

From PCA to Autoencoders

Unsupervised Representation Learning

FLORENT FOREST

PhD student in Data science & Machine learning

Safran Aircraft Engines — Université Paris 13

✉ forest@lipn.univ-paris13.fr

🌐 <http://florentfo.rest>

🐙 FlorentF9

December 17, 2019

Table of contents

1. Introduction to autoencoders

- Motivations

- Definition

- Mathematical formulation

2. Links with Principal Component Analysis (PCA)

- Can neurons learn principal components?

- Linear autoencoders

3. Real-life autoencoders

- Non-linear and deep autoencoders

- Different types of regularizations

- Variational autoencoders (VAE)

4. Applications

Introduction to autoencoders

Introduction to autoencoders

Motivations

Where we are

- Complex and high-dimensional data (i.e. $d \sim [10^3, 10^6]$)
- "Big Data" (i.e. $N \sim [10^5, 10^9]$)
- No labels (unsupervised learning)

What we have seen so far

- Linear dimensionality reduction techniques (e.g. PCA)
→ cannot learn complex transformations
- Non-linear techniques, e.g. t-SNE
→ not scalable, time complexity is $\mathcal{O}(d \cdot N^2)$, or
 $\mathcal{O}(d \cdot N \log N)$ with Barnes-Hut (but limited to output
dimension ≤ 3)

So why not use (deep) neural networks?

- Can learn complex transformations
- Comfortable in very-high-dimensional spaces
- Scale linearly with the size of data

Introduction to autoencoders

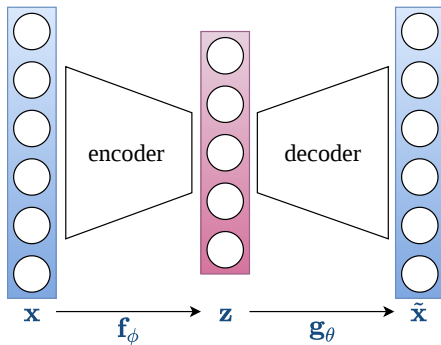
Definition

Definition

Definition

An **autoencoder** is a neural network trained to **reconstruct its inputs**. It is composed of two components:

1. an **encoder**, mapping the input to a latent representation ("code") $\mathbf{z} = \mathbf{f}_\phi(\mathbf{x})$
2. a **decoder**, mapping the code back to the input space $\tilde{\mathbf{x}} = \mathbf{g}_\theta(\mathbf{z})$



Challenge

We do not want the encoder to learn the identity function, but to learn a *good representation* of our data.

Regularization?

Reducing the size of the *hypothesis set* \mathcal{H} by constraining the space of possible solutions to the optimization problem.

- L2 weight decay
- Sparsity, L1 weight decay
- ...

Introduction to autoencoders

Mathematical formulation

What is a good representation?

Let $q_\phi(Z|X)$ be a (stochastic) parametric mapping from X to Z . A good representation Z of a random variable X maximizes **mutual information** between X and Z (*infomax principle*):

$$\begin{aligned}\mathbb{I}(X; Z) &= \mathbb{H}(X) - \mathbb{H}(X|Z) \\ &= C(X) + \mathbb{E}_{q_\phi(X, Z)} [\log q_\phi(X|Z)]\end{aligned}$$

For any parametric distribution $p_\theta(X|Z)$ we have

$$\mathbb{E}_{q_\phi(X, Z)} [\log p_\theta(X|Z)] \leq \mathbb{E}_{q_\phi(X, Z)} [\log q_\phi(X|Z)] \text{ (using } D_{KL}(q||p) \geq 0\text{)}.$$

Task: maximize a lower bound on $\mathbb{I}(X; Z)$

$$\underset{\phi, \theta}{\text{maximize}} \mathbb{E}_{q_\phi(X, Z)} [\log p_\theta(X|Z)]$$

Loss function for deterministic autoencoders

We consider **deterministic** mappings $Z = \mathbf{f}_\phi(X)$ (i.e. $q_\phi(Z|X) = \delta(Z - \mathbf{f}_\phi(X))$) and $\tilde{X} = \mathbf{g}_\theta(\mathbf{f}_\phi(X))$.

$$\underset{\phi, \theta}{\text{maximize}} \mathbb{E}_{q_\phi(X)} [\log p_\theta(X|Z = \mathbf{f}_\phi(X))]$$

Using empirical mean over a set of data samples:

$$\underset{\phi, \theta}{\text{maximize}} \sum_i \log p_\theta(\mathbf{x}^{(i)} | \mathbf{z}^{(i)} = \mathbf{f}_\phi(\mathbf{x}^{(i)}))$$

equivalent to:

$$\underset{\phi, \theta}{\text{maximize}} \sum_i \log p(\mathbf{x}^{(i)} | \tilde{\mathbf{x}}^{(i)} = \mathbf{g}_\theta(\mathbf{f}_\phi(\mathbf{x}^{(i)})))$$

Let us turn this into a minimization of the negative sum of individual loss functions $\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) = -\log p(\mathbf{x}|\tilde{\mathbf{x}})$.

Loss function for deterministic autoencoders

The reconstruction $\tilde{\mathbf{x}}$ is the **mean** of a distribution that may have generated \mathbf{x} .

Continuous variables: $\mathbf{x} \in \mathbb{R}^d$

- Gaussian distribution: $X|\tilde{X} = \tilde{\mathbf{x}} \sim \mathcal{N}(\tilde{\mathbf{x}}, \sigma^2 \mathbf{I})$
- Loss function: $\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) \propto \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2$

→ **Mean Squared Error (MSE) loss**

Binary variables: $\mathbf{x} \in \{0, 1\}^d$, or $\mathbf{x} \in [0, 1]^d$

- Bernoulli distribution: $X|\tilde{X} = \tilde{\mathbf{x}} \sim \mathcal{B}(\tilde{\mathbf{x}})$
- Loss function: $-\sum_{j=1}^d [\mathbf{x}_j \log \tilde{\mathbf{x}}_j + (1 - \mathbf{x}_j) \log(1 - \tilde{\mathbf{x}}_j)]$

→ **Cross-entropy loss**

Links with Principal Component Analysis (PCA)

Links with Principal Component Analysis (PCA)

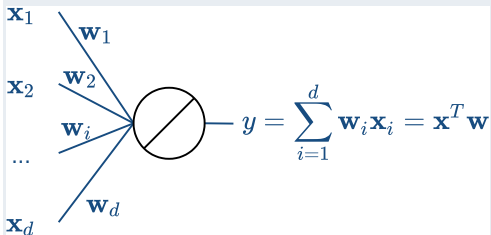
Can neurons learn principal components?

Hebb's learning rule

Back to 1949 → Hebbian learning



The artificial neuron and Hebb's learning rule [5]



Weights increase if input and output are correlated.

$$\Delta \mathbf{w}_i = \alpha x_i y$$

$$\Delta \mathbf{w} = \alpha \mathbf{x} y = \alpha \mathbf{x} \mathbf{x}^T \mathbf{w}$$

See for example Rosenblatt's **perceptron** (1958) [14] for an illustration of this learning rule.

Learning principal components with Oja's rule

Problem: the weights can "explode" with Hebb's rule. A solution is to add a *forgetting term*.

Oja's learning rule [11]

$$\Delta \mathbf{w} = \alpha(\mathbf{x}y - y^2 \mathbf{w})$$

We can show that this rule leads to *principal components* [2, 12]:

$$\Delta \mathbf{w} = \alpha(\mathbf{x}\mathbf{x}^T \mathbf{w} - \mathbf{w}^T \mathbf{x}\mathbf{x}^T \mathbf{w}\mathbf{w})$$

After convergence, we have:

$$\mathbb{E}[\Delta \mathbf{w}] = \alpha(\mathbb{E}[\mathbf{x}\mathbf{x}^T] \mathbf{w} - \mathbf{w}^T \mathbb{E}[\mathbf{x}\mathbf{x}^T] \mathbf{w}\mathbf{w}) = \alpha(\mathbf{C}\mathbf{w} - \mathbf{w}^T \mathbf{C}\mathbf{w}\mathbf{w}) = 0$$

where $\mathbf{C} = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$ is the covariance matrix. Finally:

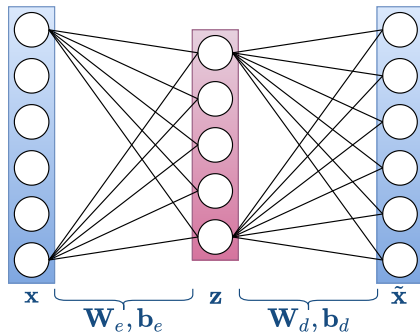
$$\mathbf{C}\mathbf{w} = \mathbf{w}^T \mathbf{C}\mathbf{w}\mathbf{w} = \lambda \mathbf{w}$$

Thus, \mathbf{w} is an eigenvector of the covariance matrix, a.k.a. a **principal component**.

Links with Principal Component Analysis (PCA)

Linear autoencoders

Definition



$$\tilde{\mathbf{x}} = \mathbf{W}_d (\mathbf{W}_e \mathbf{x} + \mathbf{b}_e) + \mathbf{b}_d$$

Ignoring the biases, the MSE loss becomes:

$$\begin{aligned} \text{MSE} &= \sum_i \|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|_2^2 = \sum_i \|\mathbf{x}^{(i)} - \mathbf{W}_d \mathbf{W}_e \mathbf{x}^{(i)}\|_2^2 \\ &= \|\mathbf{X} - \mathbf{X} \mathbf{W}_d \mathbf{W}_e\|_F^2 \end{aligned}$$

Equivalence to PCA

PCA

$$\underset{\mathbf{U} \in \mathbb{R}^{d \times r}}{\text{minimize}} \quad \|\mathbf{X} - \mathbf{X}\mathbf{U}\mathbf{U}^T\|_F^2$$

$$\text{subject to } \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

Linear AE

$$\underset{\mathbf{W}_e \in \mathbb{R}^{l \times d}, \mathbf{W}_d \in \mathbb{R}^{d \times l}}{\text{minimize}} \quad \|\mathbf{X} - \mathbf{X}\mathbf{W}_d\mathbf{W}_e\|_F^2$$

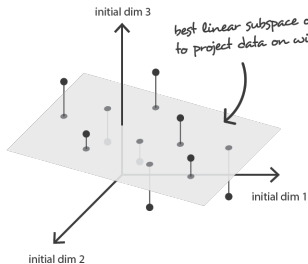
one can show that we must have

$\mathbf{W}_e = \mathbf{W}_d^\dagger$ (pseudo-inverse), then:

$$\underset{\mathbf{W} \in \mathbb{R}^{d \times l}}{\text{minimize}} \quad \|\mathbf{X} - \mathbf{X}\mathbf{W}\mathbf{W}^\dagger\|_F^2$$

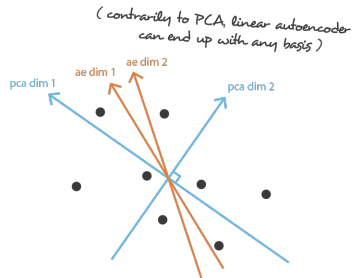
- Linear autoencoders are equivalent to PCA...
without the orthogonality constraint! [13]

Equivalence to PCA



Data in the full initial space

In order to reduce dimensionality, PCA and linear autoencoder target, in theory, the same optimal subspace to project data on...



Data projected on the best linear subspace

... but not necessarily with the same basis due to different constraints
(in PCA the first component is the one that explains the maximum of variance and components are orthogonal)

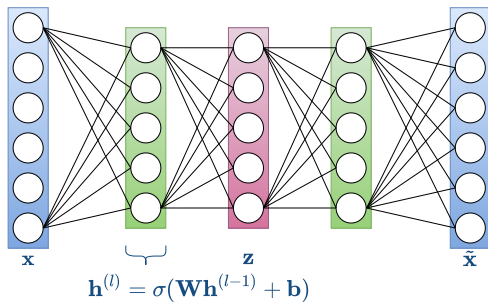
<https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

Real-life autoencoders

Real-life autoencoders

Non-linear and deep autoencoders

Non-linear and deep autoencoders



Sigmoid

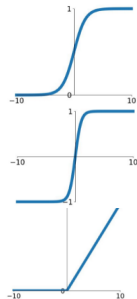
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

tanh

$$\tanh(x)$$

ReLU

$$\max(0, x)$$



► Not equivalent to PCA!

Trained end-to-end with backprop and SGD. Layer-wise pre-training [6] is in fact not necessary (thanks to ReLU, better optimization, etc.).

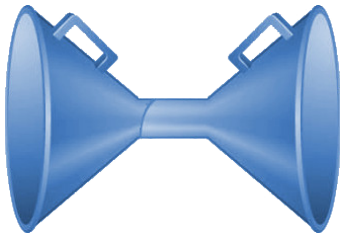
Real-life autoencoders

Different types of regularizations

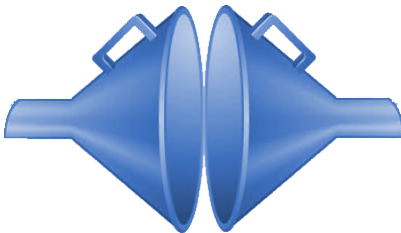
Undercomplete or overcomplete

Let d be the input dimension and l the code dimension.

$l < d$: undercomplete



$l \geq d$: overcomplete



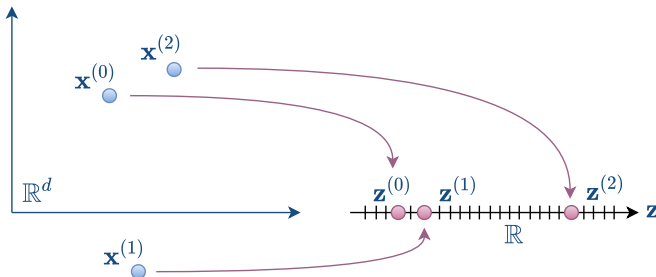
Having the autoencoder encode the data to a lower dimension (in general $l \ll d$), forces it to compress the data and learn an efficient representation and is a **form of regularization**.

colab



The danger of overfitting

Problems: overcomplete AE will not learn useful representations; and even undercomplete AE with a *single continuous latent variable* can remember an entire training set (one real number per sample).



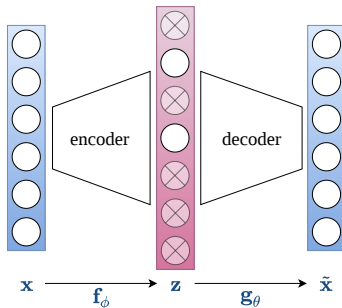
Solution: adding **constraints** to the latent space by using different types of **regularization**!

Sparse autoencoders

Principle

Sparse autoencoders can have more latent units than input units, but **only a few are allowed to activate together**:

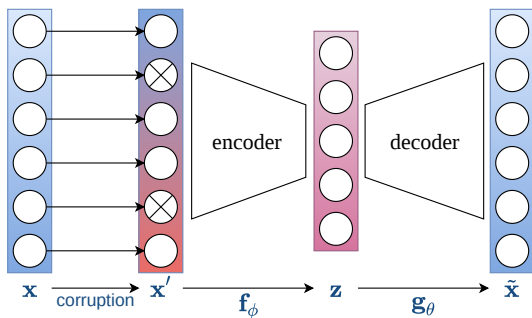
- a fixed proportion (e.g. 5%) using a KL-divergence penalty [10]
- a fixed number k [8]
- using a L1 penalty [1]



Denoising autoencoders

Principle

Denoising autoencoders [15] are trained to reconstruct **corrupted** versions of the input. Corruption can be achieved by adding noise or randomly turning off input units. Usually, they have a single hidden layer with a non-linearity.

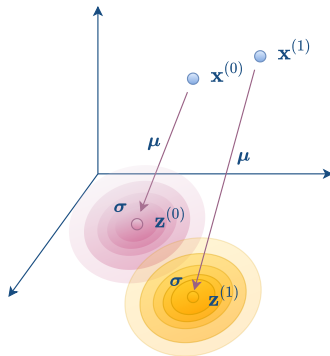
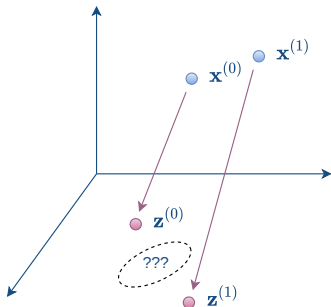


Real-life autoencoders

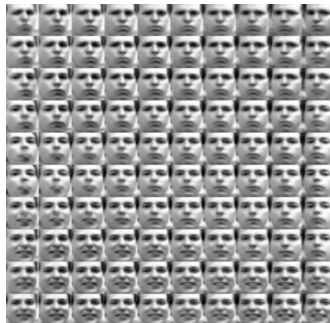
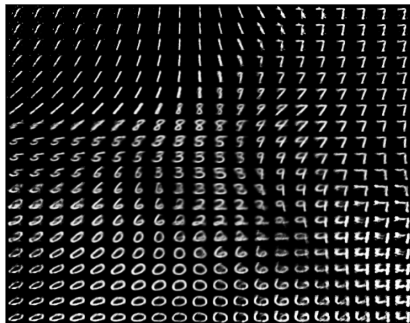
Variational autoencoders (VAE)

Motivations

Limitation of standard AE: the latent space has *no structure* and may not be continuous; it may *overfit*, and we cannot explore it nor *sample* from it.



Motivations



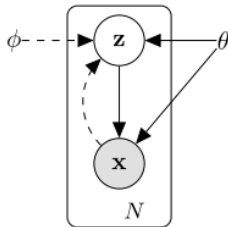
Variational autoencoders (VAE)

VAEs are **latent-variable probabilistic models**.

Probabilistic setting

Generative model $p_{\theta}(\mathbf{x}, \mathbf{z})$:

1. \mathbf{z} is sampled from the **prior** $p_{\theta}(\mathbf{z})$
2. \mathbf{x} is generated with **likelihood** $p_{\theta}(\mathbf{x}|\mathbf{z})$
(*probabilistic decoder*)

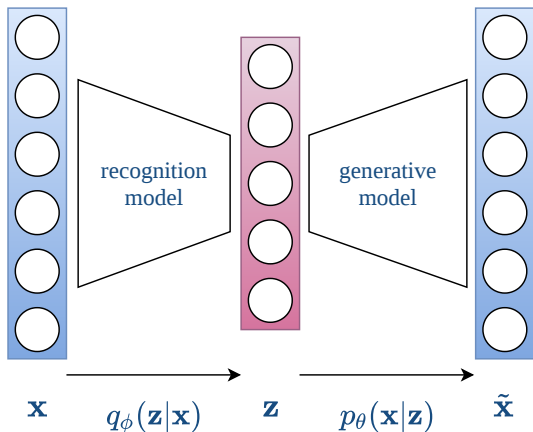


Kingma & Welling 2014 [7]

Problem: θ is a NN, so $p_{\theta}(\mathbf{x})$ and $p_{\theta}(\mathbf{z}|\mathbf{x})$ are intractable.

► Stochastic Gradient Variational Bayes (SGVB) [7] is an efficient estimation method in case of intractable marginal likelihood/posterior and large datasets.

Variational autoencoders (VAE)



Recognition model \rightarrow approximate posterior $q_{\phi}(\mathbf{z}|\mathbf{x})$
(probabilistic encoder)

VAE loss function

VAE ELBO (evidence lower bound)

$$\underset{\phi, \theta}{\text{maximize}} \quad - D_{KL}(q_{\phi}(Z|X) || p_{\theta}(Z)) + \mathbb{E}_{q_{\phi}(Z|X)} [\log p_{\theta}(X|Z)]$$

Key ideas

- The second term is a (negative) **reconstruction error** (e.g. MSE or cross-entropy) as in a deterministic AE.
- The first term, a Kullback-Leibler divergence between $q_{\phi}(Z|X)$ and $p_{\theta}(Z)$, acts as a **regularizer** pushing the encoder distribution closer to the prior distribution (typically a gaussian).

VAE loss function

Let's put gaussians everywhere!

- $p_{\theta}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$
- $q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbf{I})$

The reparameterization trick

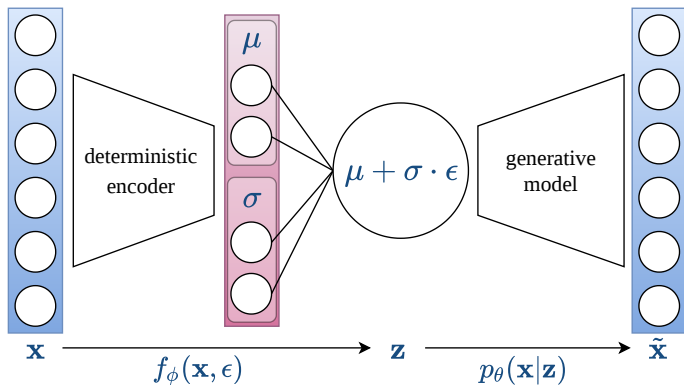
To sample from $q_{\phi}(\mathbf{z}|\mathbf{x})$, we use the reparameterization $\mathbf{z} = f_{\phi}(\mathbf{x}, \epsilon) = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

For a given $\mathbf{x}^{(i)}$, and using 1-sample Monte-Carlo estimation, the ELBO becomes:

$$\frac{1}{2} \sum_j \left(1 + \log(\boldsymbol{\sigma}_j^{(i)})^2 - (\boldsymbol{\mu}_j^{(i)})^2 - (\boldsymbol{\sigma}_j^{(i)})^2 \right) + \log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$$

$$\text{where } \mathbf{z}^{(i)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \cdot \epsilon$$

VAE: the reparameterization trick



colab



Applications

Autoencoders can extract useful low-dimensional representations from high-dimensional data. They can be used as:

- ▶ a pre-processing step for any other ML algorithm (clustering, supervised classification or regression)
- ▶ an unsupervised pre-training procedure for supervised deep neural networks (e.g. stacked AE pre-training)
→ see [6], [4], [15]

Autoencoders can be used as a compression algorithm:

- ▶ lossy
- ▶ data-specific

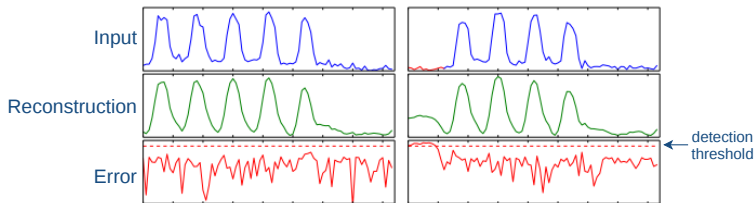
Not commonly used in practice...

The decoder model can be used to generate new data samples:

- ▶ Deterministic AEs: adding noise, interpolating or extrapolating in latent space [3]
- ▶ Generative models (VAE, GAN)

Anomaly detection

When the input differs from the training distribution (e.g. an outlier), the model will produce a large reconstruction error. This error can be used to **score anomalies** [9].



(Malhotra et al, 2016)

Questions?



D. Arpit, Y. Zhou, H. Q. Ngo, and V. Govindaraju.

Why regularized auto-encoders learn sparse representation?

33rd International Conference on Machine Learning, ICML 2016, 1:211–223, 2016.



S. Becker.

Unsupervised Learning Procedures for Neural Networks.

The International Journal of Neural Systems, 1:17–33, 1991.



T. Devries and G. W. Taylor.

Dataset Augmentation in Feature Space.

In ICLR 2017 Workshop, pages 1–12, 2017.



D. Erhan, A. Courville, and P. Vincent.

Why Does Unsupervised Pre-training Help Deep Learning ?

Journal of Machine Learning Research, 11:625–660, 2010.



D. O. Hebb.

The organization of behavior: A neuropsychological theory.

Wiley, June 1949.



G. E. Hinton and R. Salakhutdinov.

Reducing the Dimensionality of Data with Neural Networks.

Science, 313(July):504–507, 2006.



D. P. Kingma and M. Welling.

Stochastic Gradient VB and the Variational Auto-Encoder.

2nd International Conference on Learning Representations (ICLR), pages 1–14, 2014.



A. Makhzani and B. Frey.

k-Sparse Autoencoders.

2013.



P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff.

LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection.

In ICML 2016 Anomaly Detection Workshop, 2016.



A. Ng.

Sparse autoencoder.

Technical report, Stanford University, 2011.



E. Oja.

A Simplified Neuron Model as a Principal Component Analyzer.

Journal of Mathematical Biology, 15(3):267–273, 1982.



E. Oja.

Principal Components, Minor Components, and Linear Neural Networks, 1992.



E. Plaut.

From Principal Subspaces to Principal Components with Linear Autoencoders.

pages 1–6, 2018.



F. Rosenblatt.

The perceptron: A probabilistic model for information storage and organization in the brain.

Psychological Review, pages 65–386, 1958.



P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol.

Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion.

Journal of Machine Learning Research, 11:3371–3408, 2010.