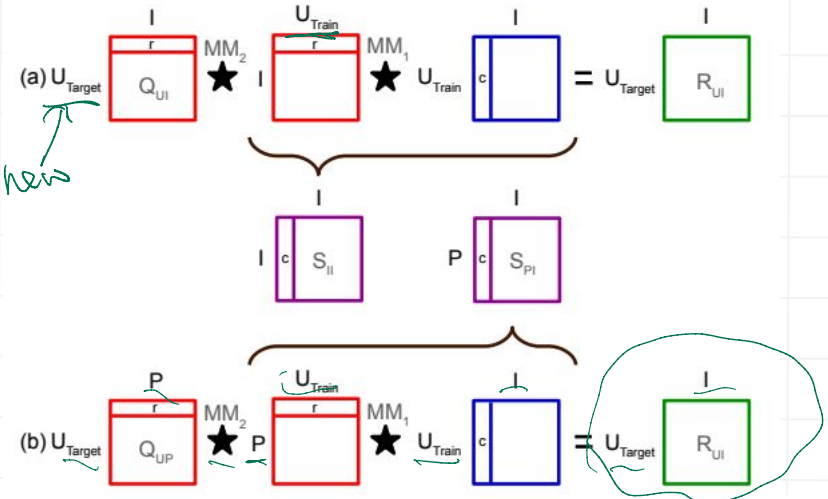


Figure 1: (a) A matrix algebra view of standard item-item neighborhood-based recommendation and (b) a variation for social cold-start recommendation that instead uses page-item similarity and does not require item purchase information for target users. U represents the user dimension (subdivided into train and test users), I the item dimension, and P the user's personal information dimension (demographics, friends, or page likes). The ★ operators annotated by MM_1 and MM_2 denote *generalized* matrix multiplication, permitting *any* similarity metric over two vectors in place of the usual inner product.

References

- Yoshua Bengio, "Learning Deep Architectures for AI", Foundations and Trends in Machine Learning, 2009
- `http://deeplearning.net/tutorial/`
- `http://www.deeplearningbook.org/contents/autoencoders.html`
- Fuchs, "On Sparse Representations in Arbitrary Redundant Bases", IEEE Trans. Info. Theory, 2004
- Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks", 2010.

Linear and Nonlinear Component Analysis

Probabilistic PCA

Autoencoders

Autoencoders for image processing

Recommender systems