Autoencoders

Unsupervised learning

Unsupervised learning

(no explicit feedback whether the outputs of the system are correct)

Unsupervised learning

What is it about?

Unsupervised learning

What is it about?

- inferring a function to describe hidden structure from unlabelled data

Unsupervised learning

What is it about?

- inferring a function to describe hidden structure from unlabelled data

What are the tasks?

Unsupervised learning

What is it about?

- inferring a function to describe hidden structure from unlabelled data

What are the tasks?

- density estimation

Unsupervised learning

What is it about?

- inferring a function to describe hidden structure from unlabelled data

What are the tasks?

- density estimation
- clustering

Unsupervised learning

What is it about?

- inferring a function to describe hidden structure from unlabelled data

What are the tasks?

- density estimation
- clustering
- feature learning / representation learning

Unsupervised learning

What is it about?

- inferring a function to describe hidden structure from unlabelled data

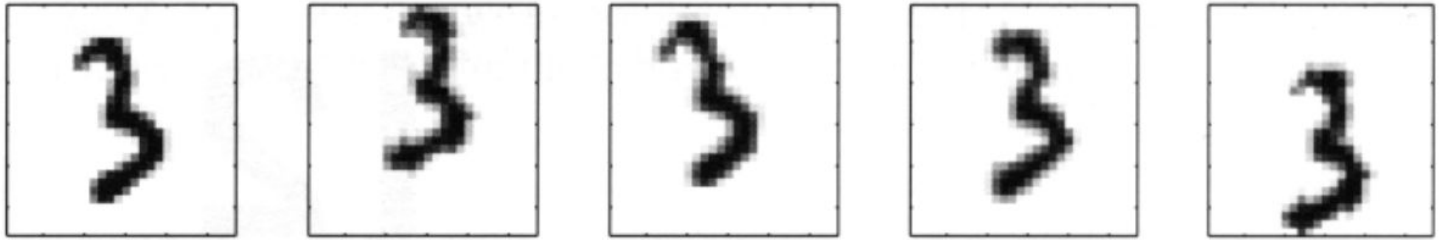
What are the tasks?

- density estimation
- clustering
- feature learning / representation learning
- dimensionality reduction

We will explore models in which the latent variables are continuous.

An important motivation for such models is that many data sets have the property that **the data points all lie close to a manifold of much lower dimensionality than that of the original data space**.

Example



We embed a 64x64 digit in a 100x100 image size varying at random the location and orientation. Each resulting image is represented as a 10k-dim data point.

Across a data set of such images, there are only three degrees of freedom of variability, corresponding to the vertical and horizontal translations and the rotations. Translation and rotation are latent variables.

Bishop, Pattern Recognition and Machine Learning

Say, our high-dimensional data lies in near linear manifold.

So, if we can find that manifold of lower dimensionality, we can project the data onto that manifold and represent it there not losing much information, ‘cause in the directions orthogonal to the manifold there is not much variation in the data.

Say, our high-dimensional data lies in near linear manifold.

So, if we can find that manifold of lower dimensionality, we can project the data onto that manifold and represent it there not losing much information, ‘cause in the directions orthogonal to the manifold there is not much variation in the data.

Efficiently: using PCA.

Non-efficiently: using NN with 1 hidden layer (with linear units).

Say, our high-dimensional data lies in near linear manifold.

So, if we can find that manifold of lower dimensionality, we can project the data onto that manifold and represent it there not losing much information, ‘cause in the directions orthogonal to the manifold there is not much variation in the data.

Efficiently: using PCA.

Non-efficiently: using NN with 1 hidden layer (with linear units).

(though, it still might be computationally efficient with huge amounts of data)

The **advantage** of the latter: generalization to curved manifolds and non-linear representations.

PCA

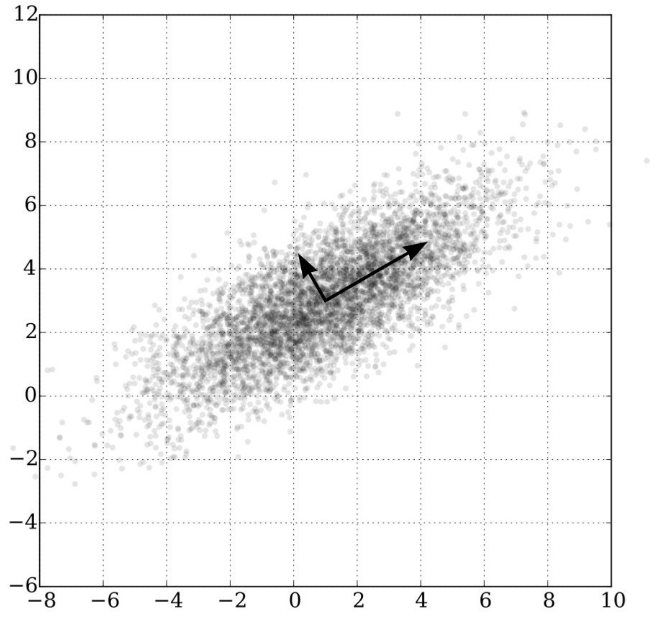
- N-dim data -> M orthogonal directions

capturing most of the variance.

– These M principal directions form a lower-dimensional subspace.

– We can represent an N-dim data point by its projections onto the M principal directions.

– This loses all information about where the data point is located in the remaining orthogonal directions.

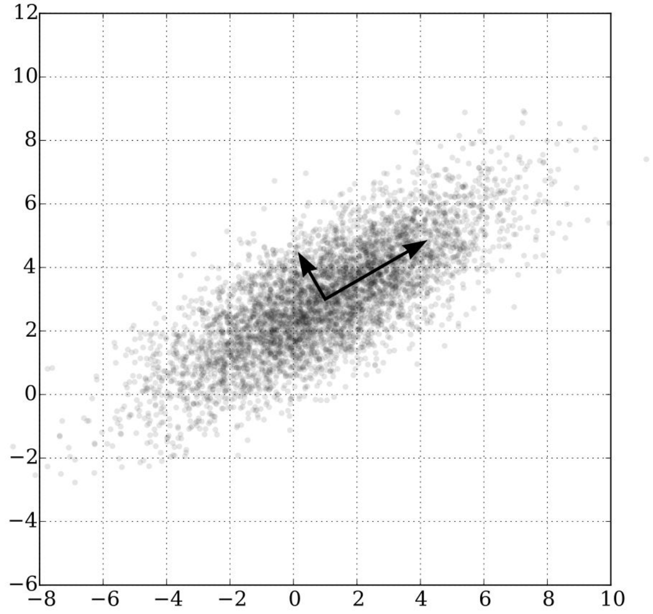


Deep Learning by Geoffrey Hinton, University of Toronto

PCA

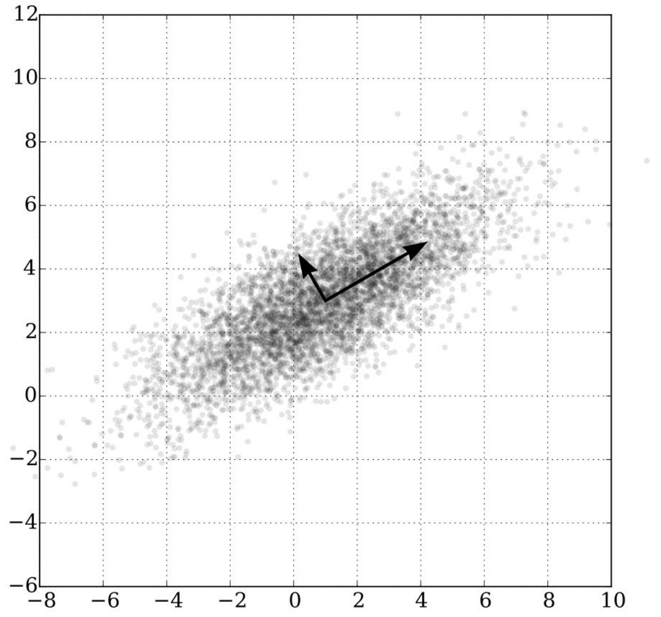
- We reconstruct by using the mean value (over all the data) on the N-M directions that are not represented.

– The reconstruction error is the sum over all these unrepresented directions of the squared differences of the datapoint from the mean.



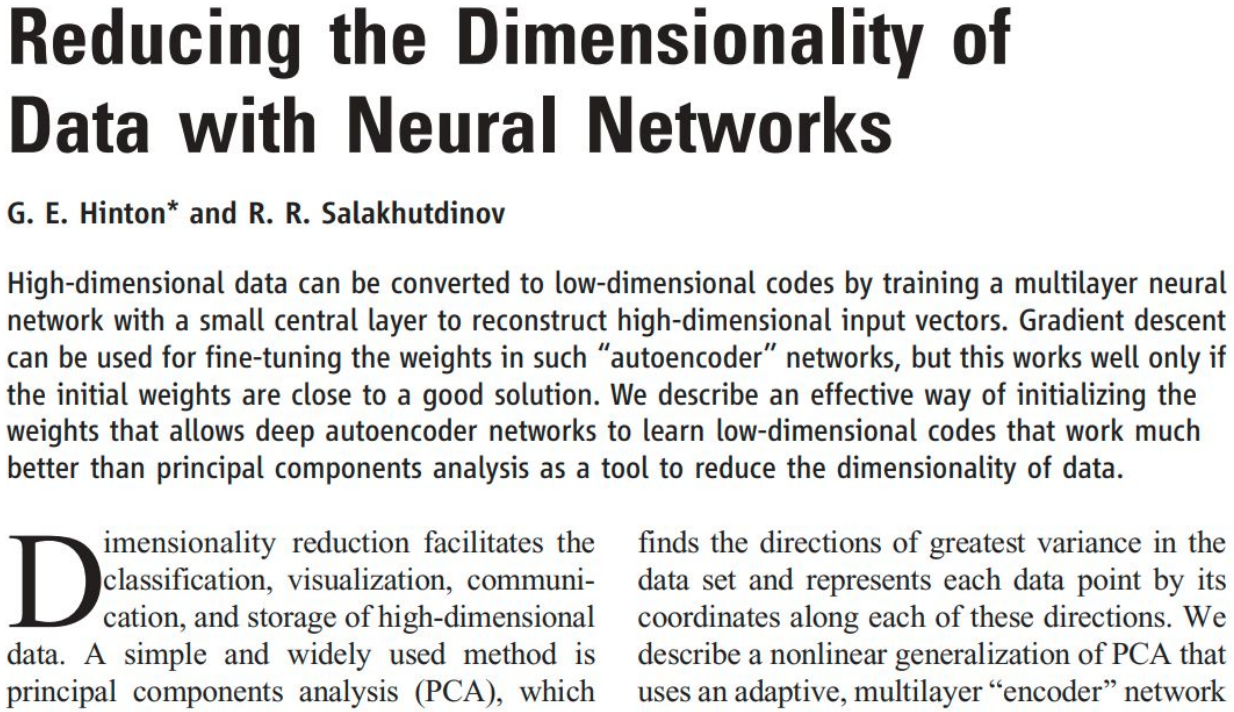
PCA

- We are looking for the orthogonal transformations the “characteristics” of the data become uncorrelated under having largest possible variance, second largest possible variance, and so on.



Deep Learning by Geoffrey Hinton, University of Toronto

Pioneering work



Reducing the Dimensionality of Data with Neural Networks

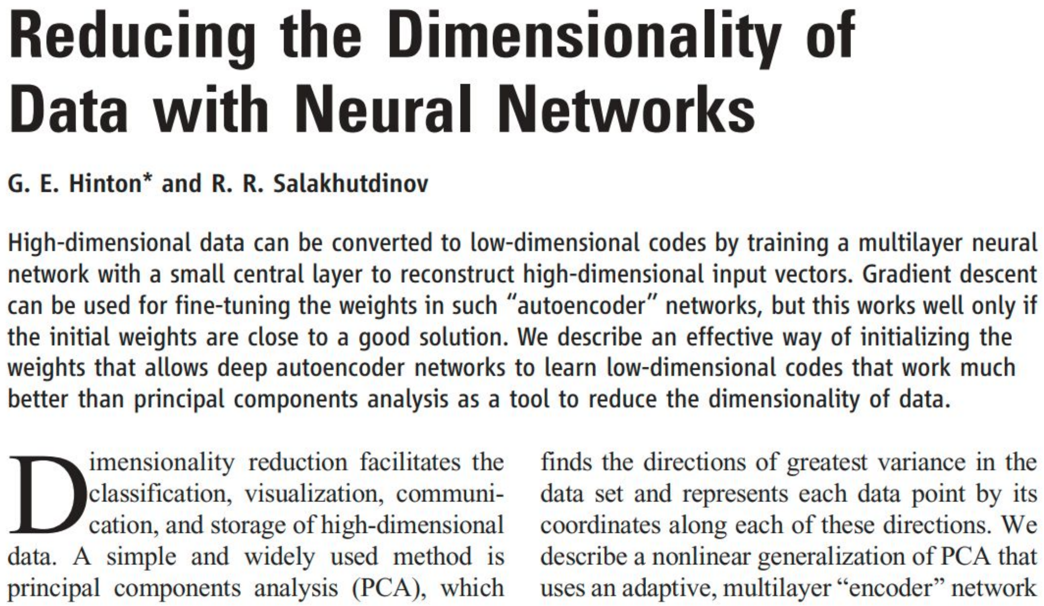
G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer "encoder" network

2006, report in Science, Vol. 313

Pioneering work

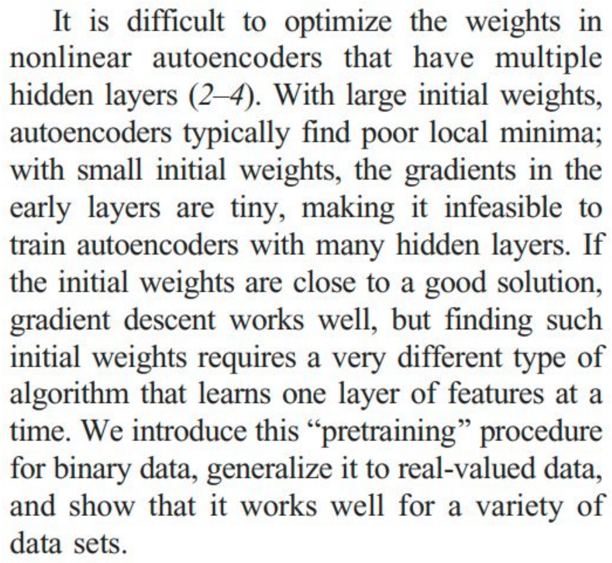


Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

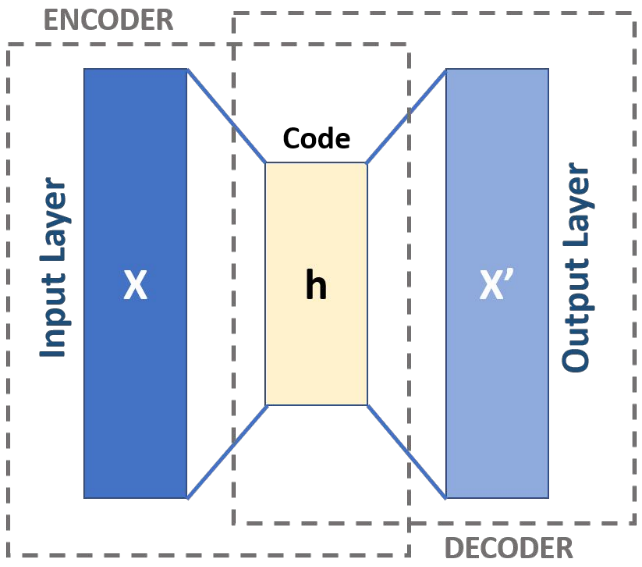
Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer "encoder" network



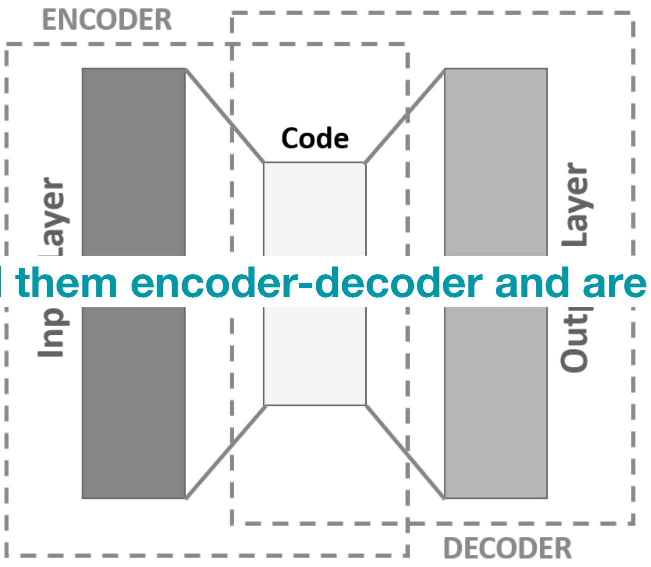
It is difficult to optimize the weights in nonlinear autoencoders that have multiple hidden layers (*2–4*). With large initial weights, autoencoders typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making it infeasible to train autoencoders with many hidden layers. If the initial weights are close to a good solution, gradient descent works well, but finding such initial weights requires a very different type of algorithm that learns one layer of features at a time. We introduce this "pretraining" procedure for binary data, generalize it to real-valued data, and show that it works well for a variety of data sets.

2006, report in Science, Vol. 313

An **autoencoder** is a neural network that is trained to attempt to copy its input to its output.

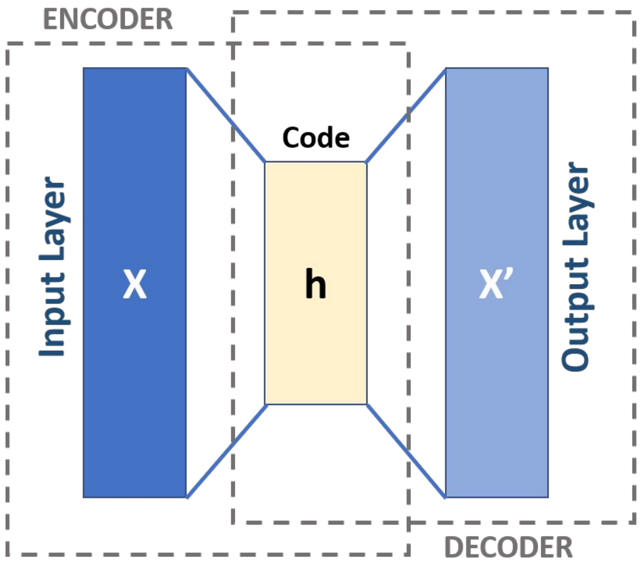


An **autoencoder** is a neural network that is trained to attempt to copy its input to its output.



Why don’t we just call them encoder-decoder and are those the same at all?

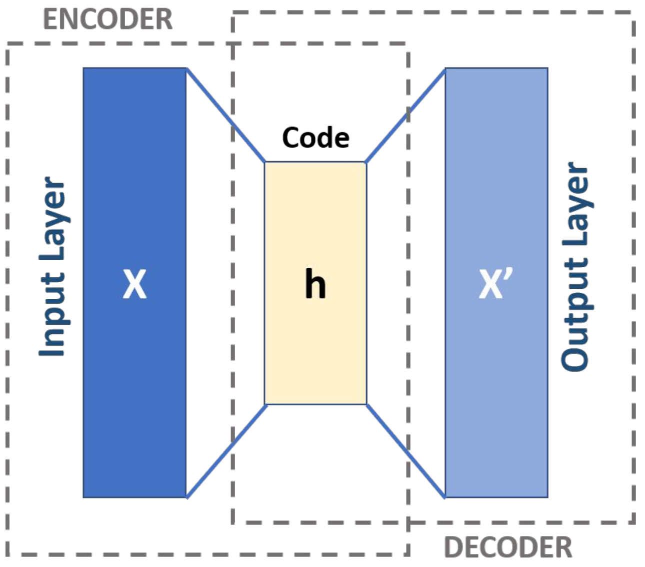
An **autoencoder** is a neural network that is trained to attempt to copy its input to its output.



Traditionally, autoencoders were used for *dimensionality reduction* or *feature learning*.

Today they are at the forefront of *generative modelling*.

An **autoencoder** is a neural network that is trained to attempt to copy its input to its output.

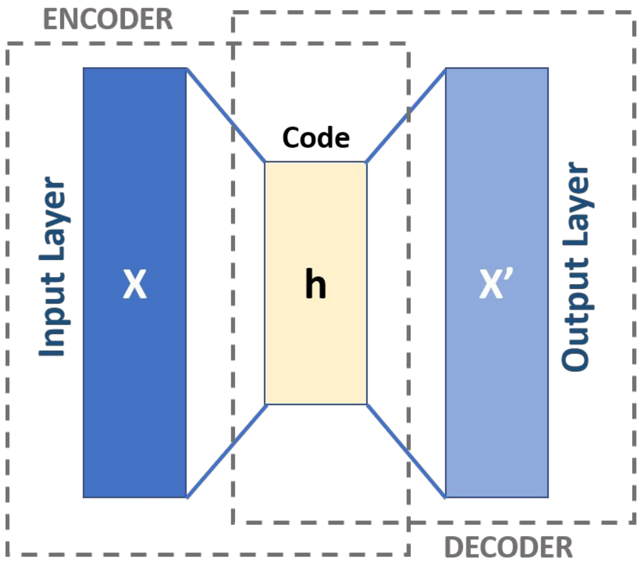


Traditionally, autoencoders were used for *dimensionality reduction* or *feature learning*.
Today they are at the forefront of *generative modelling*.

We don't want them just to learn how to copy inputs.

Usually they are *restricted* in such a way that they copy the training data only *approximately*, **prioritizing learning useful properties** of the data.

An **autoencoder** is a neural network that is trained to attempt to copy its input to its output.



If the input features were each independent of one another, this compression and subsequent reconstruction would be a very difficult task.

However, if some sort of **structure exists in the data** (i.e., correlations between input features), this structure can be learned and consequently leveraged when forcing the input through the network's bottleneck.

copying the input to the output ✗

hidden layer **h** (the **code**) taking on useful properties ✓

Undercomplete Autoencoders

`dim(code) < dim(input)`

Learning an undercomplete representation forces the autoencoder to capture the **most salient features** of the training data.

Undercomplete Autoencoders

$\mathit{dim}(\mathtt{code}) < \mathit{dim}(\mathtt{input})$

When the decoder is linear and **L** is the MSE, an undercomplete autoencoder learns to span the same subspace as **PCA.**

Autoencoder with nonlinear encoder and decoder can thus learn a more powerful nonlinear generalization of **PCA.**

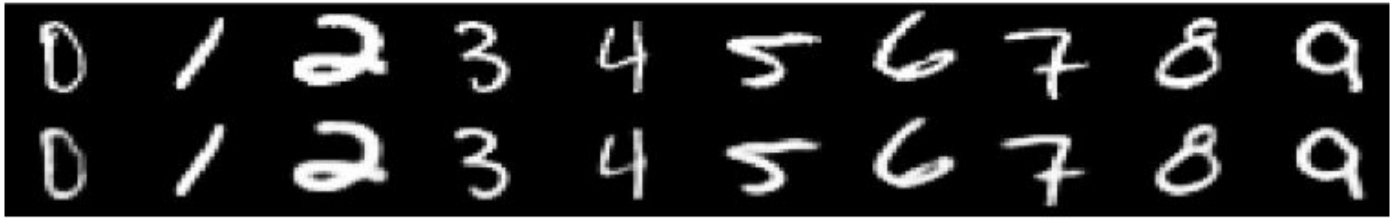
Objective function

$L(x, g(f(x)))$

PCA

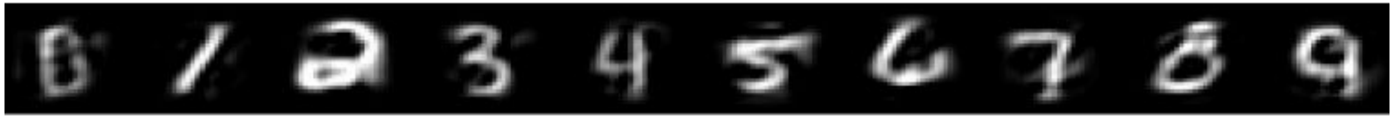
- If the hidden and output layers are linear, it will learn hidden units that are a linear function of the data and minimize the squared reconstruction error.
  - This is exactly what PCA does.
- The reconstruction error will be the same, BUT hidden units - not necessarily.
- The M hidden units will span the same space as the first M components found by PCA.
  - Their weight vectors may not be orthogonal.
  - They will tend to have equal variances.

PCA



real data

30-dim code, “deep” AE (4 layers each)

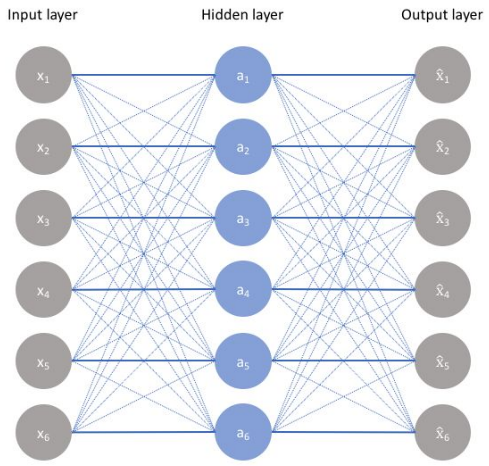


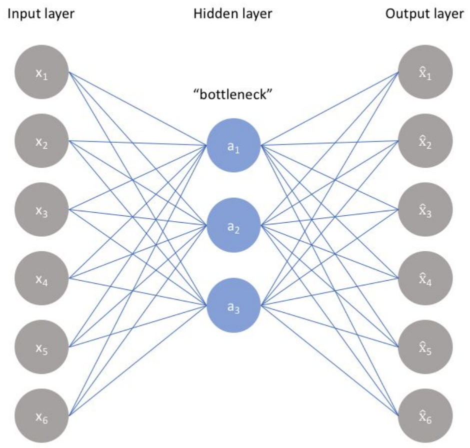
30-dim PCA

Undercomplete Autoencoders

`dim(code) < dim(input)`

If the encoder and decoder are allowed **too much capacity**, the autoencoder can learn to perform the copying task without extracting useful information about the data distribution.


Input layer
Hidden layer
Output layer
x_1 x_2 x_3 x_4 x_5 x_6
a_1 a_2 a_3 a_4 a_5 a_6
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$ $\hat{x}_5$ $\hat{x}_6$


Input layer
Hidden layer
Output layer
"bottleneck"
x_1 x_2 x_3 x_4 x_5 x_6
a_1 a_2 a_3
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$ $\hat{x}_5$ $\hat{x}_6$

Undercomplete Autoencoders

$dim(\texttt{code}) < dim(\texttt{input})$

If the encoder and decoder are allowed **too much capacity**, the autoencoder can learn to perform the copying task without extracting useful information about the data distribution.

Example:
$dim(\texttt{code}) = 1$
powerful non-linear encoder $f(x^{(i)}) = i$
powerful non-linear decoder $g(i) = x^{(i)}$

decoder learns to map indices to specific training examples
autoencoder does not learn anything useful about the training data

Regularized Autoencoders

Here, INSTEAD OF

keeping the encoder and decoder shallow

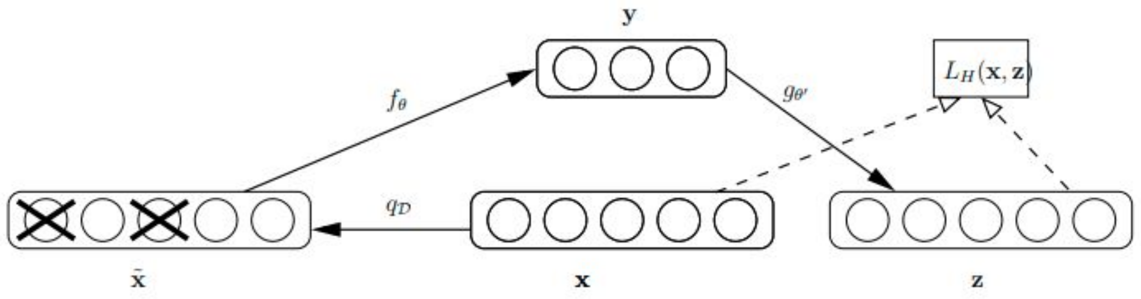
keeping the code dimension small

WE USE

additional limitations introduced via loss function:

- sparsity of the representation
- smallness of the derivative of the representation
- robustness to noise or to missing inputs

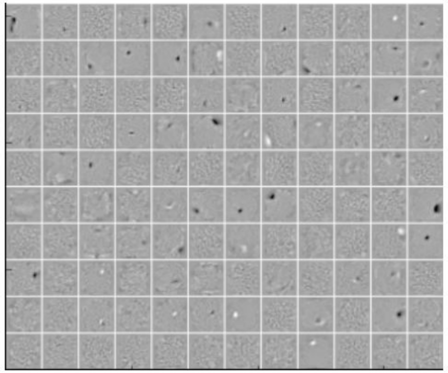
Denoising autoencoders (Vincent et al., 2008)

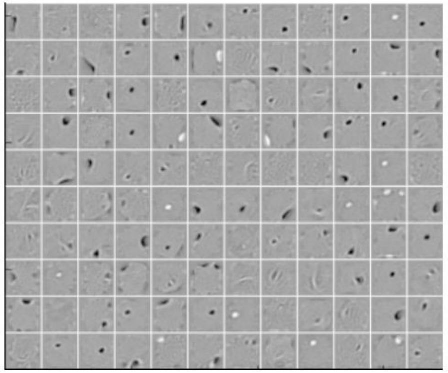


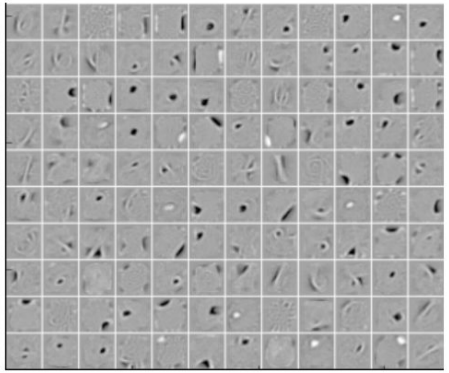
This makes the encoder retain information about structures beyond local noise.

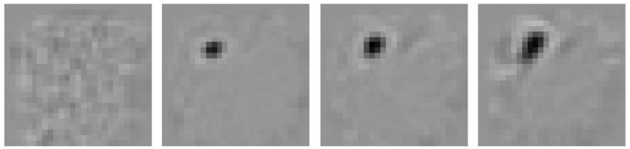
$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}})))$$

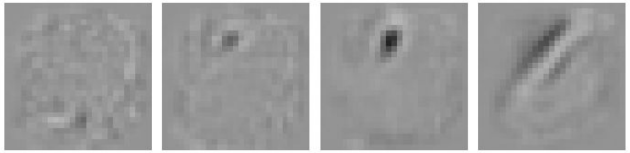
Denoising autoencoders


(a) No destroyed inputs


(b) 25% destruction


(c) 50% destruction


(d) Neuron A (0%, 10%, 20%, 50% destruction)


(e) Neuron B (0%, 10%, 20%, 50% destruction)

Contractive autoencoders (Rifai et al., 2011)

Here we try to make the activities of the hidden units as insensitive as possible to the inputs.

Contractive autoencoders (Rifai et al., 2011)

Here we try to make the activities of the hidden units as insensitive as possible to the inputs.

We achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs.

The codes tend to have the property that only a small subset of the hidden units are sensitive to changes in the input.

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x})$$

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i ||\nabla_{\boldsymbol{x}} h_i||^2$$

Contractive autoencoders (Rifai et al., 2011)

Here we try to make the activities of the hidden units as insensitive as possible to the inputs.

We achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs.

The codes tend to have the property that only a small subset of the hidden units are sensitive to changes in the input.

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x})$$

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i ||\nabla_{\boldsymbol{x}} h_i||^2$$

What would happen if we had only the second component?

Regularizations - how are they similar and how are they different?

Regularizations - how are they similar and how are they different?

- Contractive AEs and regularization via weight decay

Regularizations - how are they similar and how are they different?

- Contractive AEs and regularization via weight decay

  Linear autoencoders with L2 weight decay?

Regularizations - how are they similar and how are they different?

- Contractive AEs and regularization via weight decay

- Contractive AEs and sparse AEs

Regularizations - how are they similar and how are they different?

- Contractive AEs and regularization via weight decay

- Contractive AEs and sparse AEs

  Sparse AEs output many close-to-zero features -> highly contractive mapping (not explicitly)

Regularizations - how are they similar and how are they different?

- Contractive AEs and regularization via weight decay
- Contractive AEs and sparse AEs
- Contractive AEs and denoising AEs

Regularizations - how are they similar and how are they different?

- Contractive AEs and regularization via weight decay

- Contractive AEs and sparse autoencoders

- Contractive AEs and denoising AEs

  CAEs explicitly encourage robustness of representation f(x)
  DAEs encourages robustness of reconstruction $(g \circ f)(x)$

  that's why CAEs are better as feature extractors

  Also, DAEs obtain robustness stochastically, whereas CAEs - analytically.

Loss functions

- reconstruction (MSE) loss.

$$\mathbb{E}_X \ell(x, \overset{\text{decoder}}{d}(\overset{\text{encoder}}{e}(\underset{\substack{\text{corruption process} \\ \text{(denoising AE setting)}}}{C}(x); \phi); \theta))$$

Loss functions

- reconstruction (MSE) loss.

$$\mathbb{E}_X \ell(x, \overset{\text{decoder}}{d}(\overset{\text{encoder}}{e}(\underset{\substack{\text{corruption process} \\ \text{(denoising AE setting)}}}{C}(x);\phi);\theta))$$

- *"Here we consider mostly the squared loss and Gaussian noise corruption, again because it is easier to handle them mathematically".*

The optimal reconstruction function learned by a DAE trained to minimise the MSE is equivalent to the gradient ascent step in data space moving towards regions of higher likelihood.

$$\lim_{\sigma \to 0} \{R^*_\sigma(x)\} = x + \sigma^2 \frac{\partial \log p(x)}{\partial x}$$

Alain et al, What Regularized Auto-Encoders Learn from the Data-Generating Distribution, 2014

Loss functions

- BCE loss

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Loss functions

- BCE loss

The data distribution consists of images which take values in [0, 1] and the pixel intensity can be thought of as the probability of a pixel being “on”.

$$\mathcal{L}_{DAE} = \mathbb{E}_X[\ell_{\text{BCE}}(x, R(\tilde{x}))]$$

loss function

$$R(\tilde{x}) = d(e(\tilde{x}; \phi); \theta)$$

reconstruction model

$$R^*_\sigma(x) = \frac{\mathbb{E}_\epsilon[p(x-\epsilon)(x-\epsilon)]}{\mathbb{E}_\epsilon[p(x-\epsilon)]}$$

optimal reconstruction model, exactly the same formula as in Theorem 1 of Alain and Bengio’s paper (prev. slide)

Creswell et al, On denoising autoencoders trained to minimise binary cross-entropy, 2017

Autoencoders as an initialization method

Success of the layer-wise training strategy (back then):

unsupervised pre-training helps to mitigate the difficult optimization problem of deep networks by better initializing the weights of all layers.

Bengio et al, Greedy Layer-Wise Training of Deep Networks, NIPS 2007

Autoencoders as an initialization method

Success of the layer-wise training strategy (back then):

unsupervised pre-training helps to mitigate the difficult optimization problem of deep networks by better initializing the weights of all layers.

Why is it difficult?

- The magnitudes of gradients in the lower layers and in the higher layers are different,
- The landscape of the objective function is difficult for SGD to find a good local optimum,
- Many parameters -> remembering the training data and not generalizing well

Bengio et al, Greedy Layer-Wise Training of Deep Networks, NIPS 2007

Autoencoders as an initialization method

Success of the layer-wise training strategy (back then):

unsupervised pre-training helps to mitigate the difficult optimization problem of deep networks by better initializing the weights of all layers.

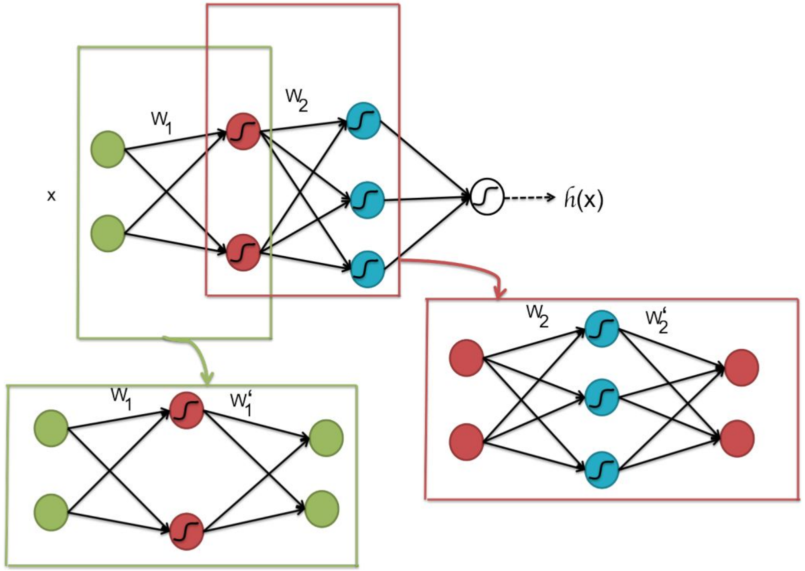
Why is it difficult?

- The magnitudes of gradients in the lower layers and in the higher layers are different,
- The landscape of the objective function is difficult for SGD to find a good local optimum,
- Many parameters -> remembering the training data and not generalizing well

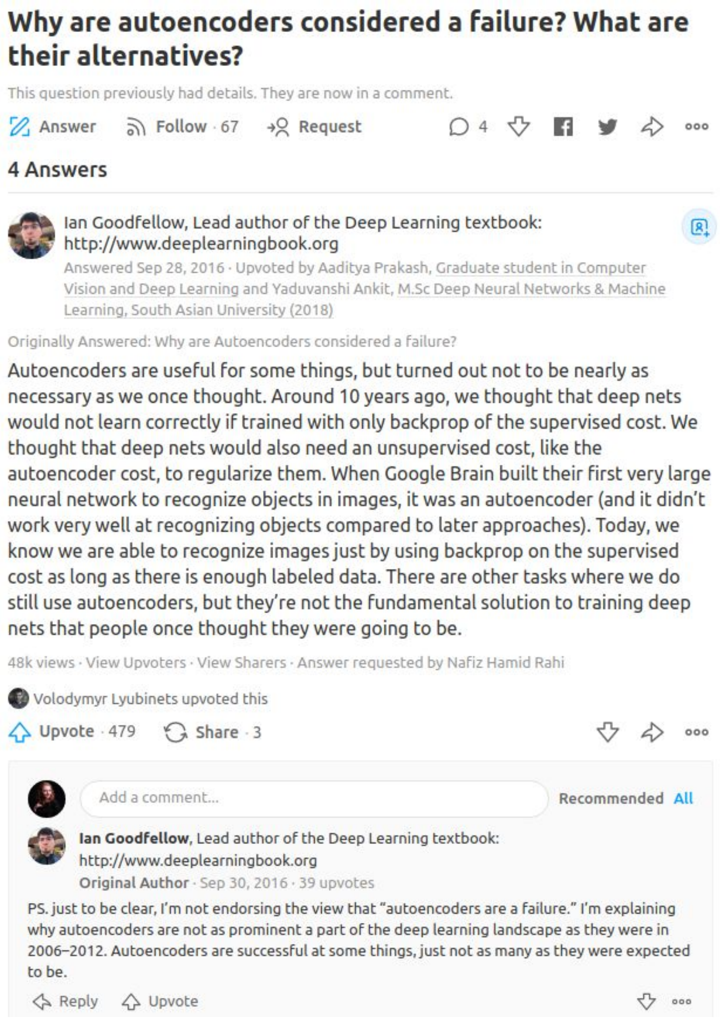
Not surprisingly, Schmidhuber claims he did it before it became popular:
multi-level hierarchy of networks in 1992.

Bengio et al, Greedy Layer-Wise Training of Deep Networks, NIPS 2007

Autoencoders as an initialization method



http://robotics.stanford.edu/~quocle/tutorial2.pdf



Why are autoencoders considered a failure? What are their alternatives?

This question previously had details. They are now in a comment.

4 Answers

Ian Goodfellow, Lead author of the Deep Learning textbook: http://www.deeplearningbook.org
Answered Sep 28, 2016 · Upvoted by Aaditya Prakash, Graduate student in Computer Vision and Deep Learning and Yaduvanshi Ankit, M.Sc Deep Neural Networks & Machine Learning, South Asian University (2018)

Originally Answered: Why are Autoencoders considered a failure?

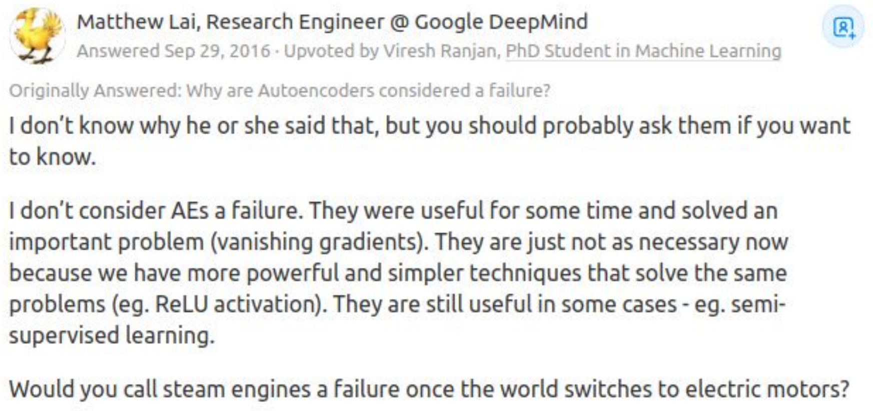
Autoencoders are useful for some things, but turned out not to be nearly as necessary as we once thought. Around 10 years ago, we thought that deep nets would not learn correctly if trained with only backprop of the supervised cost. We thought that deep nets would also need an unsupervised cost, like the autoencoder cost, to regularize them. When Google Brain built their first very large neural network to recognize objects in images, it was an autoencoder (and it didn't work very well at recognizing objects compared to later approaches). Today, we know we are able to recognize images just by using backprop on the supervised cost as long as there is enough labeled data. There are other tasks where we do still use autoencoders, but they're not the fundamental solution to training deep nets that people once thought they were going to be.

48k views · View Upvoters · View Sharers · Answer requested by Nafiz Hamid Rahi

Volodymyr Lyubinets upvoted this

Ian Goodfellow, Lead author of the Deep Learning textbook: http://www.deeplearningbook.org
Original Author · Sep 30, 2016 · 39 upvotes

PS. just to be clear, I'm not endorsing the view that "autoencoders are a failure." I'm explaining why autoencoders are not as prominent a part of the deep learning landscape as they were in 2006–2012. Autoencoders are successful at some things, just not as many as they were expected to be.



Matthew Lai, Research Engineer @ Google DeepMind
Answered Sep 29, 2016 · Upvoted by Viresh Ranjan, PhD Student in Machine Learning

Originally Answered: Why are Autoencoders considered a failure?

I don't know why he or she said that, but you should probably ask them if you want to know.

I don't consider AEs a failure. They were useful for some time and solved an important problem (vanishing gradients). They are just not as necessary now because we have more powerful and simpler techniques that solve the same problems (eg. ReLU activation). They are still useful in some cases - eg. semi-supervised learning.

Would you call steam engines a failure once the world switches to electric motors?

Useful links:

- *Bishop, Pattern Recognition and Machine Learning*
- *Goodfellow, Deep Learning*
- *Hinton, Deep Learning @ University of Toronto*
- *dig deeper into the papers referred to in the slides - there's lots of fun math there <3*

See you next time!

Agenda:

- VAEs
- GANs