Advanced Machine Learning (Semester 1 2023)

Autoencoders

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

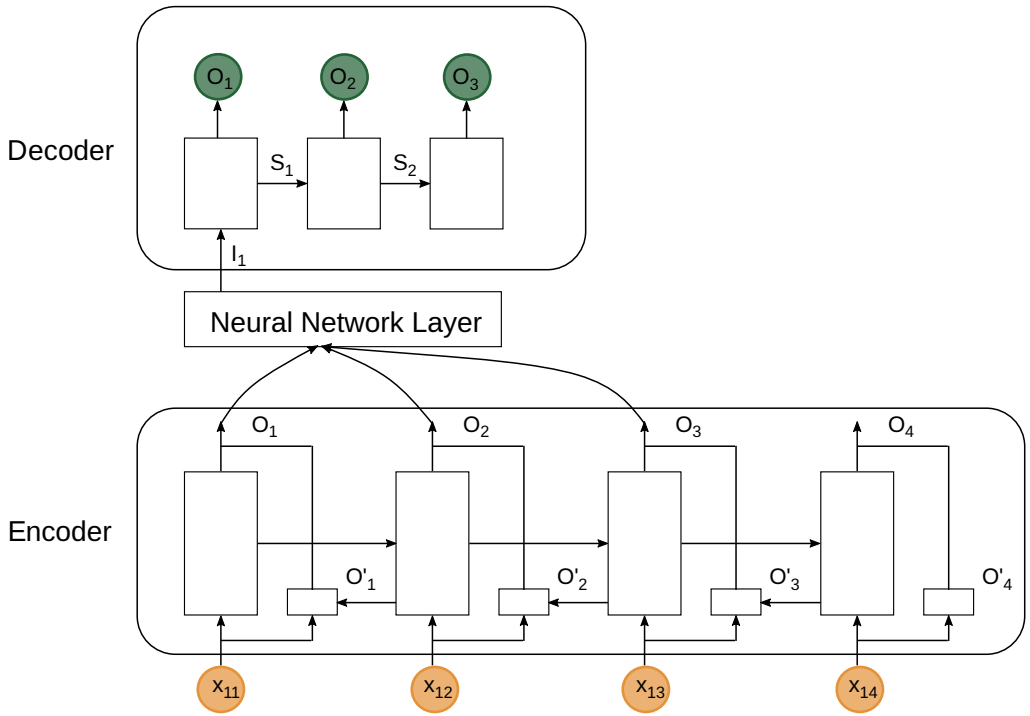
June 12, 2023

Recap

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We have looked at the **Encoder-Decoder architecture** in Sequence to Sequence (Seq2Seq) learning: Input / output networks each consisting of LSTM or GRU cells receive input data in one language and translate it into another language.

("Language" can be very abstract; also codes are a language.)

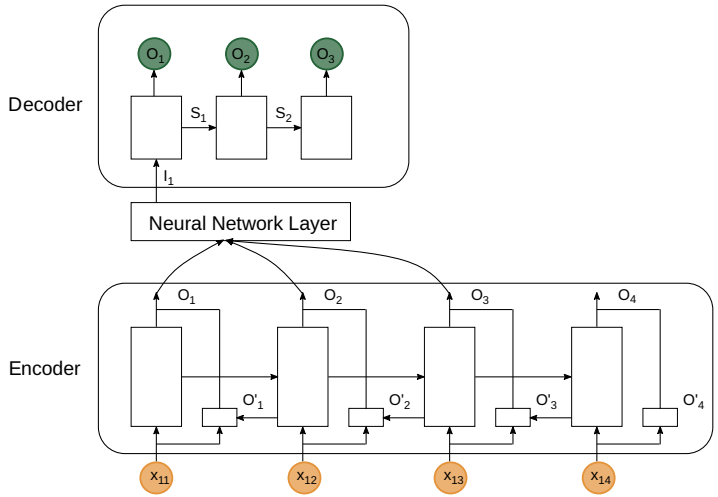


Recap

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We have looked at the **Encoder-Decoder architecture** in Sequence to Sequence (Seq2Seq) learning: Input / output networks each consisting of LSTM or GRU cells receive input data in one language and translate it into another language.

("Language" can be very abstract; also codes are a language.)



The drawback of Encoder and Decoder:
The context vector summarizes the input sequence but not all words will be equally relevant. This is overcome by **Attention-Based models** where a neural network between encoder and decoder learns filtering information.

Motivation

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

remember: unsupervised learning where we have access to training inputs $x_1, ..., x_N$ but not to training outputs $y_1, ..., y_N$

Motivation

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

remember: unsupervised learning where we have access to training inputs $x_1, ..., x_N$ but not to training outputs $y_1, ..., y_N$

this is a task relevant to:

1. extract interesting information from data
   - clustering
   - data compression
   - trends, regions in parameter space

Motivation

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

remember: unsupervised learning where we have access to training inputs $x_1, ..., x_N$ but not to training outputs $y_1, ..., y_N$

this is a task relevant to:

1. extract interesting information from data
   - clustering
   - data compression
   - trends, regions in parameter space
2. learning better representations
   - feature extraction
   - dimensionality reduction

Motivation

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

remember: unsupervised learning where we have access to training inputs $x_1, ..., x_N$ but not to training outputs $y_1, ..., y_N$

this is a task relevant to:

1. extract interesting information from data
   - clustering
   - data compression
   - trends, regions in parameter space
2. learning better representations
   - feature extraction
   - dimensionality reduction

good representation without overfitting the data
better generalization

Motivation

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

remember: unsupervised learning where we have access to training inputs $x_1, ..., x_N$ but not to training outputs $y_1, ..., y_N$

this is a task relevant to:

1. extract interesting information from data
   - clustering
   - data compression
   - trends, regions in parameter space
2. learning better representations
   - feature extraction
   - dimensionality reduction

good representation without overfitting the data
better generalization

this can also be used in a 2-step process to initialize a supervised classification process

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Definition

An autoencoder is a type of algorithm with the primary purpose of learning an informative representation of the data that can be used for different applications by learning to reconstruct a set of input observations well enough.

D. Bank, N. Koenigstein & R. Giryes, `https://arxiv.org/abs/2003.05991`

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

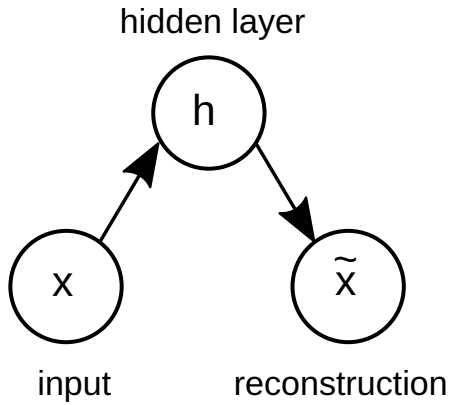
Definition

An autoencoder is a type of algorithm with the primary purpose of learning an informative representation of the data that can be used for different applications by learning to reconstruct a set of input observations well enough.

D. Bank, N. Koenigstein & R. Giryes, `https://arxiv.org/abs/2003.05991`

Structure of an Autoencoder:
An autoencoder is a neural network that is trained to attempt to copy its input to its output.
It has a hidden layer h that represents the input.
Its neural network be viewed as consisting of two parts:

- encoder function $h = f(x)$
- decoder function that produces a reconstruction: $\tilde{x} = g(h) = g(f(x))$

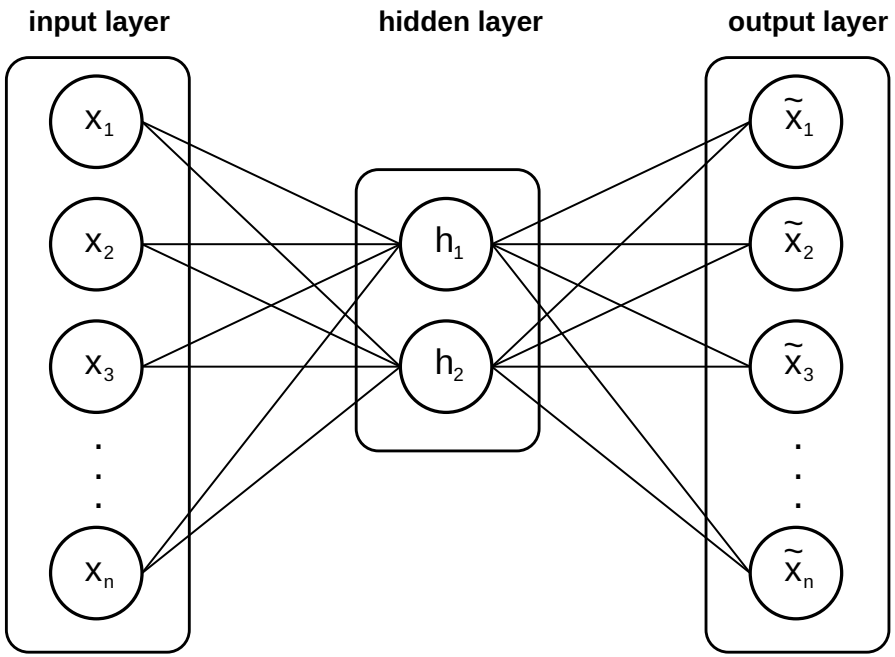


Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

self-encoding: feed-forward network intended to reproduce the input

a **basic** autoencoder:

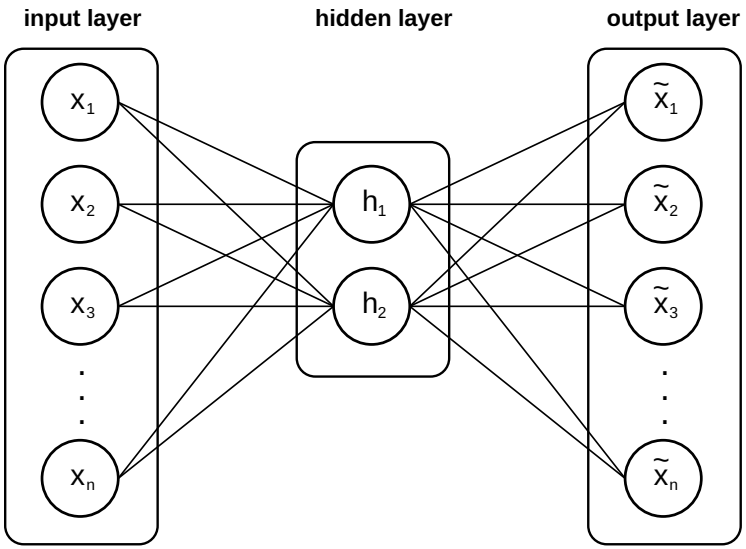


encoder: $f = \varphi(\mathbf{W}\mathbf{x})$
decoder: $g = \varphi(\mathbf{W}'\mathbf{h})$
score function: $\tilde{x} = f(g(\mathbf{x}))$, $E(\tilde{\mathbf{x}}, \mathbf{x})$

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

self-encoding: feed-forward network intended to reproduce the input



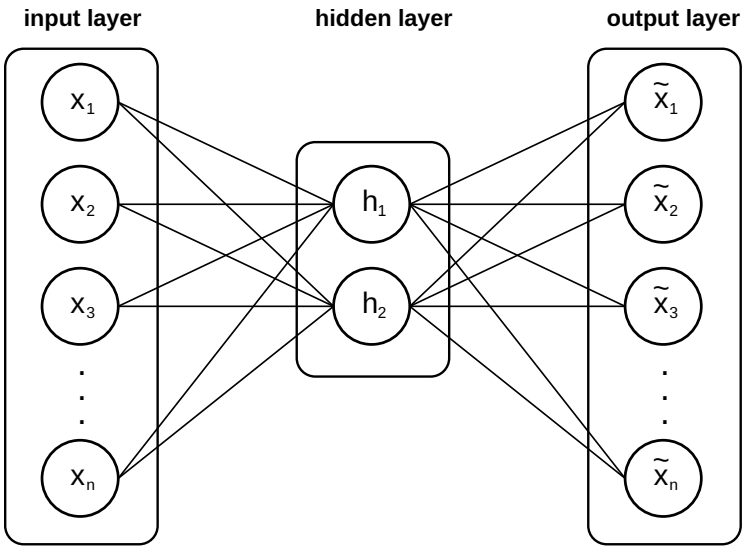
in general: usage of **subnetworks**, i.e:

- encoder subnetwork $f(\cdot)$ maps input to an embedded representation $\mathbf{h}$: $\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$
- decoder subnetwork $g(\cdot)$ maps embedded representation $\mathbf{h}$ back into input space $\tilde{x} = g(W'\mathbf{h} + \mathbf{c})$

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

self-encoding: feed-forward network intended to reproduce the input



in general: usage of **subnetworks**, i.e:

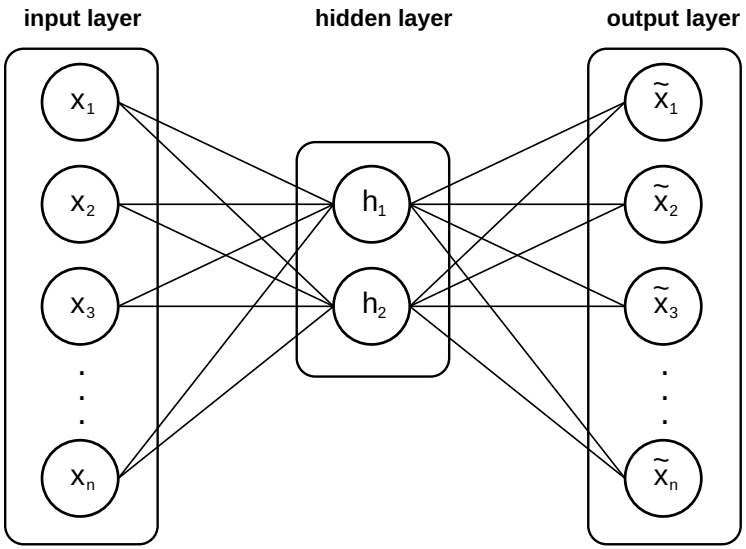
- encoder subnetwork $f(\cdot)$ maps input to an embedded representation h: $\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$
- decoder subnetwork $g(\cdot)$ maps embedded representation h back into input space $\tilde{x} = g(W'\mathbf{h} + \mathbf{c})$

⇒ this can be thought as a **lossy compression** of the input which needs to identify the important attributes of inputs

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

self-encoding: feed-forward network intended to reproduce the input



further considerations:

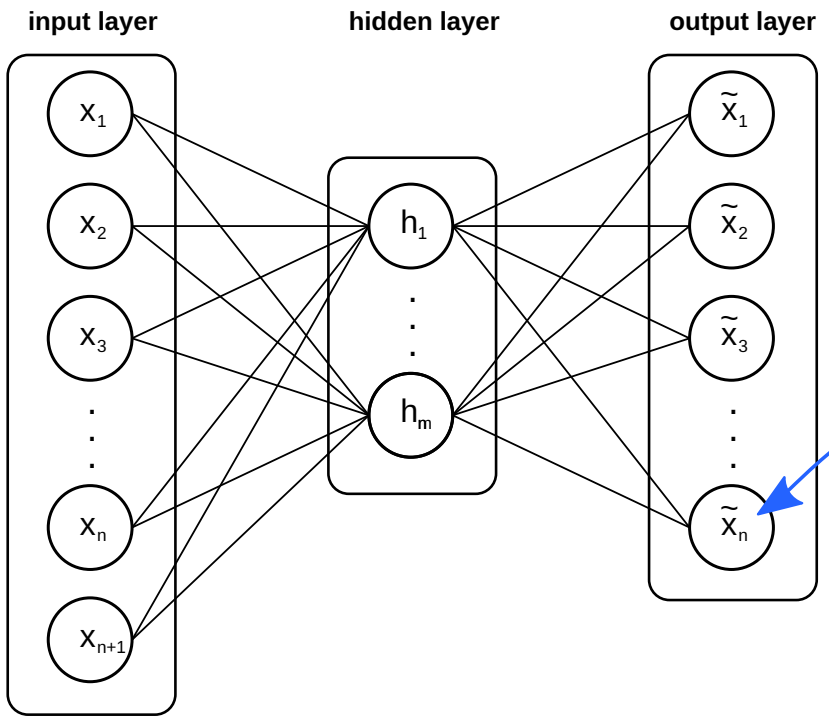
- activation function φ depends on type of data (e.g. sigmoid for binary)
- typically multiple hidden layers are used
- often uses tied weights $\mathbf{W}' = \mathbf{W}$

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

traditional application for autoencoders: unsupervised learning
goal: learning the structure of data without using labels

⇒ dimensionality reduction due to the autoencoder's **architecture**



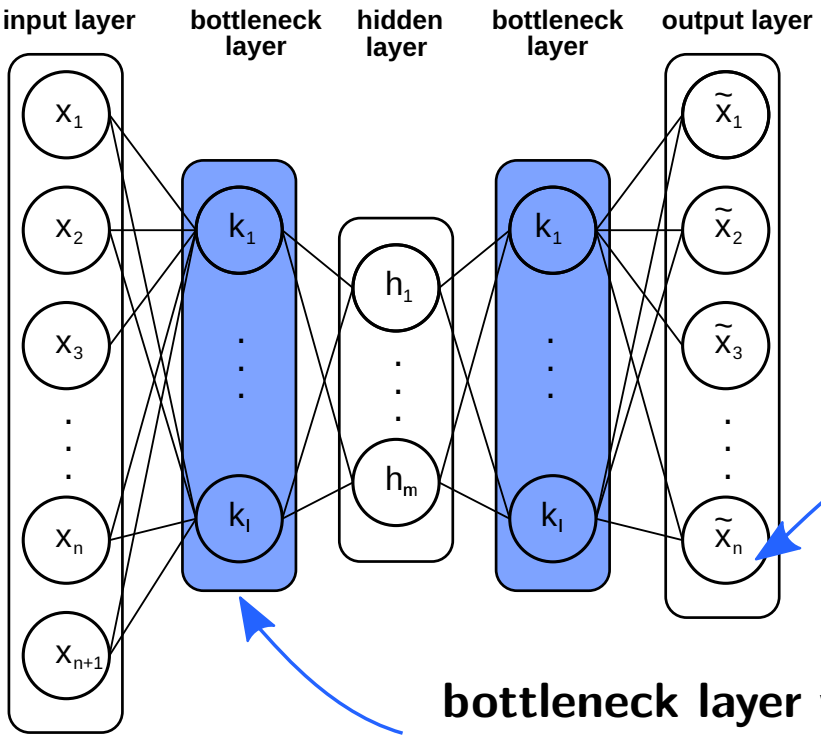
one less in the output layer - dimensionality reduction

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

traditional application for autoencoders: unsupervised learning
goal: learning the structure of data without using labels

⇒ dimensionality reduction due to the autoencoder's **architecture**



one less in the output layer - dimensionality reduction

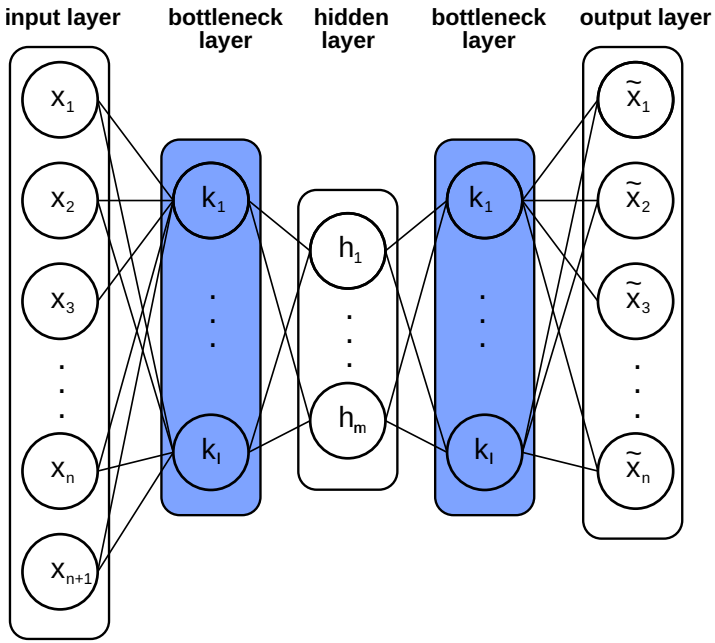
bottleneck layer with $n < l < m$
make this non-trivial

Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

General Autoencoder Architecture

Encoder: compresses the train-validate-test set input data into an encoded representation several orders of magnitude smaller than the input data.
Bottleneck, Hidden Layer: contains the compressed knowledge representations.
Decoder: decompresses the knowledge representations and reconstructs the data from its encoded form. In training, the output is then compared with a ground truth.



Autoencoders - Typical Applications

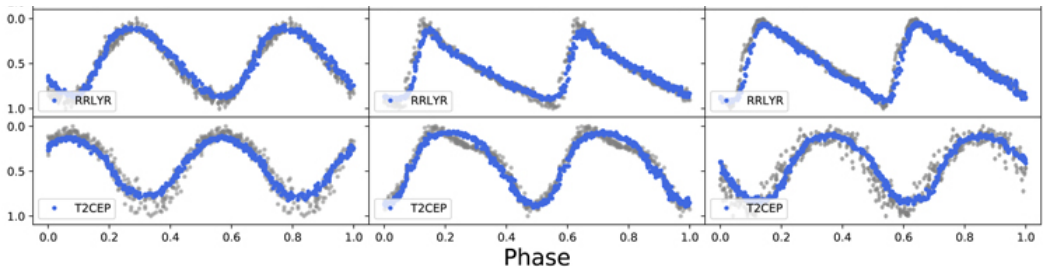
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Autoencoders are generally used to reduce data dimensionality or find a better suited data representation

applications:

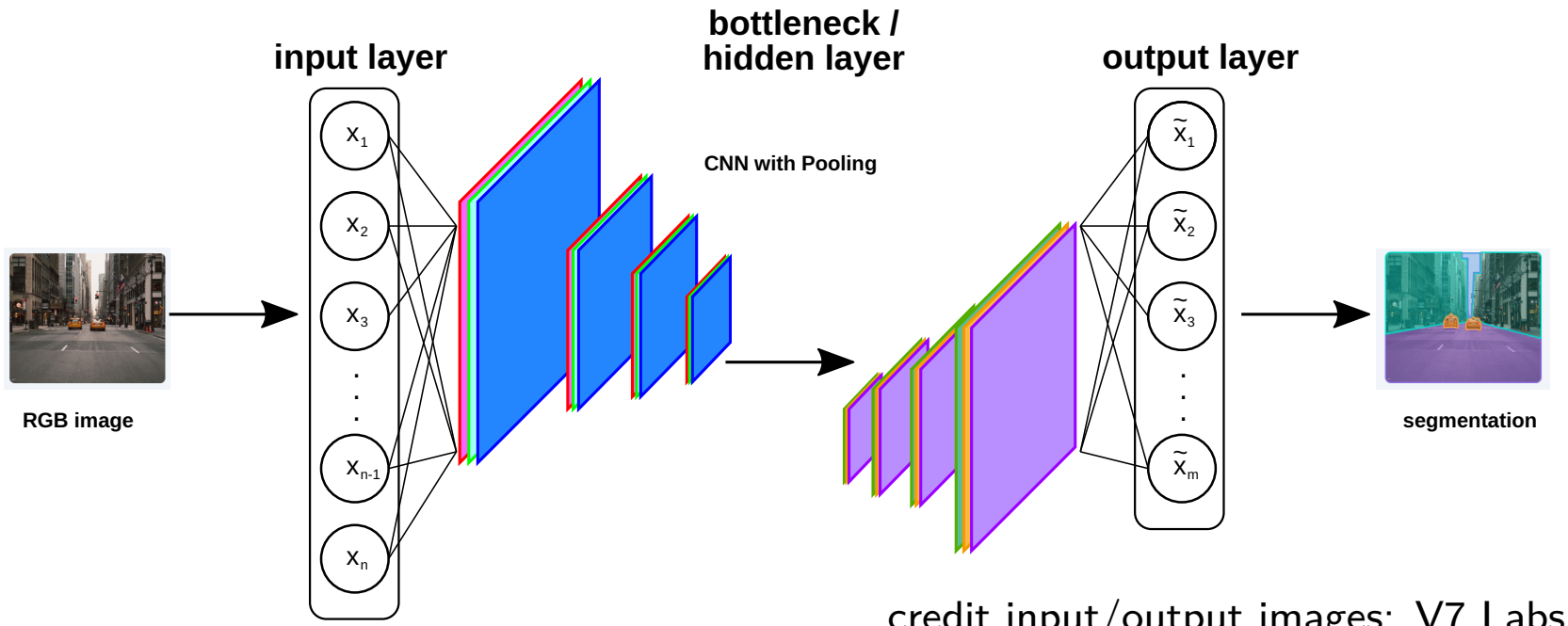
- generation of new instances similar to those of the input/ training set
- mapping high-dimensional data to two dimensions for visualization
- data compression, i.e. file size reduction
- generalizing: learning abstract features in an unsupervised way to be applied later to a supervised task

astronomy example:
Deep Generative Modeling of Periodic Variable Stars Using Physical Parameters, J. Martìnez-Palomera, J. S. Bloom, E. S. Abrahams (2022)



Autoencoders and CNN

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications



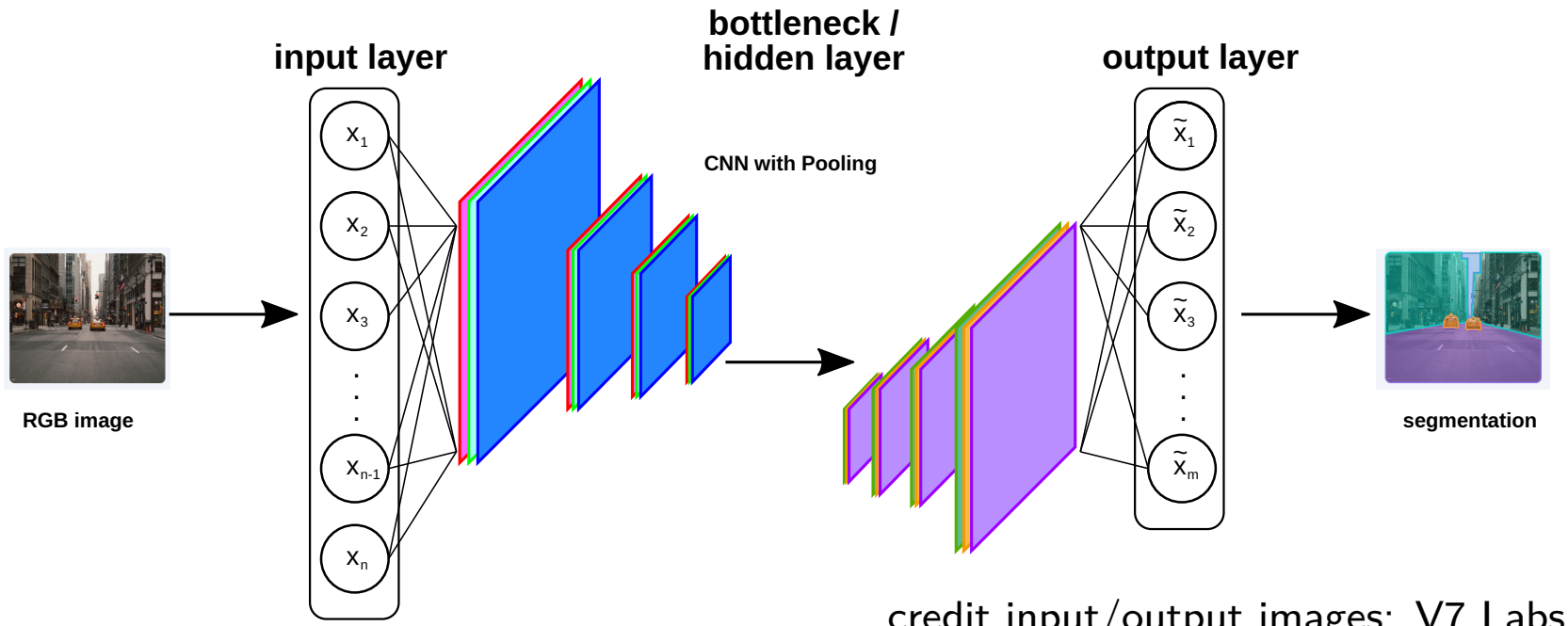
credit input/output images: V7 Labs

Encoder

The encoder is a set of convolutional blocks followed by pooling modules that compress the input.

Autoencoders and CNN

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications



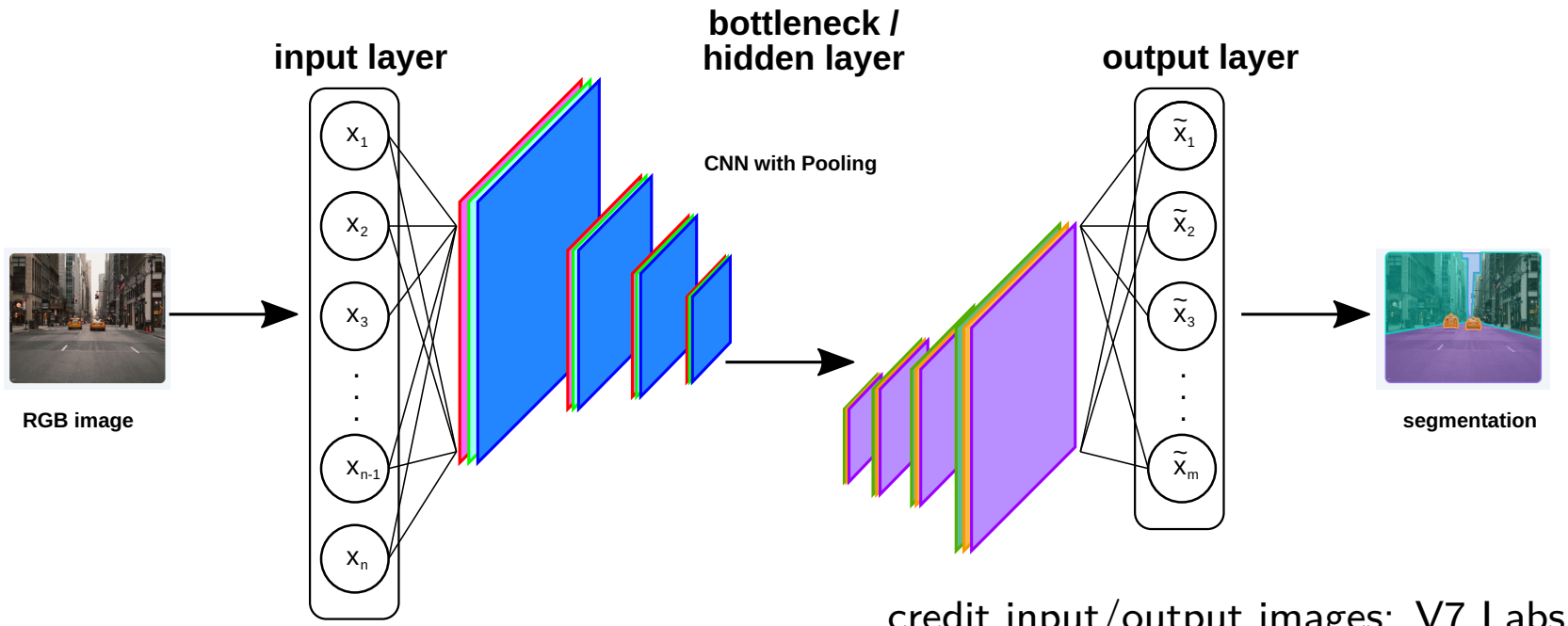
credit input/output images: V7 Labs

Bottleneck

The bottleneck restricts the information flow from the encoder to the decoder, allowing only the most vital information to pass through. It forms a knowledge-representation of the input by establishing useful correlations between various inputs within the network. This representation further prevents the neural network from memorising the input and overfitting.

Autoencoders and CNN

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications



credit input/output images: V7 Labs

Decoder

Finally, the decoder is a set of upsampling and convolutional blocks which input is the compressed knowledge representation. The decoder serves as a decompressor and builds back the input from its latent attributes. For simple autoencoders, the output is expected to be the a denoised version of the input.

Feature Representations in Autoencoders

How to **learn** feature representations?

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

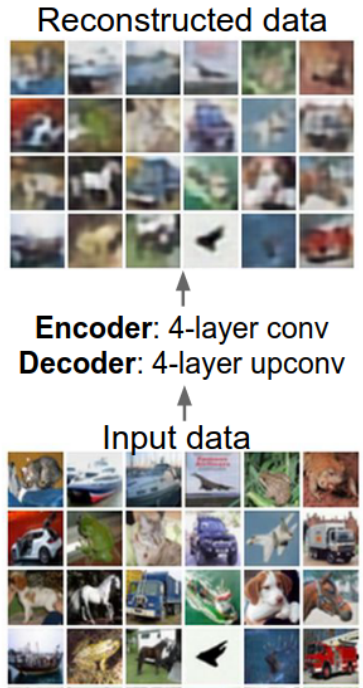
Feature Representations in Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

How to **learn** feature representations?

train such that features can be used to reconstruct original data, thus "autoencoding"

using a L_2 loss function $\|\mathbf{x} - \tilde{\mathbf{x}}\|^2$

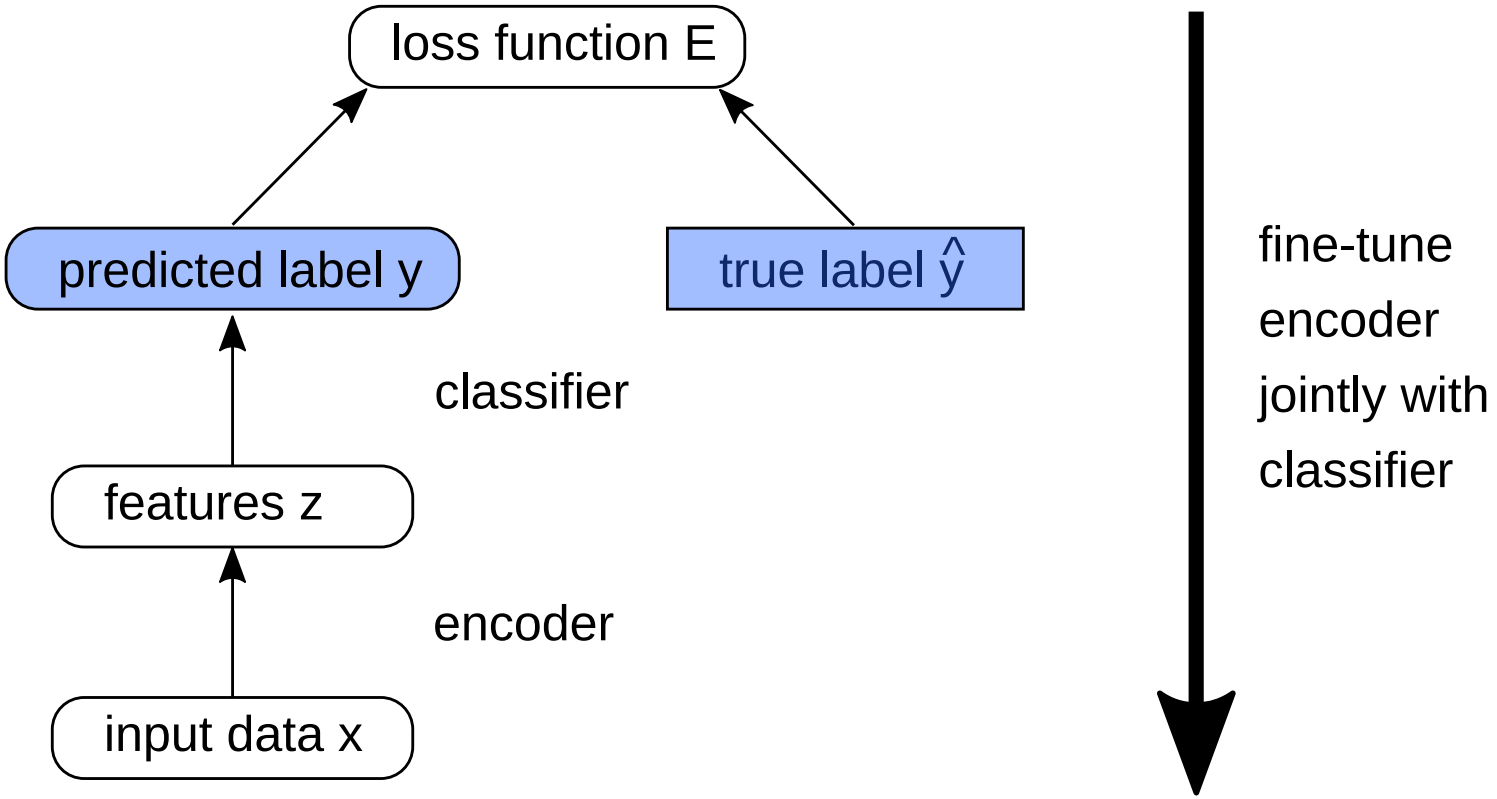


credit input/output images: V7 Labs

Feature Representations in Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

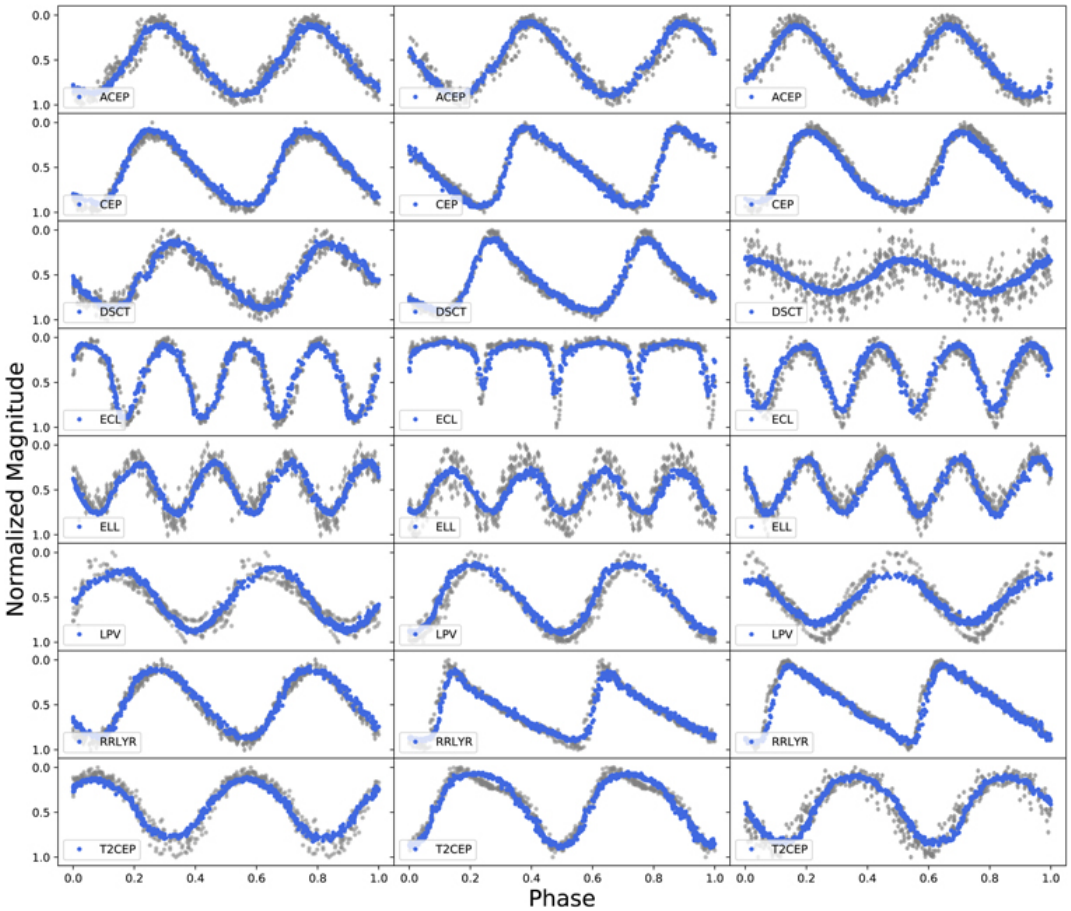
The encoder can be used to **initialize a supervised** model



Feature Representations in Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

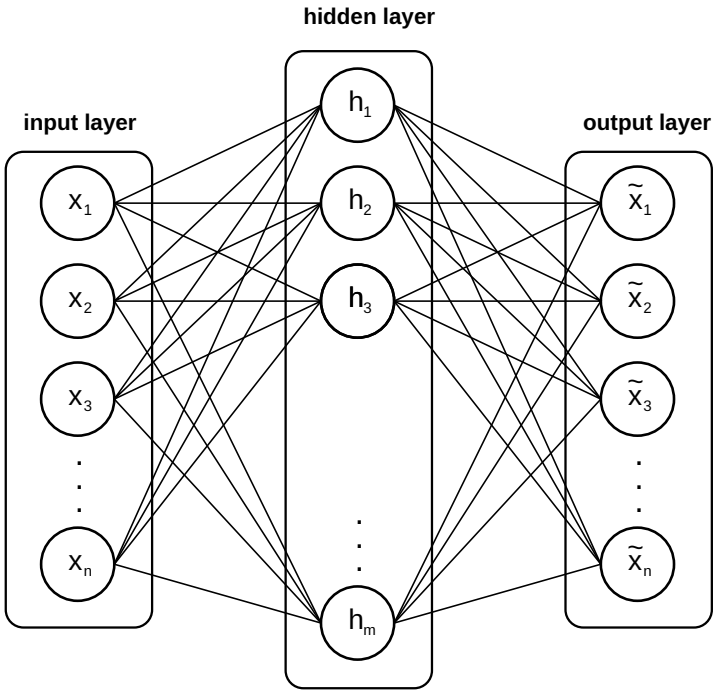
feature representations can be used to **generate** new images:



Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

consider the case where $m = \dim(\mathbf{h}) \geq \dim(\mathbf{x}) = n$
$\Rightarrow$ **overcomplete autoencoder**



$$\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$$
$$\tilde{\mathbf{x}} = g(W'\mathbf{h} + \mathbf{c})$$

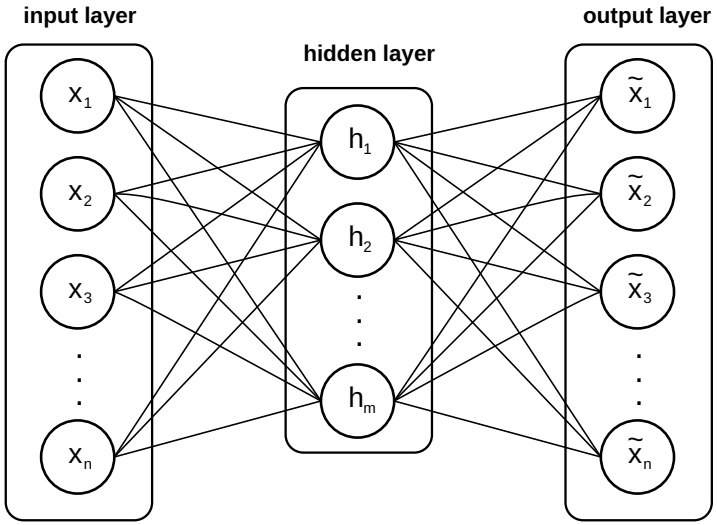
in such a case the autoencoder could learn a **trivial encoding** by simply copying $\mathbf{x}$ into $\mathbf{h}$ and then copying $\mathbf{h}$ into $\tilde{\mathbf{x}}$

such an identity encoding is useless in practice as it does not tell us anything about the important characteristics of the data

Autoencoders

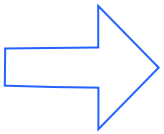
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

consider the case where $m = \dim(\mathbf{h}) < \dim(\mathbf{x}) = n$
$\Rightarrow$ **undercomplete autoencoder**



$$\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$$
$$\tilde{\mathbf{x}} = g(W'\mathbf{h} + \mathbf{c})$$

if we are still able to reconstruct $\tilde{\mathbf{x}}$ perfectly from h, this tells us that h is a loss-free encoding of $\mathbf{x}$: it captures all the important characteristics of $\mathbf{x}$



analogy with PCA

Autoencoders and Principal Component Analysis

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

What is the difference between a PCA and an autoencoder?

Autoencoders and Principal Component Analysis

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

What is the difference between a PCA and an autoencoder?

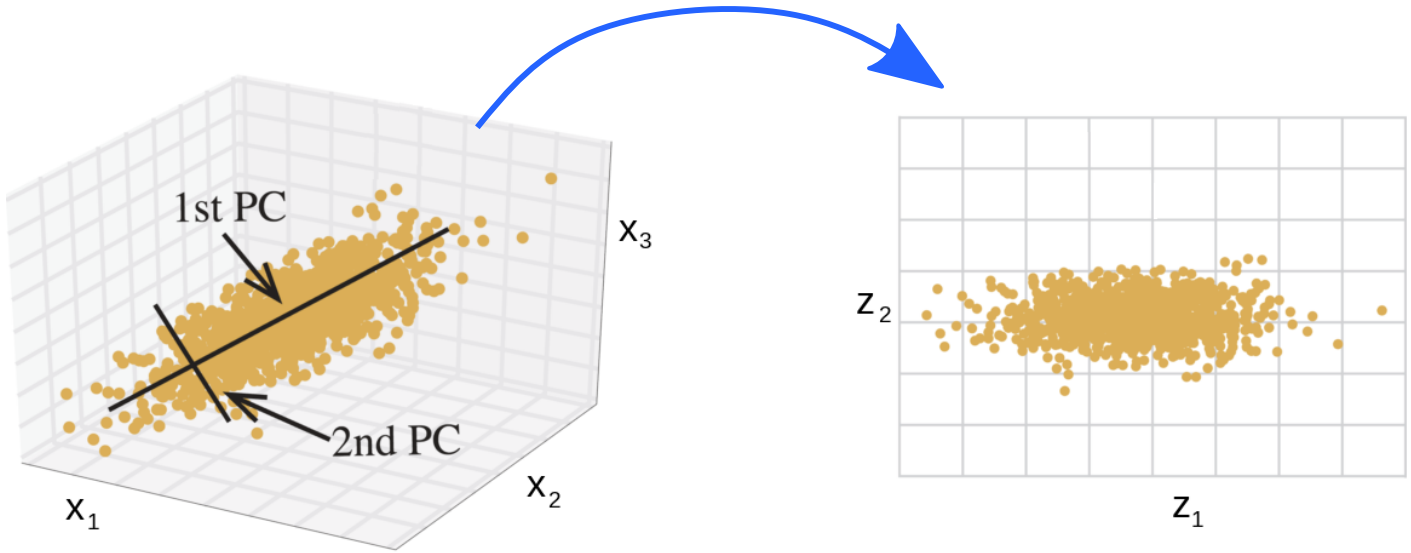
PCA is restricted to a linear map, while autoencoders can have nonlinear enoder/decoders.

Autoencoders and Principal Component Analysis

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

An autoencoder with one hidden layer, linear activations, normalization of the inputs and a square loss function performs principal component analysis:

finding **linear transformation** of data into new coordinates to maximize variance

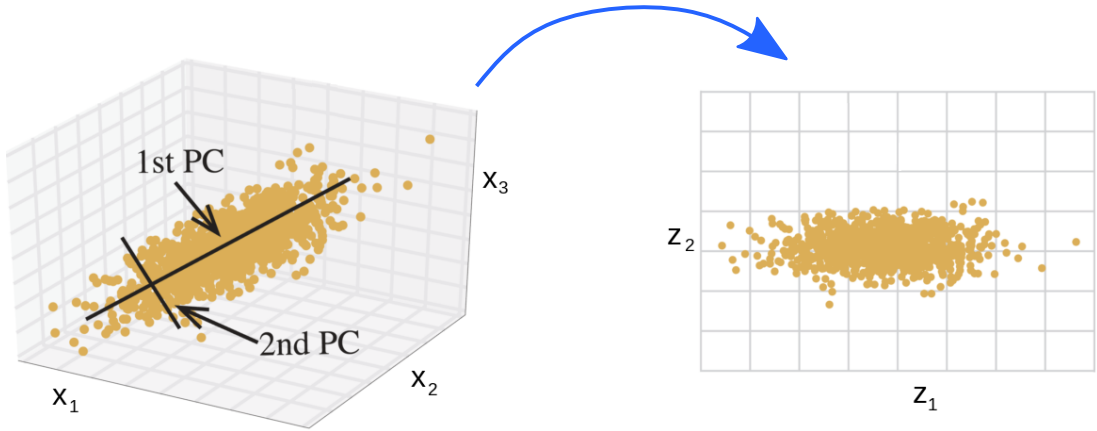


Autoencoders and Principal Component Analysis

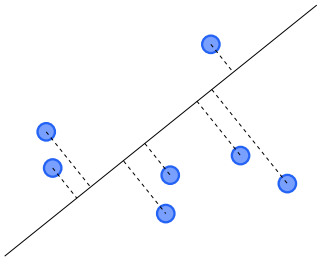
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

An autoencoder with one hidden layer, linear activations, normalization of the inputs and a square loss function performs principal component analysis:

finding **linear transformation** of data into new coordinates to maximize variance



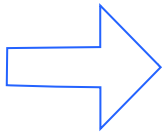
this is possible as a linear autoencoder that maps a n-dimensional input into a m-dimensional subspace Z: by definition, the projection of $\mathbf{x}$ into Z is the point in Z which minimizes the distance to $\mathbf{x}$



the linear autoencoder represents the projection by chosing $W = Q$ and $W' = Q^T$ where Q is an orthonormal basis for Z

Autoencoders and Principal Component Analysis

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

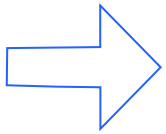


an autoencoder is equivalent to PCA for the following choices:

- linear encoder
- linear decoder
- square loss function
- normalize the inputs to $\tilde{x}_{ij} = \frac{1}{\sqrt{m}}(x_{ij} - \frac{1}{m}\sum_{k=1}^{m} x_{kj})$

Autoencoders and Principal Component Analysis

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications



an autoencoder is equivalent to PCA for the following choices:

- linear encoder
- linear decoder
- square loss function
- normalize the inputs to $\tilde{x}_{ij} = \frac{1}{\sqrt{m}}(x_{ij} - \frac{1}{m}\sum_{k=1}^{m} x_{kj})$

Why **normalizing** the inputs?

The operation in the brackets ensures that each data now has 0 mean along each dimension j. With zero-mean matrix X', the above normalization give us $X = \frac{1}{\sqrt{m}}X'$. Now $X^T X = \frac{1}{m}(X')^T X'$ is the covariance matrix.
If we use a linear decoder with squared loss function, the optimal solution to the following objective function is obtained by using a linear encoder:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}(x_{ij} - \tilde{x}_{ij})^2$$

Training Autoencoders

We can think of each neuron as a **filter** which will fire (or get maximally activated) for a certain input configuration $\mathbf{x}_i$

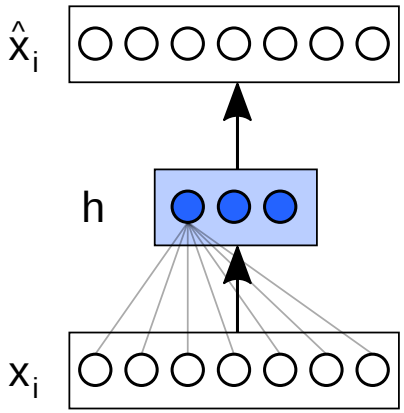
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We can think of each neuron as a **filter** which will fire (or get maximally activated) for a certain input configuration $\mathbf{x}_i$

example: $h_{11} = \sigma(W_{11}^T \mathbf{x}_i)$ (here ignoring bias **b**) where W_{11} is the trained weight vector connecting the input to the first neuron in hidden layer 1, h_{11} which values of x_i will cause h_{11} to be maximally activated?



suppose normalized inputs, i.e. $\|x_i\| = 1$
solution:

$$x_i = \frac{W_{11}}{\sqrt{W_{11}^T W_{11}}}$$

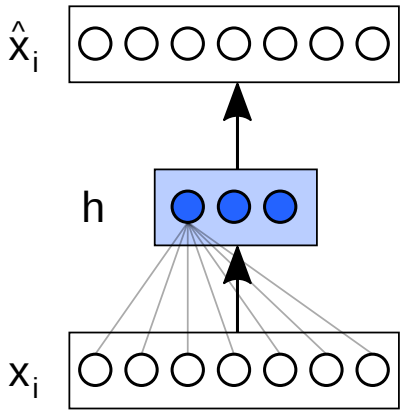
thus the inputs $x_i = \frac{W_{11}}{\sqrt{W_{11}^T W_{11}}}, \ldots, \frac{W_{1n}}{\sqrt{W_{1n}^T W_{1n}}}$ will cause hidden neurons $1 - n$ in hidden layer 1 to fire maximally

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We can think of each neuron as a **filter** which will fire (or get maximally activated) for a certain input configuration $\mathbf{x}_i$

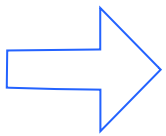
example: $h_{11} = \sigma(W_{11}^T \mathbf{x}_i)$ (here ignoring bias **b**) where W_{11} is the trained weight vector connecting the input to the first neuron in hidden layer 1, h_{11} which values of x_i will cause h_{11} to be maximally activated?



suppose normalized inputs, i.e. $\|x_i\| = 1$
solution:

$$x_i = \frac{W_{11}}{\sqrt{W_{11}^T W_{11}}}$$

thus the inputs $x_i = \frac{W_{11}}{\sqrt{W_{11}^T W_{11}}}, \ldots, \frac{W_{1n}}{\sqrt{W_{1n}^T W_{1n}}}$ will cause hidden neurons $1 - n$ in hidden layer 1 to fire maximally



with this idea we can tune **W** (and **b**) so they fire maximally for the correct output

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

training is done by **minimizing the reconstruction loss function** $E = \sum |\mathbf{x} - \tilde{\mathbf{x}}|$

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

training is done by **minimizing the reconstruction loss function** $E = \sum |\mathbf{x} - \tilde{\mathbf{x}}|$

we need to set **hyperparameters:**

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

training is done by **minimizing the reconstruction loss function** $E = \sum |\mathbf{x} - \tilde{\mathbf{x}}|$

we need to set **hyperparameters:**

- $f(\cdot)$ and $g(\cdot)$
- code size: this is the size of the bottleneck; it decides how much the data is compressed and can also act as a regularisation term
- depth: number of layers; while a higher depth increases model complexity, a lower depth is faster to process
- nodes per layer: typically, the number of nodes decreases with each subsequent layer in the autoencoder as the input to each of these layers becomes smaller
- reconstruction loss E: the loss function we use to train the autoencoder is highly dependent on the type of input and output we want the autoencoder to adapt to

popular loss functions for reconstructing image data: MSE and L1 loss
popular loss for inputs within the range [0,1]: Binary Cross Entropy

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We consider here the case of an autoencoder with one hidden layer. We've seen: the objective of the autoencoder is to reconstruct $\tilde{\mathbf{x}}$ to be as close to $\mathbf{x}$ as possible.

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We consider here the case of an autoencoder with one hidden layer. We've seen: the objective of the autoencoder is to reconstruct $\tilde{\mathbf{x}}$ to be as close to $\mathbf{x}$ as possible.

With what we already had before

$$\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$$
$$\tilde{\mathbf{x}} = g(W'\mathbf{h} + \mathbf{c})$$

for real-valued inputs this can be formalized using the following objective function for a training set with N objects:

$$\min_{W,W',c,b} \frac{1}{N} \sum_{i=1}^{N} (\tilde{\mathbf{x}}_i - \mathbf{x}_i)^T (\tilde{\mathbf{x}}_i - \mathbf{x}_i)$$

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We consider here the case of an autoencoder with one hidden layer. We've seen: the objective of the autoencoder is to reconstruct $\tilde{\mathbf{x}}$ to be as close to $\mathbf{x}$ as possible.

With what we already had before

$$\mathbf{h} = f(W\mathbf{x} + \mathbf{b})$$
$$\tilde{\mathbf{x}} = g(W'\mathbf{h} + \mathbf{c})$$

for real-valued inputs this can be formalized using the following objective function for a training set with N objects:

$$\min_{W,W',c,b} \frac{1}{N} \sum_{i=1}^{N} (\tilde{\mathbf{x}}_i - \mathbf{x}_i)^T (\tilde{\mathbf{x}}_i - \mathbf{x}_i)$$

We can then train the autoencoder just like a regular feedforward network using **backpropagation**.

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

In the same way, when we consider the case where the inputs are binary: We use a sigmoid decoder which will produce outputs between 0 and 1 that can be interpreted as probabilities for a single n-dimensional ith input we can use the following loss function:

$$\min \left\{ -\sum_{j=1}^{n} (x_{ij} \log \tilde{x}_{ij} + (1 - x_{ij}) \log(1 - \tilde{x}_{ij})) \right\}$$

Again, this can be solved with backpropagation.

Training Autoencoders

How to define a loss function?

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

How to define a loss function?

We've seen in the beginning: $g(\cdot)$ is the decoder producing $\tilde{\mathbf{x}}$. We can then define a loss function as the difference between input $\mathbf{x}$ and output $\tilde{\mathbf{x}}$, i.e. a L2 loss function $\|\mathbf{x} - \tilde{\mathbf{x}}\|^2$.

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

How to define a loss function?

We've seen in the beginning: $g(\cdot)$ is the decoder producing $\tilde{\mathbf{x}}$. We can then define a loss function as the difference between input $\mathbf{x}$ and output $\tilde{\mathbf{x}}$, i.e. a L2 loss function $\|\mathbf{x} - \tilde{\mathbf{x}}\|^2$.

⇒ the autoencoder tries to learn an approximation of the identity

If we have the same number of nodes (or more) in the hidden layer than in the input, the autoencoder would learn identity which would be useless

⇒ we've seen we restrict the number of nodes in the hidden layer (**bottleneck**)

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

another way to prevent overfitting: add **Regularization** to the autoencoder

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

another way to prevent overfitting: add **Regularization** to the autoencoder

the simplest solution: add a L_2 regularization term to the objective function

$$\min_{\theta, W, W', c, b} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} (\tilde{\mathbf{x}}_{ij} - \mathbf{x}_{ij})^T \lambda ||\theta||^2$$

$\Rightarrow$ this is very easy to implement and just adds a term λW to the gradient $\frac{\partial E(\theta)}{\partial W}$ and similarly for other parameters

Training Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

another way to prevent overfitting: add **Regularization** to the autoencoder

the simplest solution: add a L_2 regularization term to the objective function

$$\min_{\theta, W, W', c, b} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} (\tilde{\mathbf{x}}_{ij} - \mathbf{x}_{ij})^T \lambda ||\theta||^2$$

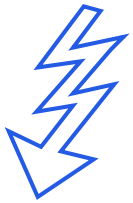
$\Rightarrow$ this is very easy to implement and just adds a term λW to the gradient $\frac{\partial E(\theta)}{\partial W}$ and similarly for other parameters

Another trick (we've seen this before, but without explanation) is to tie the weights of the encoder and decoder, i.e. $W' = W^T$. This effectively **reduces the capacity** of the autoencoder and acts as a regularizer.

Autoencoder Architectures

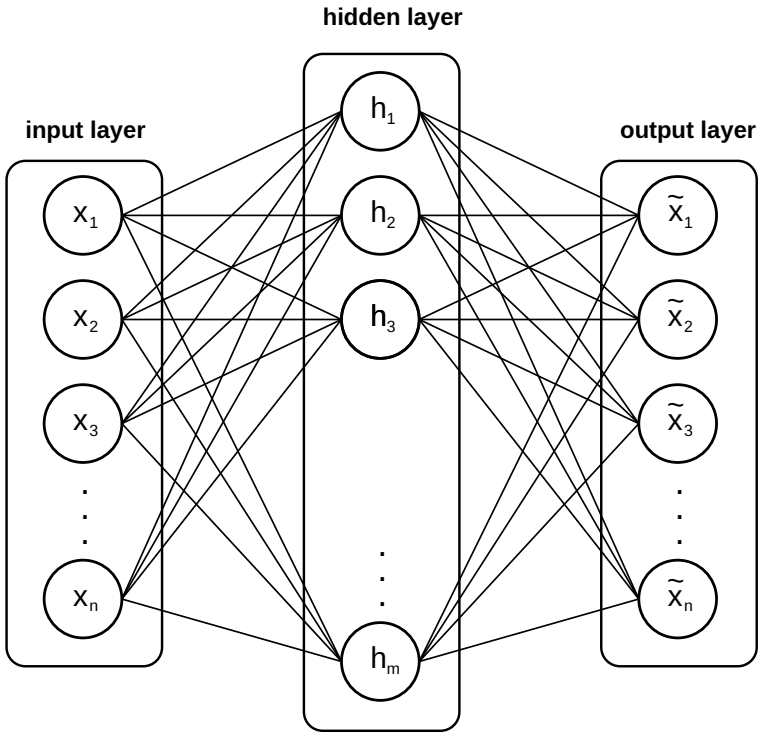
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

we've seen before:



A standard autoencoder has too much capacity

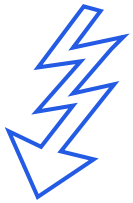
overcomplete autoencoder: $m = \dim(\mathbf{h}) \geq \dim(\mathbf{x}) = n$



Autoencoder Architectures

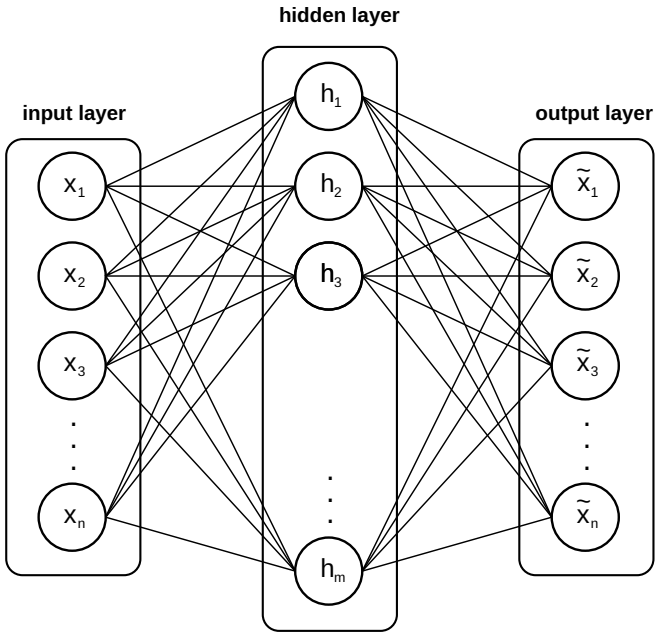
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

we've seen before:



A standard autoencoder has too much capacity

overcomplete autoencoder: $m = \dim(\mathbf{h}) \geq \dim(\mathbf{x}) = n$



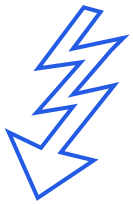
such an autoencoder could learn a **trivial encoding** (identity coding, copying) by simply copying **x** into **h** and then copying **h** into $\tilde{\mathbf{x}}$

$\Rightarrow$ useless unless used with regularization: does not tell us anything about important characteristics of the data but learns to represent each training example $x^{(i)}$

Autoencoder Architectures

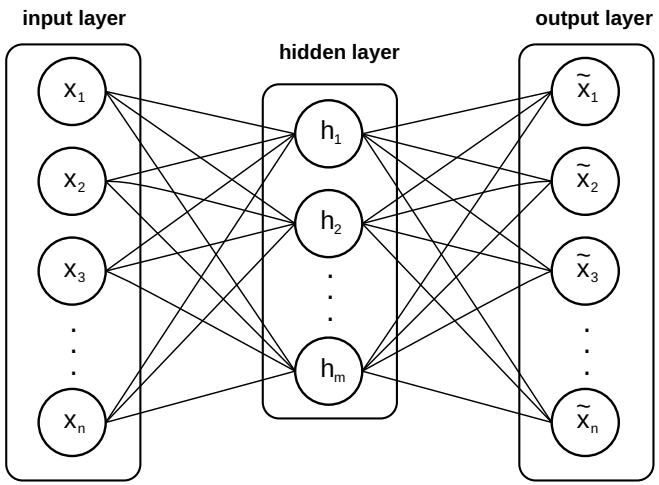
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

we've seen before:



A standard autoencoder has too much capacity

undercomplete autoencoder: $m = \dim(\mathbf{h}) < \dim(\mathbf{x}) = n$



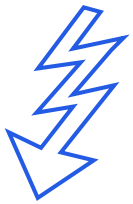
such an autoencoder actually **learns**:

if we are still able to reconstruct $\tilde{\mathbf{x}}$ perfectly from $\mathbf{h}$, this tells us that h is a loss-free encoding of $\mathbf{x}$: it captures all the important characteristics of $\mathbf{x}$

Autoencoder Architectures

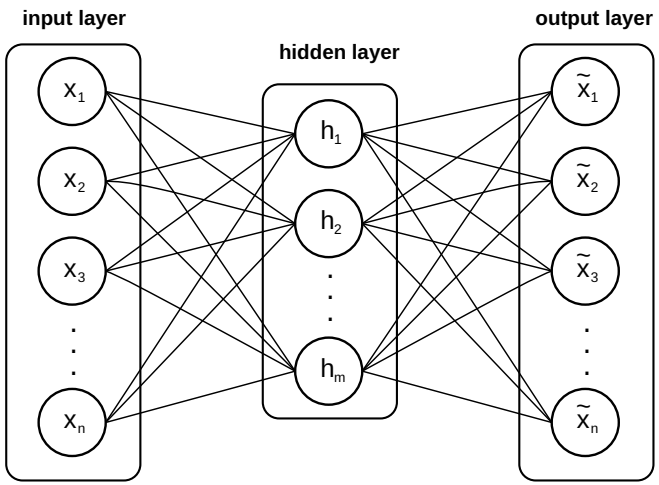
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

we've seen before:



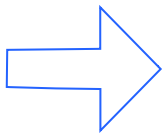
A standard autoencoder has too much capacity

undercomplete autoencoder: $m = \dim(\mathbf{h}) < \dim(\mathbf{x}) = n$



such an autoencoder actually **learns**:

if we are still able to reconstruct $\tilde{\mathbf{x}}$ perfectly from $\mathbf{h}$, this tells us that h is a loss-free encoding of $\mathbf{x}$: it captures all the important characteristics of $\mathbf{x}$



adjust hidden layer dimensionality

Autoencoder Architectures

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Hidden Layer Dimensionality

undercomplete
will compress the data as it must discard some information in **h**

overcomplete
must be used with regularization to avoid trivial identity, otherwise no features are learned $\Rightarrow$ enforce a sparsity constraint

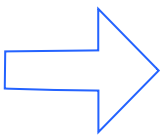
Autoencoder Architectures

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Hidden Layer Dimensionality

undercomplete
will compress the data as it must discard some information in **h**

overcomplete
must be used with regularization to avoid trivial identity, otherwise no features are learned $\Rightarrow$ enforce a sparsity constraint



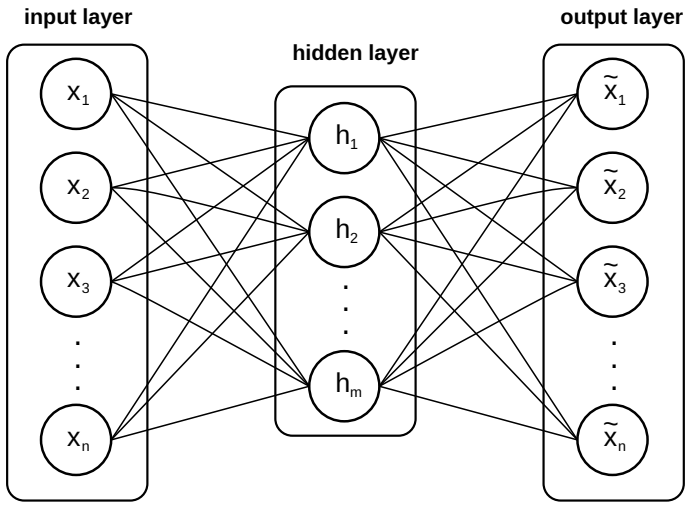
autoencoder architectures by the choice of the hidden layer(s)

Autoencoder Architectures

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

the simplest autoencoder:
encode with a fully connected network (FCN) as the bottleneck
undercomplete autoencoder: $m = \dim(\mathbf{h}) < \dim(\mathbf{x}) = n$

create a **bottleneck** with e.g. 10^4 nodes at the input and output layer and (only) 20 nodes at the hidden layer



Deep Autoencoders

this concept can be enhanced:
Deep (Nonlinear) Autoencoders (G. Hinton, R. Salakhutdinov)

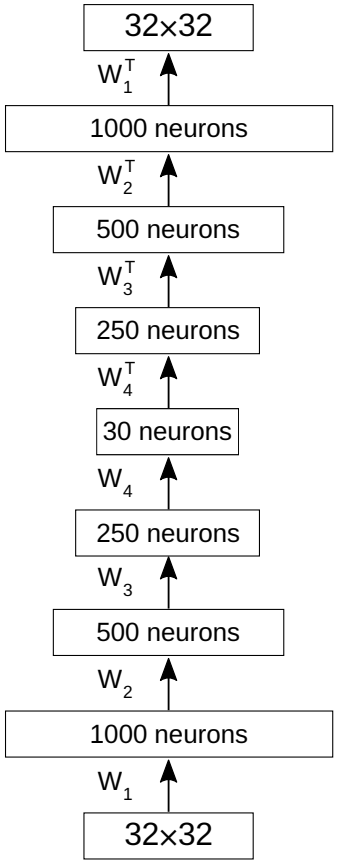
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Deep Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

this concept can be enhanced:

Deep (Nonlinear) Autoencoders (G. Hinton, R. Salakhutdinov)



deep (nonlinear) autoencoders learn to project the data not onto a subspace but onto a nonlinear manifold

this manifold is the image of the decoder

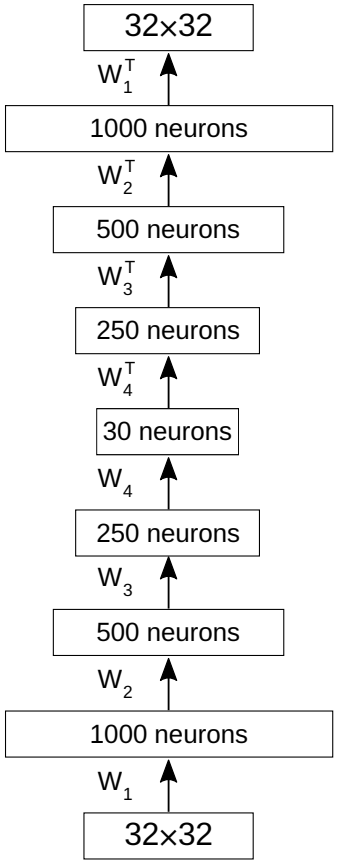
$\Rightarrow$ nonlinear dimensionality reduction which is more powerful than linear encoders (PCA) with same dimensionality

Deep Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

this concept can be enhanced:

Deep (Nonlinear) Autoencoders (G. Hinton, R. Salakhutdinov)



deep (nonlinear) autoencoders learn to project the data not onto a subspace but onto a nonlinear manifold

this manifold is the image of the decoder

$\Rightarrow$ nonlinear dimensionality reduction which is more powerful than linear encoders (PCA) with same dimensionality

however: difficult to optimize with backpropagation - require layer-wise training

Deep Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Deep Autoencoders (DAE) as Generative Model:

DAEs implicitly estimate the underlying data-generating process

intuition: If $\mathbf{x}$ is a typical sample, then iteratively applying the noise and the denoising will reproduce the sample very often

estimator $p(\mathbf{x})$: Markov chain sampling by alternating denoising model $p(\mathbf{x}|\hat{\mathbf{x}})$ and corruption process $C(\hat{\mathbf{x}}|\mathbf{x})$

converging MC yields estimator $p(\mathbf{x})$

often expensive and hard to assess convergence
$\Rightarrow$ **variational autoencoders** much more common

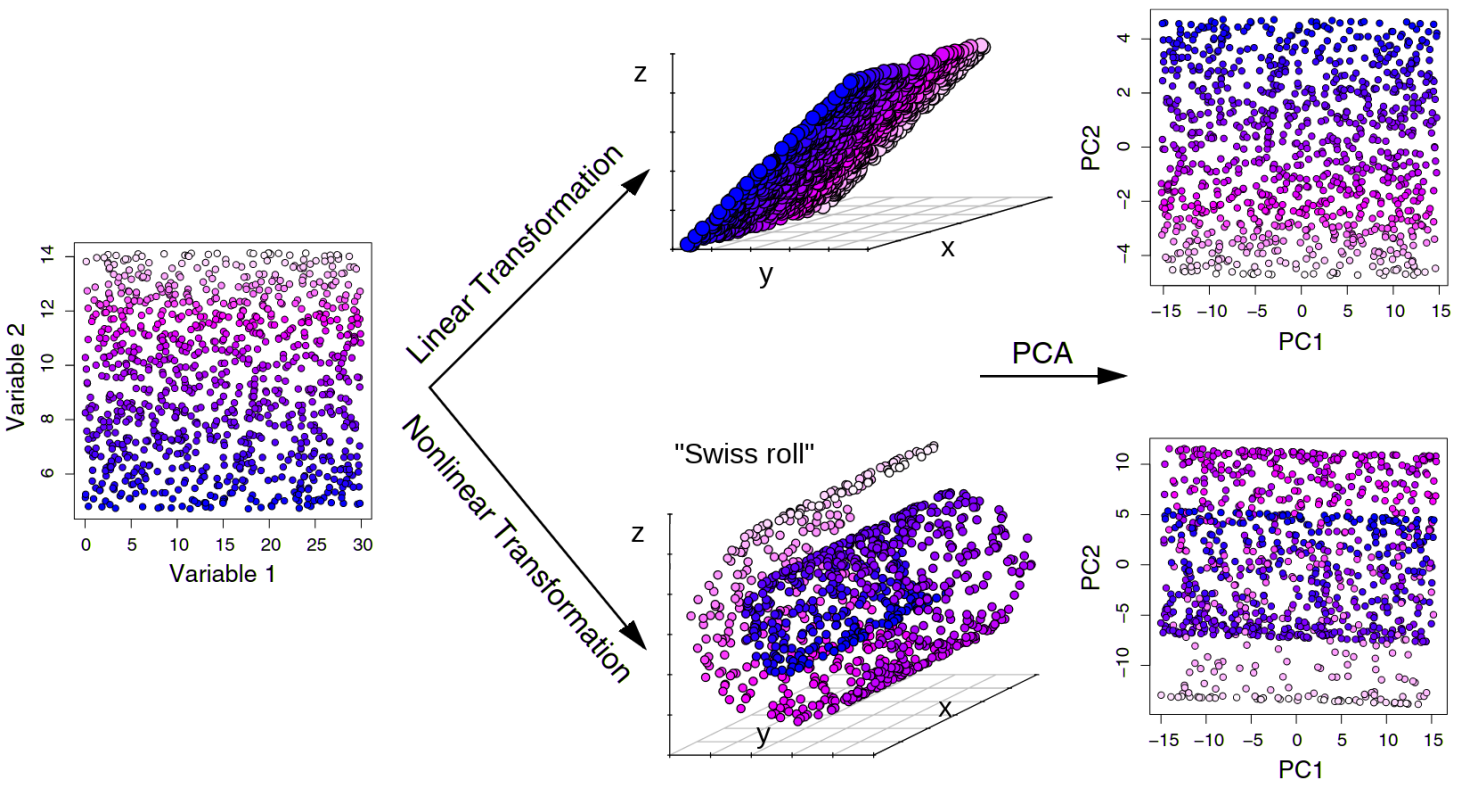
more on this later on

Undercomplete Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

The primary use of autoencoders is the generation of the latent space or the bottleneck, a compressed substitute of the input data. This form of **non-linear dimensionality reduction** where the autoencoder learns a non-linear manifold is also called **manifold learning**.

PCA cannot solve this

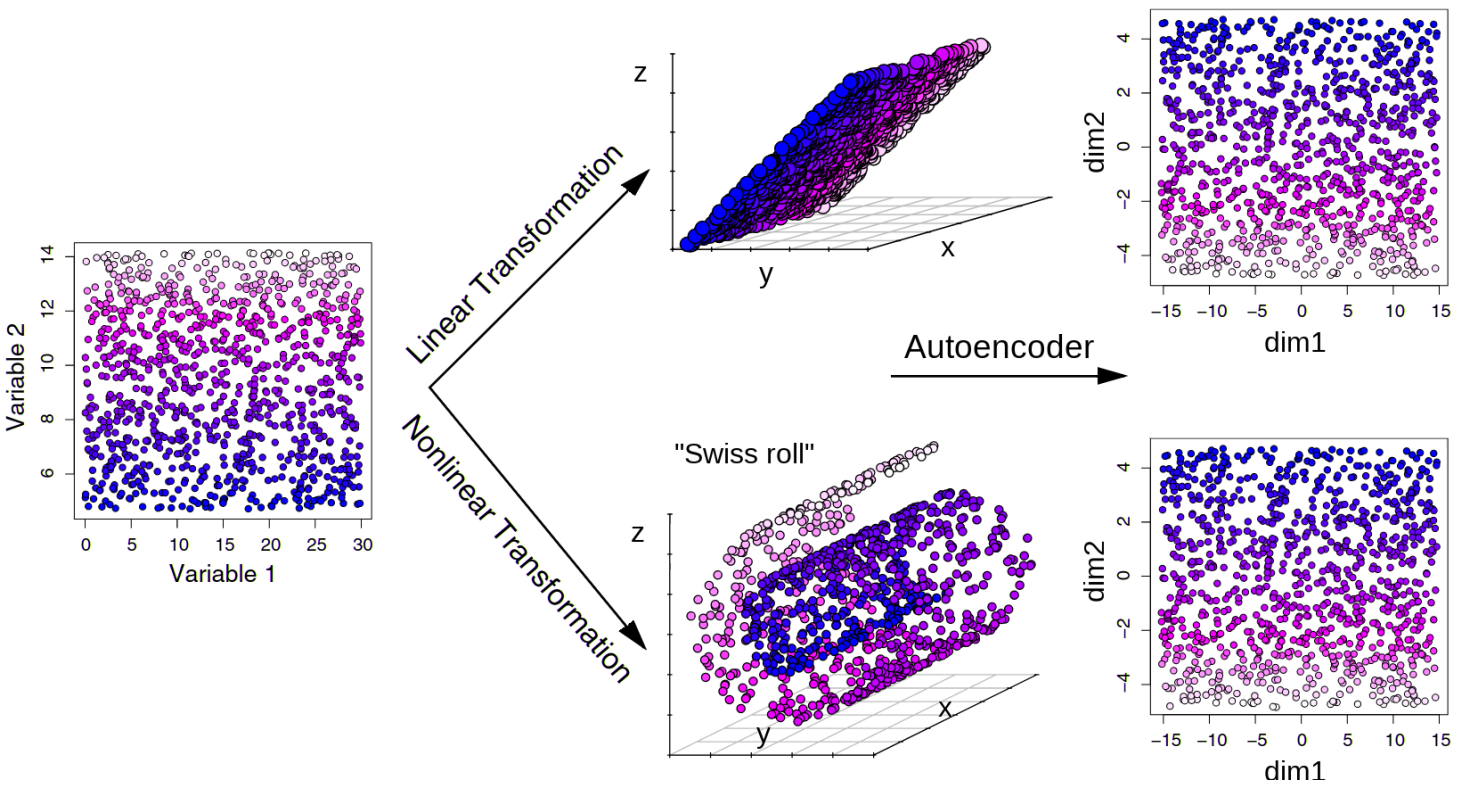


Undercomplete Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

The primary use of autoencoders is the generation of the latent space or the bottleneck, a compressed substitute of the input data. This form of **non-linear dimensionality reduction** where the autoencoder learns a non-linear manifold is also called **manifold learning**.

PCA cannot solve this, but an undercomplete autoencoder can:

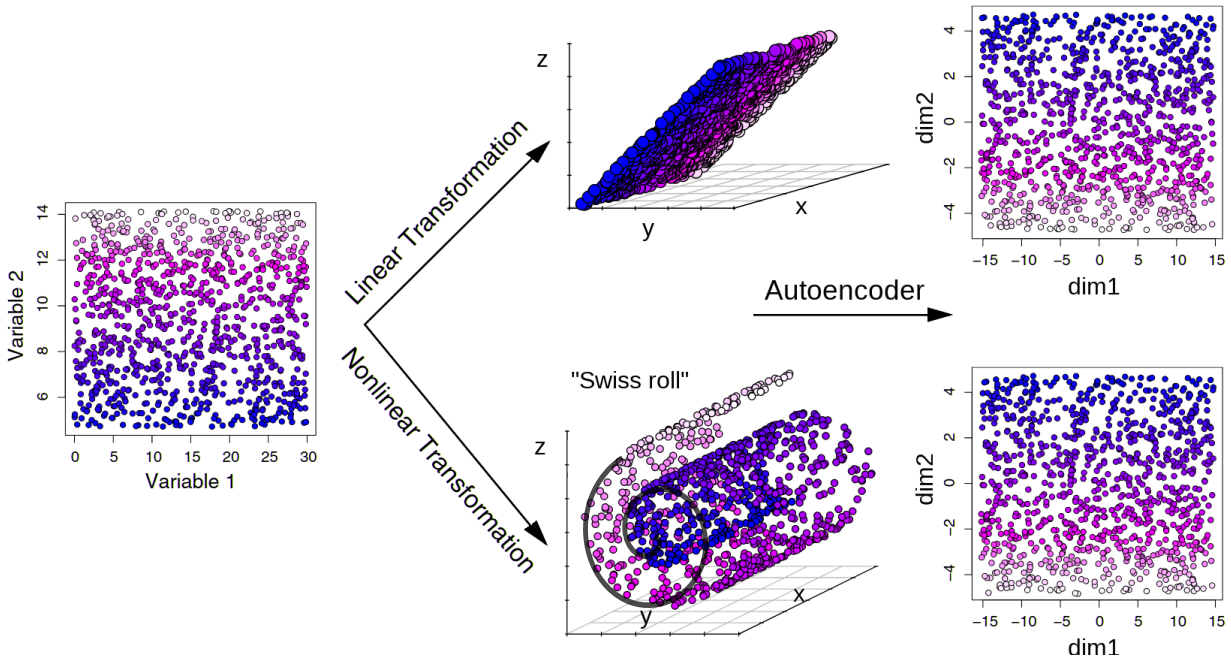


Undercomplete Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

The primary use of autoencoders is the generation of the latent space or the bottleneck, a compressed substitute of the input data. This form of **non-linear dimensionality reduction** where the autoencoder learns a non-linear manifold is also called **manifold learning**.

PCA cannot solve this, but an undercomplete autoencoder can:



autoencoder preserves local structure of data

Undercomplete Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

The loss function used to train an undercomplete autoencoder is called **reconstruction loss**, as it is a check of how well the image has been reconstructed from the input.

Undercomplete Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

The loss function used to train an undercomplete autoencoder is called **reconstruction loss**, as it is a check of how well the image has been reconstructed from the input.

Minimizing the loss function $E(\mathbf{x}, g(f(\mathbf{x})))$, such as e.g. the L_2 loss function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{N}(\text{true} - \text{reconstruced})^2 = \frac{1}{2}\sum_{k=1}^{N}(\mathbf{x}_k - g(f(\mathbf{x}_k)))^2$$

Undercomplete Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

The loss function used to train an undercomplete autoencoder is called **reconstruction loss**, as it is a check of how well the image has been reconstructed from the input.

Minimizing the loss function $E(\mathbf{x}, g(f(\mathbf{x})))$, such as e.g. the L_2 loss function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{N}(\text{true} - \text{reconstruced})^2 = \frac{1}{2}\sum_{k=1}^{N}(\mathbf{x}_k - g(f(\mathbf{x}_k)))^2$$

In the case of inputs represented as bit vectors the cross-entropy can be used:

$$E(\mathbf{w}) = \sum_{k=1}^{N}\sum_{\text{bits}}\mathbf{x}_k \cdot \log(g(f(\mathbf{x}_k)))$$

Undercomplete Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Applications:

- dimensionality reduction
- visualization
- feature extraction: feature importance
- color reduction (which is dimensionality reduction): how to best preserve the image

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

While undercomplete autoencoders are **regulated and fine-tuned** by the size of the bottleneck, and typical regularizers use a penalty on the size of the weights at the nodes, the sparse autoencoder is regulated by **changing the number of nodes at each hidden layer**.

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

While undercomplete autoencoders are **regulated and fine-tuned** by the size of the bottleneck, and typical regularizers use a penalty on the size of the weights at the nodes, the sparse autoencoder is regulated by **changing the number of nodes at each hidden layer**.

Cannot be done directly - sparse autoencoders work by **penalizing the activation** of some neurons in hidden layers:

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

While undercomplete autoencoders are **regulated and fine-tuned** by the size of the bottleneck, and typical regularizers use a penalty on the size of the weights at the nodes, the sparse autoencoder is regulated by **changing the number of nodes at each hidden layer**.

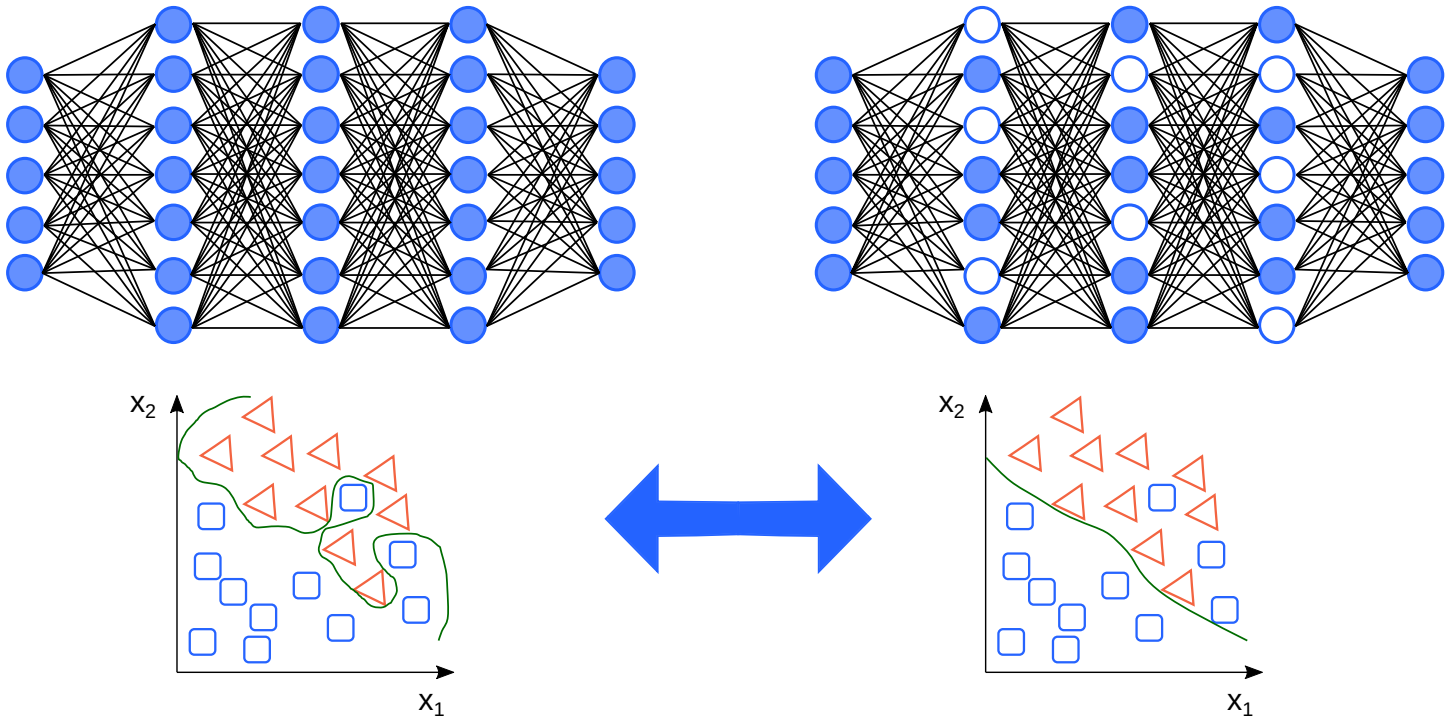
Cannot be done directly - sparse autoencoders work by **penalizing the activation** of some neurons in hidden layers:

remember: dropout regularization (lecture 4)

recap: Dropout Regularization

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Dropout turns off random neurons with probability p for each mini-batch during training (forward pass).
This forces the network to have redundant representation, preventing co-adaptation for features.



model has too many free parameters and over-fits

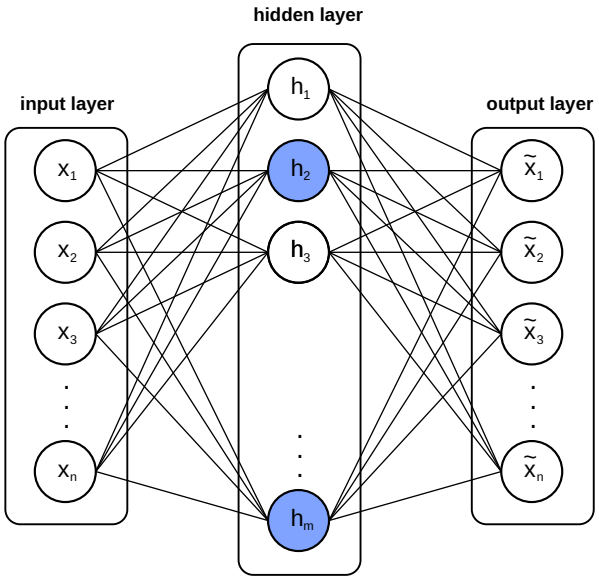
too many neurons dropped out leading to underfitting

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

While undercomplete autoencoders are **regulated and fine-tuned** by the size of the bottleneck, and typical regularizers use a penalty on the size of the weights at the nodes, the sparse autoencoder is regulated by **changing the number of nodes at each hidden layer**.

cannot be done directly - sparse autoencoders work by **penalizing the activation** of some neurons in hidden layers



- builds on an overcomplete autoencoder (no bottleneck yet)
- the bottleneck is created by enforcing sparsity on the activation

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

penalizing the activation of some neurons in hidden layers:

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

penalizing the activation of some neurons in hidden layers:

The average activation value of the activation of a neuron l is given by

$$\tilde{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

penalizing the activation of some neurons in hidden layers:

The average activation value of the activation of a neuron l is given by

$$\tilde{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

A sparse autoencoder uses a **sparsity parameter** $\rho \sim 0$, e.g. $\rho = 0.005$ and tries to enforce the constraint $\tilde{\rho}_l = \rho$.

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

penalizing the activation of some neurons in hidden layers:

The average activation value of the activation of a neuron l is given by

$$\tilde{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

A sparse autoencoder uses a **sparsity parameter** $\rho \sim 0$, e.g. $\rho = 0.005$ and tries to enforce the constraint $\tilde{\rho}_l = \rho$.
We ensure this by adding the following term to the objective function:

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\tilde{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \tilde{\rho}_l}$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

penalizing the activation of some neurons in hidden layers:

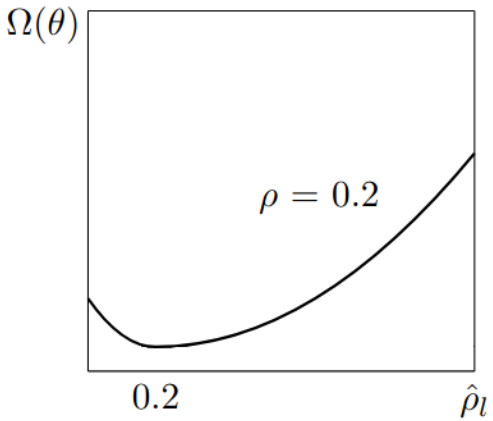
The average activation value of the activation of a neuron l is given by

$$\tilde{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(x_i)_l$$

A sparse autoencoder uses a **sparsity parameter** $\rho \sim 0$, e.g. $\rho = 0.005$ and tries to enforce the constraint $\tilde{\rho}_l = \rho$.
We ensure this by adding the following term to the objective function:

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\tilde{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\tilde{\rho}_l}$$

$\Omega(\theta)$ will reach its minimum for $\tilde{\rho}_l = \rho$.



Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We start with

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\tilde{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\tilde{\rho}_l}$$

$$= \sum_{l=1}^{k} \rho \log \rho - \rho \log \tilde{\rho}_l + (1-\rho) \log(1-\rho) \log(1-\tilde{\rho}_l)$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We start with

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\tilde{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\tilde{\rho}_l}$$

$$= \sum_{l=1}^{k} \rho \log \rho - \rho \log \tilde{\rho}_l + (1-\rho) \log(1-\rho) \log(1-\tilde{\rho}_l)$$

With the chain rule, we get:

$$\frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \tilde{\rho}} \cdot \frac{\partial \tilde{\rho}}{\partial W}$$

$$\frac{\partial \Omega(\theta)}{\partial \tilde{\rho}} = \left[\frac{\partial \Omega(\theta)}{\partial \tilde{\rho}_1}, \ldots, \frac{\partial \Omega(\theta)}{\partial \tilde{\rho}_k} \right]^T$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We start with

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\tilde{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\tilde{\rho}_l}$$

$$= \sum_{l=1}^{k} \rho \log \rho - \rho \log \tilde{\rho}_l + (1-\rho) \log(1-\rho) \log(1-\tilde{\rho}_l)$$

With the chain rule, we get:

$$\frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \tilde{\rho}} \cdot \frac{\partial \tilde{\rho}}{\partial W}$$

$$\frac{\partial \Omega(\theta)}{\partial \tilde{\rho}} = \left[\frac{\partial \Omega(\theta)}{\partial \tilde{\rho}_1}, ..., \frac{\partial \Omega(\theta)}{\partial \tilde{\rho}_k} \right]^T$$

For each neuron $l \in 1, ..., k$ in the hidden layer, we have

$$\frac{\partial \Omega(\theta)}{\partial \tilde{\rho}_l} = -\frac{\rho}{\tilde{\rho}_l} + \frac{1-\rho}{1-\tilde{\rho}_l}$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

In the same way, for each neuron $l \in 1, ..., k$ in the hidden layer, we have

$$\frac{\partial \tilde{\rho}}{\partial W} = \left[\frac{\partial \tilde{\rho}_1}{\partial W}, ..., \frac{\partial \tilde{\rho}_k}{\partial W}\right]$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

In the same way, for each neuron $l \in 1, ..., k$ in the hidden layer, we have

$$\frac{\partial \tilde{\rho}}{\partial W} = \left[\frac{\partial \tilde{\rho}_1}{\partial W}, ..., \frac{\partial \tilde{\rho}_k}{\partial W}\right]$$

For each element in this equation we can calculate $\frac{\partial \tilde{\rho}_l}{\partial W}$ (the partial derivative of a scalar w.r.t. a matrix is a matrix). For a single element W_{jl}:

$$\frac{\partial \tilde{\rho}_l}{\partial W_{jl}} = \frac{\partial \left[\frac{1}{m}\sum_{i=1}^{m} g(W_{:,l}^T x_i + b_l)\right]}{\partial W_{jl}}$$

$$= \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \left[g(W_{:,l}^T x_i + bl)\right]}{\partial W_{jl}} = \frac{1}{m}\sum_{i=1}^{m} g'(W_{:,l}^T x_i + b_l) x_{ij}$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

In the same way, for each neuron $l \in 1, ..., k$ in the hidden layer, we have

$$\frac{\partial \tilde{\rho}}{\partial W} = \left[\frac{\partial \tilde{\rho}_1}{\partial W}, ..., \frac{\partial \tilde{\rho}_k}{\partial W}\right]$$

For each element in this equation we can calculate $\frac{\partial \tilde{\rho}_l}{\partial W}$ (the partial derivative of a scalar w.r.t. a matrix is a matrix). For a single element W_{jl}:

$$\frac{\partial \tilde{\rho}_l}{\partial W_{jl}} = \frac{\partial \left[\frac{1}{m}\sum_{i=1}^{m} g(W_{:,l}^T x_i + b_l)\right]}{\partial W_{jl}}$$

$$= \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \left[g(W_{:,l}^T x_i + bl)\right]}{\partial W_{jl}} = \frac{1}{m}\sum_{i=1}^{m} g'(W_{:,l}^T x_i + b_l) x_{ij}$$

So in matrix notation, we can write this as:

$$\frac{\partial \tilde{\rho}_l}{\partial W} = x_i (g'(W^T x_i + \mathbf{b}))^T$$

Sparse Autoencoders

We've seen $\tilde{E}(\theta) = E(\theta) + \Omega(\theta)$ where $E(\theta)$ is the squared error loss or the cross-entropy loss, and $\Omega(\theta)$ is the sparsity constraint.

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We've seen $\tilde{E}(\theta) = E(\theta) + \Omega(\theta)$ where $E(\theta)$ is the squared error loss or the cross-entropy loss, and $\Omega(\theta)$ is the sparsity constraint.

From this, we finally get

$$\frac{\partial \tilde{E}(\theta)}{\partial W} = \frac{\partial E(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W}$$

Sparse Autoencoders

We've seen $\tilde{E}(\theta) = E(\theta) + \Omega(\theta)$ where $E(\theta)$ is the squared error loss or the cross-entropy loss, and $\Omega(\theta)$ is the sparsity constraint.

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We've seen $\tilde{E}(\theta) = E(\theta) + \Omega(\theta)$ where $E(\theta)$ is the squared error loss or the cross-entropy loss, and $\Omega(\theta)$ is the sparsity constraint.

From this, we finally get

$$\frac{\partial \tilde{E}(\theta)}{\partial W} = \frac{\partial E(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W}$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We've seen $\tilde{E}(\theta) = E(\theta) + \Omega(\theta)$ where $E(\theta)$ is the squared error loss or the cross-entropy loss, and $\Omega(\theta)$ is the sparsity constraint.

From this, we finally get

$$\frac{\partial \tilde{E}(\theta)}{\partial W} = \frac{\partial E(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W}$$

With our chain rule:

$$\frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \tilde{\rho}} \cdot \frac{\partial \tilde{\rho}}{\partial W}$$

$$\frac{\partial \tilde{\rho}}{\partial W} = \left[\frac{\partial \tilde{\rho}_1}{\partial W}, \ldots, \frac{\partial \tilde{\rho}_k}{\partial W}\right]$$

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

We've seen $\tilde{E}(\theta) = E(\theta) + \Omega(\theta)$ where $E(\theta)$ is the squared error loss or the cross-entropy loss, and $\Omega(\theta)$ is the sparsity constraint.

From this, we finally get

$$\frac{\partial \tilde{E}(\theta)}{\partial W} = \frac{\partial E(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W}$$

With our chain rule:

$$\frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \tilde{\rho}} \cdot \frac{\partial \tilde{\rho}}{\partial W}$$

$$\frac{\partial \tilde{\rho}}{\partial W} = \left[\frac{\partial \tilde{\rho}_1}{\partial W}, \ldots, \frac{\partial \tilde{\rho}_k}{\partial W}\right]$$

Where we have seen that in matrix notation, we can write this as:

$$\frac{\partial \tilde{\rho}_l}{\partial W} = x_i (g'(W^T x_i + \mathbf{b}))^T$$

Sparse Autoencoders

Why does this work?

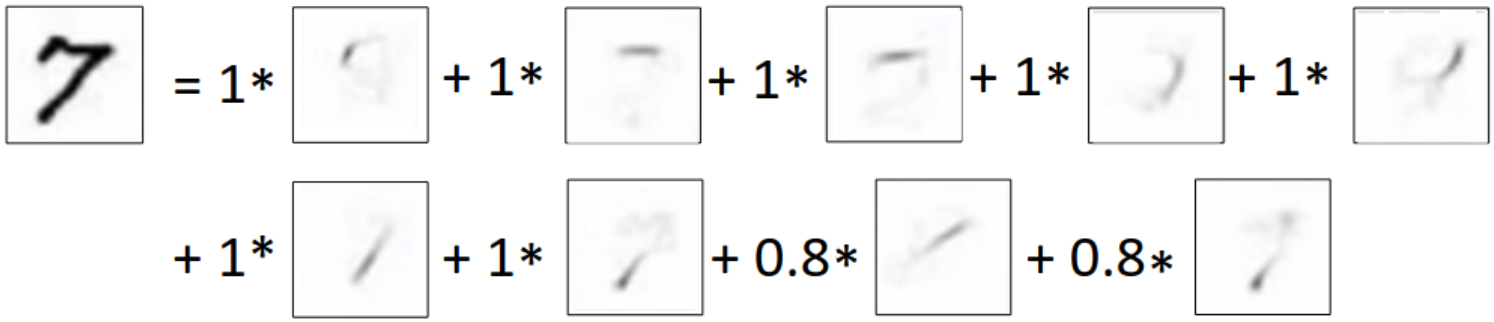
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Sparse Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Why does this work?

We want our learned features to be as sparse as possible - with sparse features we can **generalize better**.



An autoencoder that is regularized to be sparse must respond to unique statistical features of the training data set.

In this way, training to perform the copying task with a sparsity penalty can yield a model that has useful features.

Contractive Autoencoders

idea: similar inputs should have similar encodings

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Contractive Autoencoders

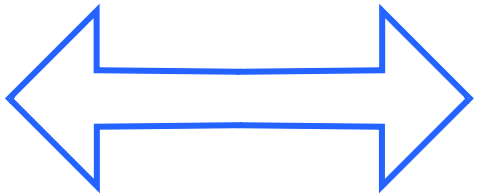
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

idea: similar inputs should have similar encodings

a regularization term is used to prevent an overcomplete autoencoder from learning the identity function

two terms that **contradict** each other:

reconstruction loss
differences between inputs



regularization term
enable model to ignore variations between inputs

Contractive Autoencoders

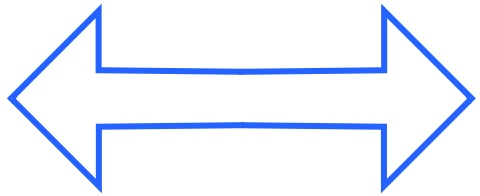
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

idea: similar inputs should have similar encodings

a regularization term is used to prevent an overcomplete autoencoder from learning the identity function

two terms that **contradict** each other:

reconstruction loss
differences between inputs



regularization term
enable model to ignore variations between inputs

These contradictory conditions in the loss function enable us to train a network where the hidden layers now capture only the most essential information.

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

implementation: similar to a sparse autoencoder, but uses the following regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m} \sum_{i=1}^{n} \left(\frac{\partial h_i}{\partial x_j} \right)^2$$

where $\|\cdot\|$ is the L_2 **norm** and $J_x(\mathbf{h})$ is the Jacobian of the encoder.

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

implementation: similar to a sparse autoencoder, but uses the following regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m}\sum_{i=1}^{n}\left(\frac{\partial h_i}{\partial x_j}\right)^2$$

where $\|\cdot\|$ is the L_2 **norm** and $J_x(\mathbf{h})$ is the Jacobian of the encoder.

square root of the sum of absolute squares of matrix elements,

$$\|A\|_F \equiv \sqrt{\sum_{j=1}^{m}\sum_{i=1}^{n}|a_{ij}|^2}$$

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

implementation: similar to a sparse autoencoder, but uses the following regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m}\sum_{i=1}^{n}\left(\frac{\partial h_i}{\partial x_j}\right)^2$$

where $\|\cdot\|$ is the L_2 **norm** and $J_x(\mathbf{h})$ is the Jacobian of the encoder.

If the input has n dimensions and the hidden layer has k dimensions, then:

$$J_x(\mathbf{h}) = \begin{vmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} \end{vmatrix}$$

The (j, i) entry of the Jacobian captures the variation in the output in the jth neuron with a small variation in the ith input.

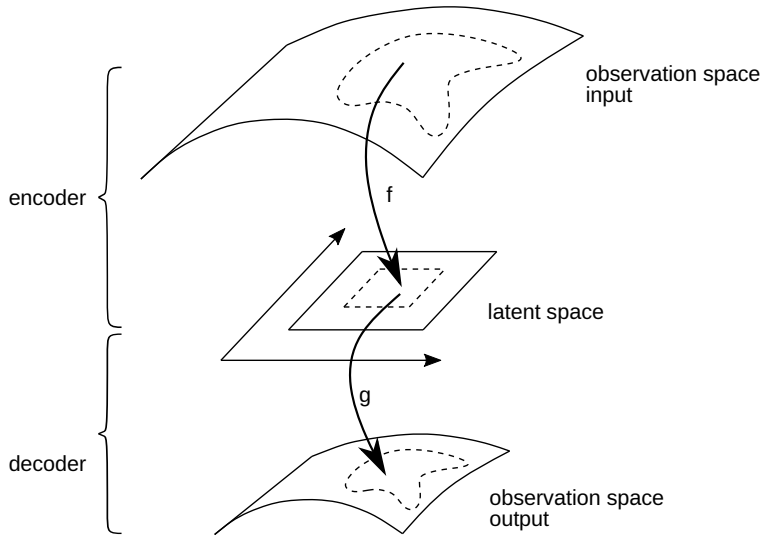
Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

implementation: similar to a sparse autoencoder, but uses the following regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m} \sum_{i=1}^{n} \left(\frac{\partial h_i}{\partial x_j} \right)^2$$

where $\|\cdot\|$ is the L_2 **norm** and $J_x(\mathbf{h})$ is the Jacobian of the encoder. Penalizing large partial derivatives of encoder outputs w.r.t. input values contracts the output space: mapping points in a neighborhood near x to a smaller output neighborhood near $f(x) \Rightarrow$ resists pertubations of input x.



Contractive Autoencoders

What is the **intuition** behind this?

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

What is the **intuition** behind this?

We have the regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m}\sum_{i=1}^{n}\left(\frac{\partial h_i}{\partial x_j}\right)^2$$

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

What is the **intuition** behind this?

We have the regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m}\sum_{i=1}^{n}\left(\frac{\partial h_i}{\partial x_j}\right)^2$$

Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$?

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

What is the **intuition** behind this?

We have the regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m}\sum_{i=1}^{n}\left(\frac{\partial h_i}{\partial x_j}\right)^2$$

Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$?
It means that this neuron is not very sensitive to variations in the input x_1.

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

What is the **intuition** behind this?

We have the regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m}\sum_{i=1}^{n}\left(\frac{\partial h_i}{\partial x_j}\right)^2$$

Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$?
It means that this neuron is not very sensitive to variations in the input x_1.

But doesn't this contradict our other goal of minimizing $E(\theta)$ which requires $\mathbf{h}$ to capture variations in the input?

Contractive Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

What is the **intuition** behind this?

We have the regularization term

$$\Omega(\mathbf{h}) = \|J_x(\mathbf{h})\|_F^2 = \sum_{j=1}^{m}\sum_{i=1}^{n}\left(\frac{\partial h_i}{\partial x_j}\right)^2$$

Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$?
It means that this neuron is not very sensitive to variations in the input x_1.

But doesn't this contradict our other goal of minimizing $E(\theta)$ which requires **h** to capture variations in the input?
Indeed it does, and this is **the idea**: By putting these two contradicting objectives against each other we ensure that **h** is sensitive to only very important variations as observed in the training data.

Denoising Autoencoders

idea: remove noise from an image by focusing on its important parts

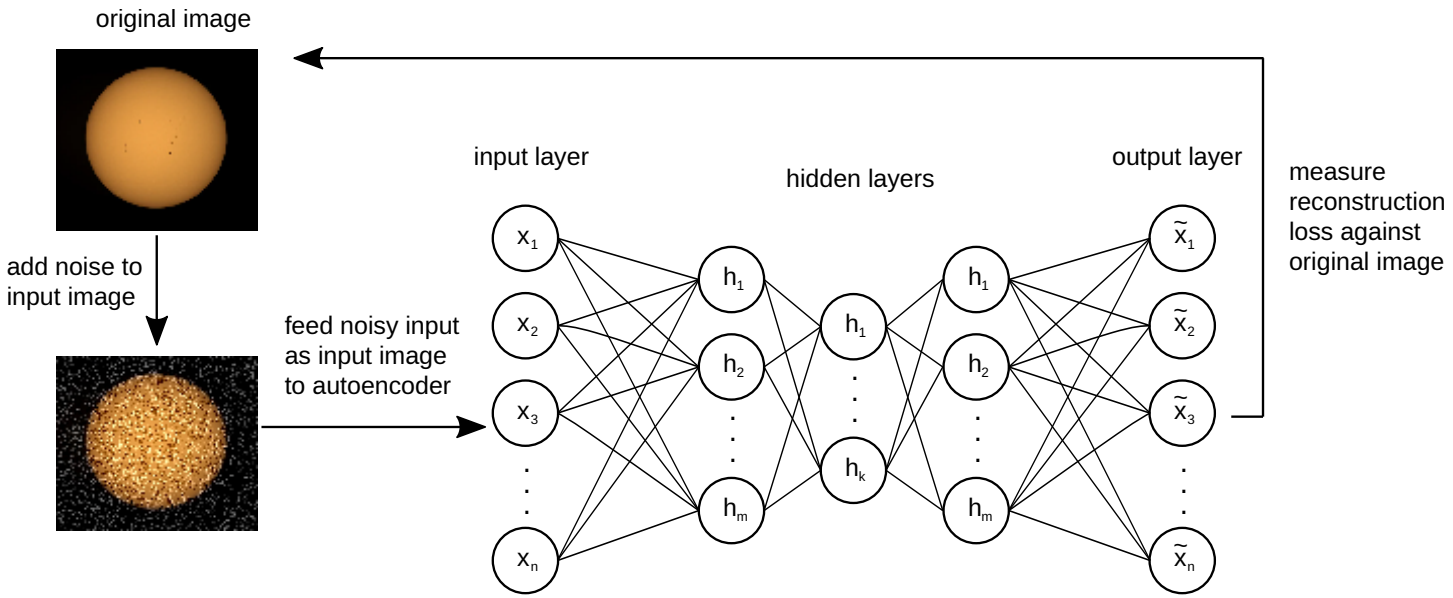
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Denoising Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

idea: remove noise from an image by focusing on its important parts

implementation: does not have the input image as its ground truth in training (ground truth: noise-free image)



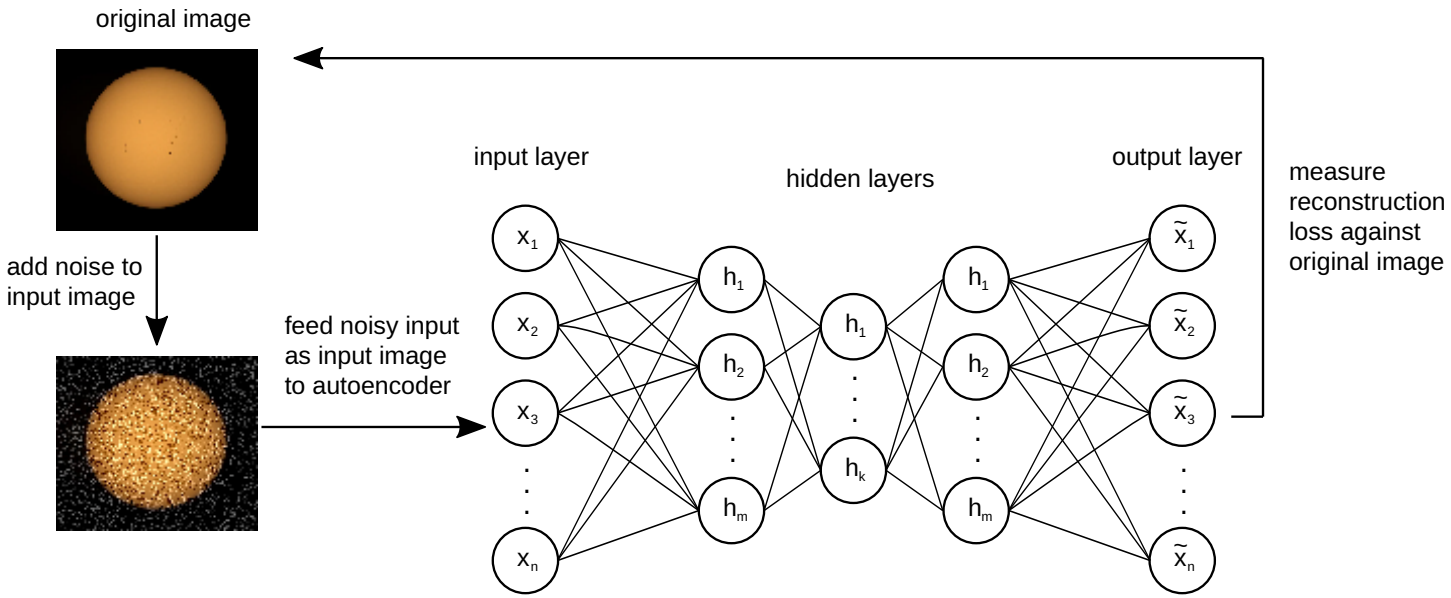
The autoencoder performs denoising by mapping the input data into a lower-dimensional manifold using non-linear dimensionality reduction (like in undercomplete autoencoders), where filtering of noise becomes much easier. The loss used is L2 or L1 loss.

Denoising Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

idea: remove noise from an image by focusing on its important parts

implementation: does not have the input image as its ground truth in training (ground truth: noise-free image)



Note: different noise is added during each epoch.

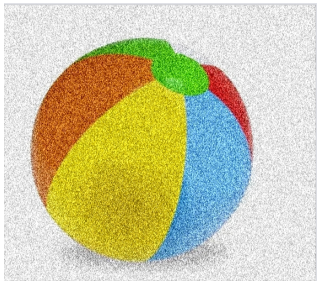
Denoising Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

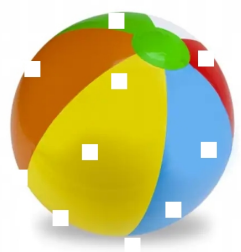
A denoising autoencoder corrupts the input data using a probabilistic process $P(x'_{ij}|x_{ij})$ before feeding it to the network.

example noise models:

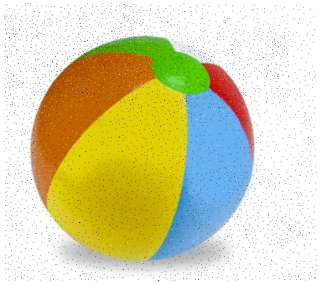
Gaussian noise: $z \sim N(0, \sigma^2)$



Masking noise: zero out some fraction of the image



Salt-and-pepper noise: chose some fraction of components of the image and set each to its min or max value (equally likely)



Denoising Autoencoders

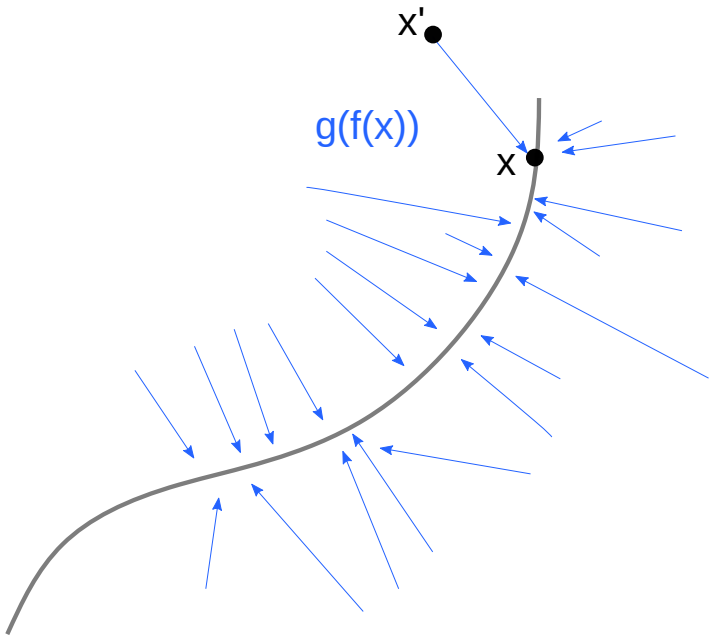
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Why does it work?

Even though images are in a high-dimensional space (length $\times$ width $\times$ color depth $\times$ channels), they lie on a low-dimensional manifold that captures their most important features

The corruption process of adding noise moves instance x off the manifold to x'.

Encoder f and decoder g are trained to project x' back onto the manifold.

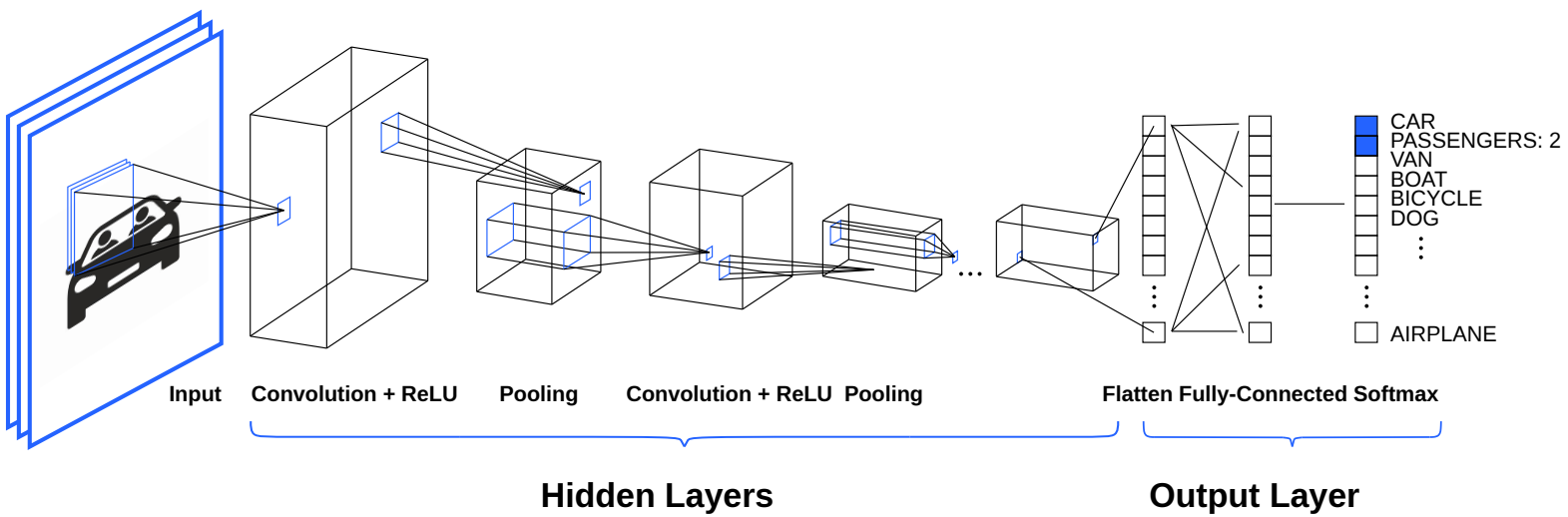


Convolutional Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

recap: A Convolutional Neural Network (CNN) consists of alternating **Convolutional layers** and **Pooling layers** that extract features.
Finally, their output is flattened into a one-dimensional feature layer that is passed into a Multi-Layer Perceptron (fully-connected layers) to obtain a prediction.

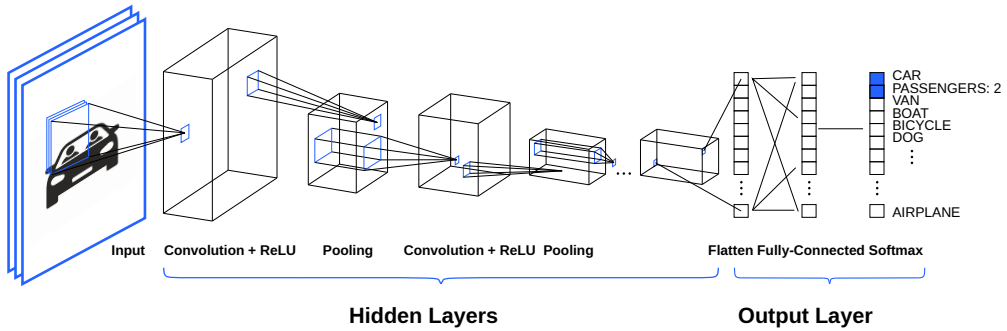
In this way they preserve the input image's spatial and temporal dependencies, mimicking how animal (human) vision works.



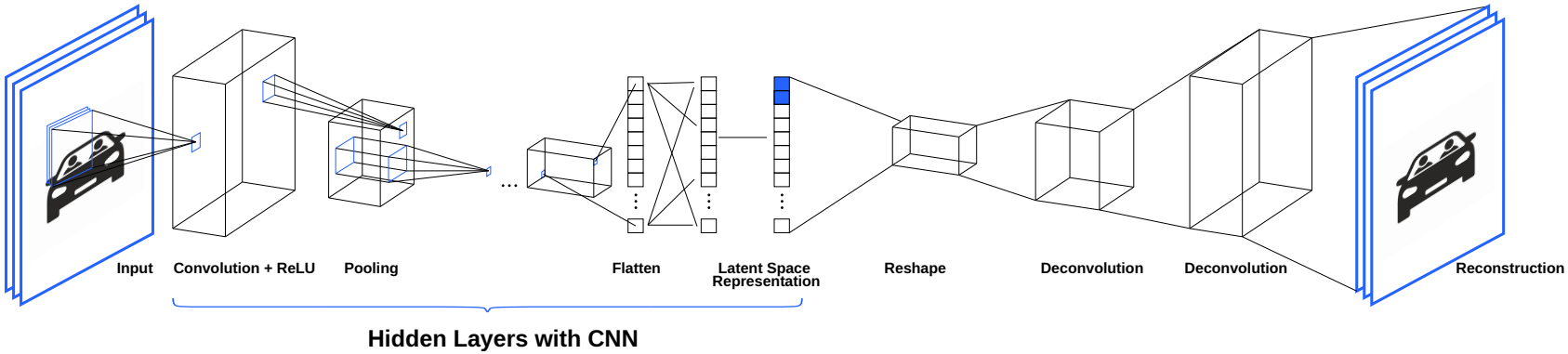
Convolutional Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Convolutional Neural Network (CNN)



Convolutional Autoencoder (CAE)



In a CAE, the CNN is used in the encoding and decoding parts of the autoencoder.

Convolutional Autoencoders

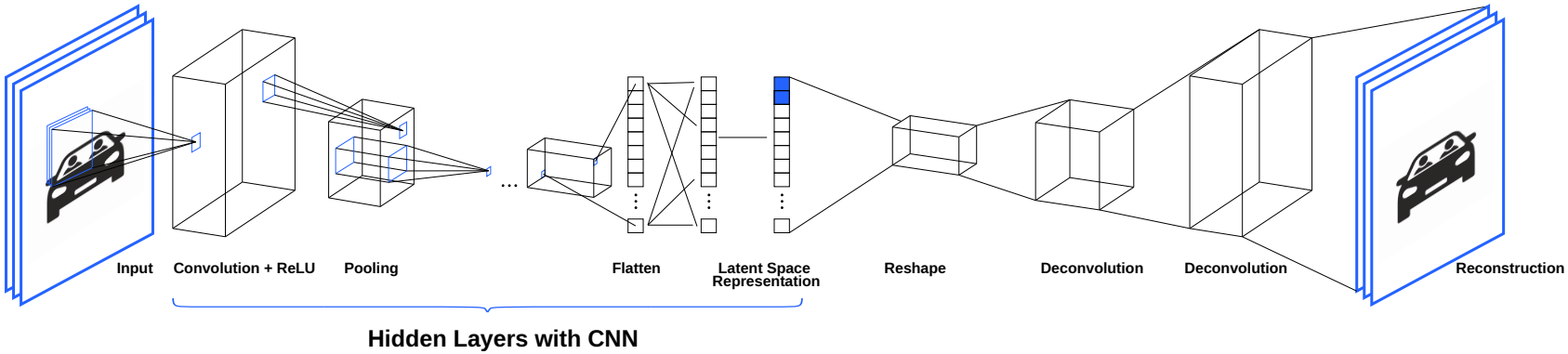
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Why does it work?

In image data, we see a huge loss of information when slicing and stacking the data.

In a CNN, features are extracted in convolution layers and are then passed unto linear layers which perform the actual classification. In a CAE, the features are fed into the Autoencoder part instead. In this way, Convolution Autoencoders retain spatial information of the input image data.

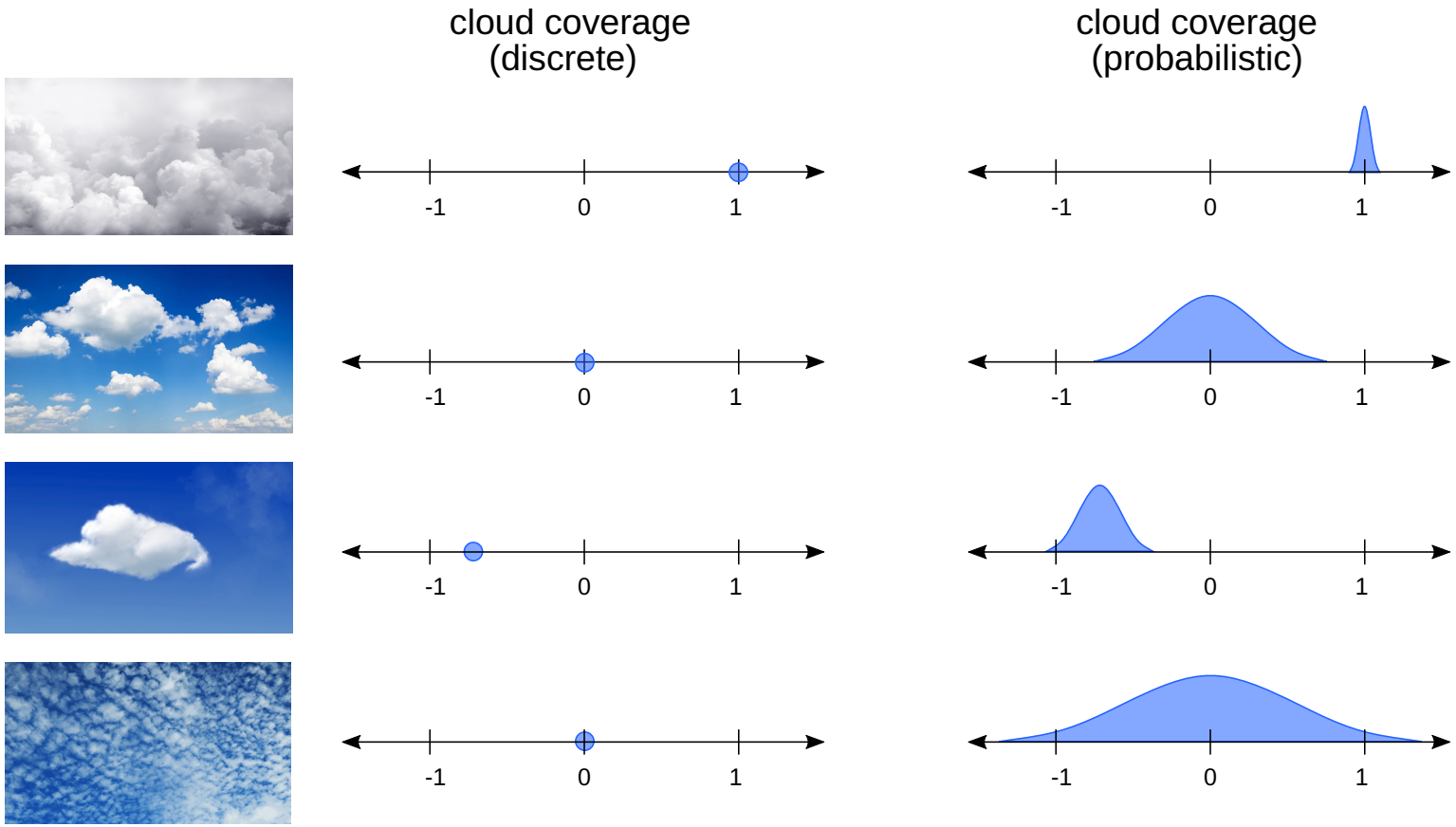
Convolutional Autoencoder (CAE)



Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

idea: Whereas "traditional" autoencoders compute a deterministic feature vector describing attributes of the input in latent space, variational autoencoders describe the latent space in probabilistic manner.
This allows to model uncertainty in the input data.



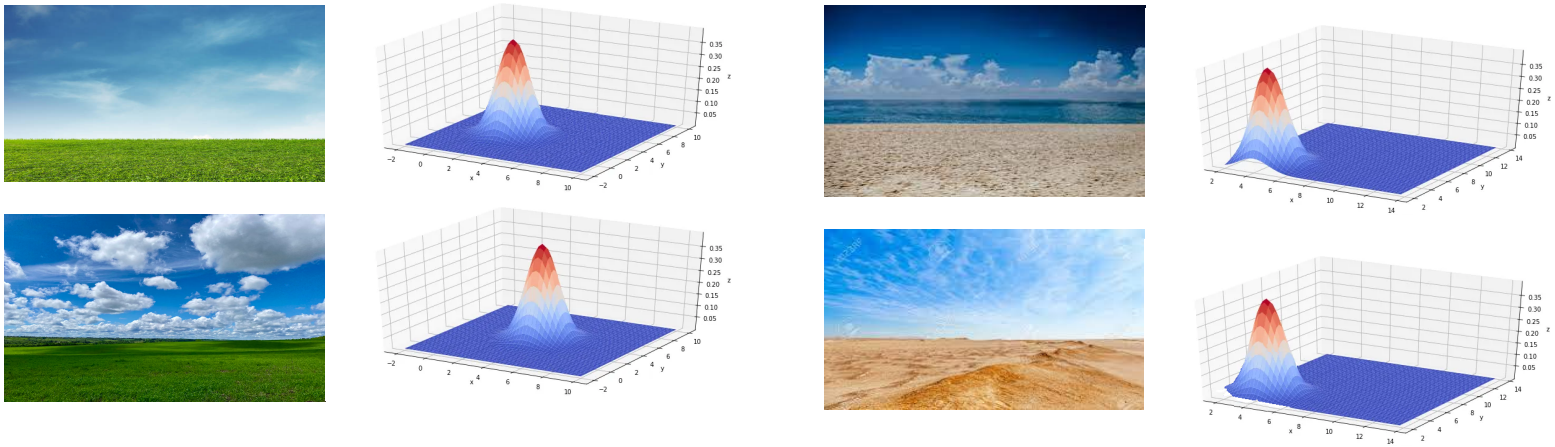
Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

The latent space isn't 1D:
representation as probability distribution enforces a continuous, smooth latent space representation

similar latent space **vectors** should correspond to similar reconstructions

aridity vs. cloud coverage



Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

more technical:

assumption: a hidden latent variable z that generates some observation x
$\Rightarrow$ train the variational autoencoder by determining the distribution of z

Variational Autoencoders

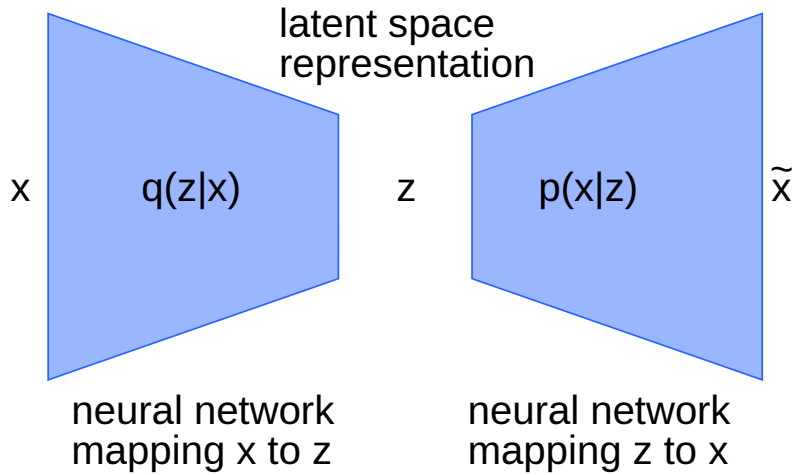
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

more technical:

assumption: a hidden latent variable z that generates some observation x
$\Rightarrow$ train the variational autoencoder by determining the distribution of z

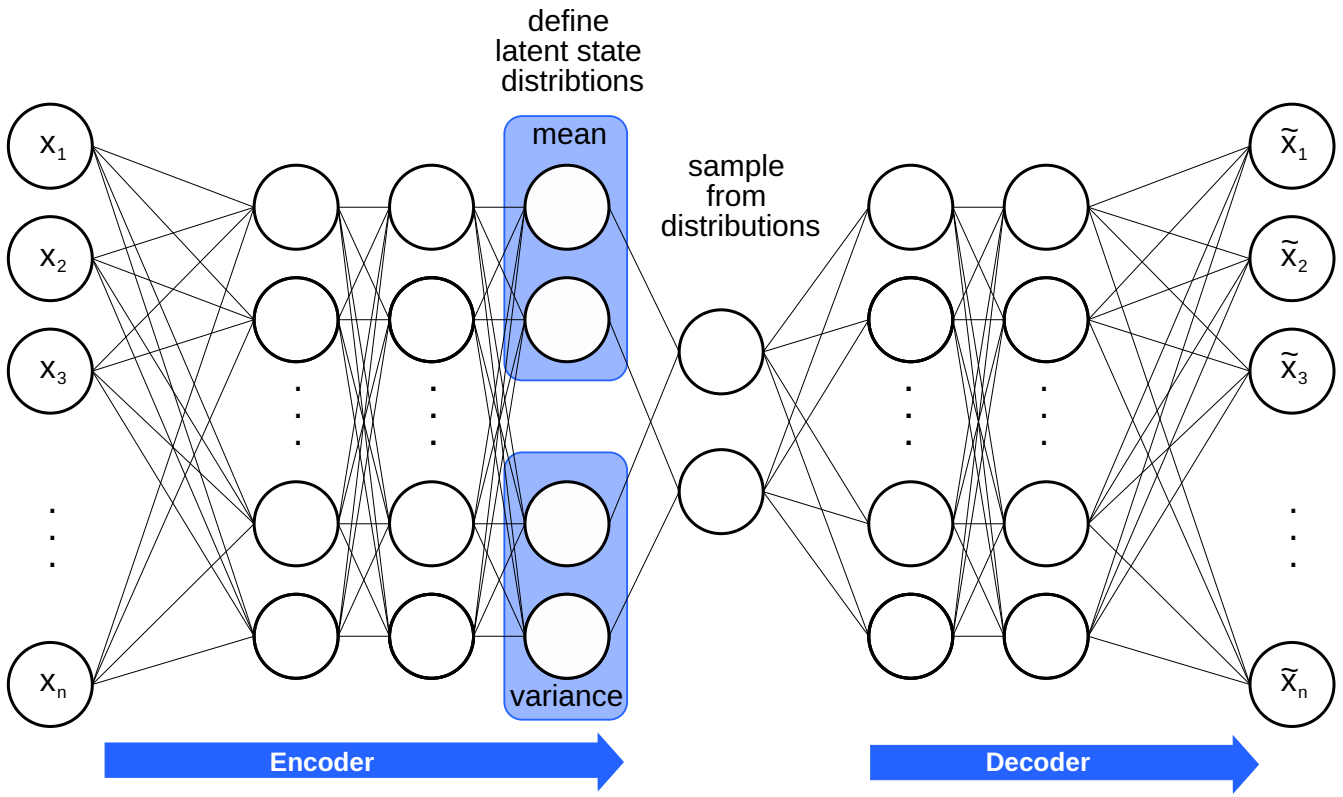
problem: computation of distribution $p(z|x)$ is usually intractable $\Rightarrow$
solution: approximate $p(z|x)$ by a tractable distribution $q(z|x)$

determine parameters for q, e.g. from an (isotropic) Gaussian distribution (parameter vectors μ and σ) by using neural network to estimate $q(z|x)$ and $p(x|z)$



Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications



VAE architecture differs from classic AE regarding the bottleneck: In AEs, the encoder converts high-dimensional input data to low-dimensional latent representation. In VAEs, the encoder learns the mean vector and the standard deviation diagonal matrix such that the latent representation follows a Gaussian distribution $z \sim \mathcal{N}(\mu, \sigma)$.

Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

defining the **Loss Function**:

In variational autoencoders, the loss function is composed of a reconstruction term on the final layer (that makes the encoding-decoding scheme efficient) and a regularisation term on the latent layer (that makes the latent space regular).

Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

defining the **Loss Function**:

In variational autoencoders, the loss function is composed of a reconstruction term on the final layer (that makes the encoding-decoding scheme efficient) and a regularisation term on the latent layer (that makes the latent space regular).

regularisation term: expressed as the **Kullback-Leibler Divergence** D_{KL} between $p(z|x)$ and $q(z|x)$ to force $q(z|x)$ to be similar to the true prior distribution $p(z|x)$

$D_{\mathrm{KL}}(P, Q)$ is a type of statistical distance:
a measure of how a probability distribution P differs from a reference probability distribution Q.
Closed form for two Gaussian distributions P and Q:

$$D_{\mathrm{KL}}(P, Q) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

defining the **Loss Function**:

In variational autoencoders, the loss function is composed of a reconstruction term on the final layer (that makes the encoding-decoding scheme efficient) and a regularisation term on the latent laye (that makes the latent space regular).

regularisation term: expressed as the **Kullback-Leibler Divergence** D_{KL} between $p(z|x)$ and $q(z|x)$ to force $q(z|x)$ to be similar to the true prior distribution $p(z|x)$.

$\Rightarrow$ Loss function:

$$E(\theta, x, z) = ||x - \tilde{x}||^2 - D_{\mathrm{KL}}(q(z|x), p(x|z))$$

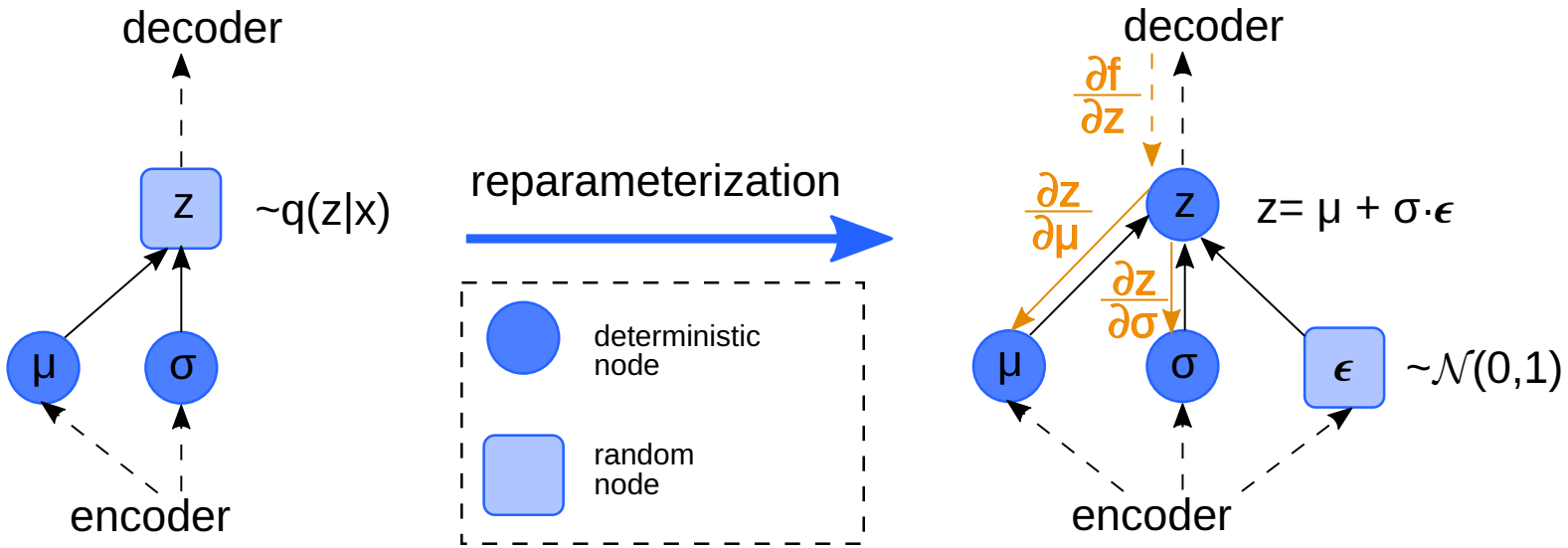
problem: network contains sampling operator $\Rightarrow$ we cannot backpropagate

Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

the solution: **reparameterization** enables deterministic backpropagation

idea: we reparameterize z in the forward-pass to be deterministically as $z = \mu + \sigma \cdot \epsilon$ instead of sampling from $q(z|x)$.



only ϵ is randomly sampled, and scaled by the latent distribution variance σ and shifted by its the mean $\mu \Rightarrow$ allows us to backpropagate into μ and σ

Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

For variational autoencoders, instead of encoding input as single point, we encode it as distribution over the latent space.

The model is then **trained** as follows:

1. the input is encoded as distribution over the latent space
2. a point from the latent space is sampled from that distribution
3. the sampled point is decoded and the reconstruction error can be computed
4. finally, the reconstruction error is backpropagated through the network

Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Variational Autoencoders can serve as **Generative Models**

new data can be generated by sampling from distributions in latent space, reconstructed by the decoder

example: smoothly varying degree of smile and head pose (Kingma & Welling 2013)

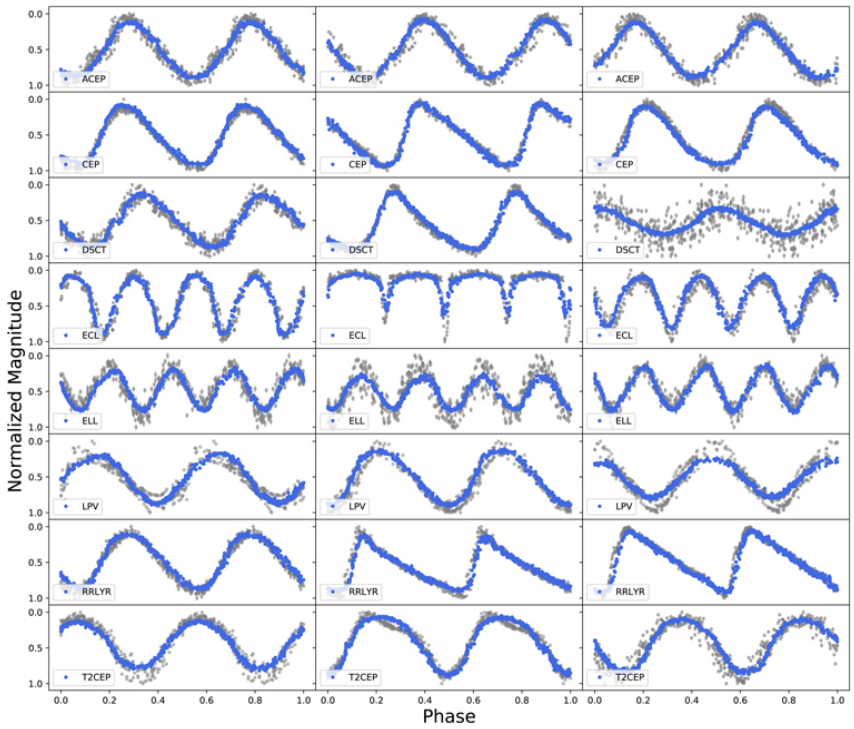
Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Variational Autoencoders can serve as **Generative Models**

new data can be generated by sampling from distributions in latent space, reconstructed by the decoder

astronomy example:
Deep Generative Modeling of Periodic Variable Stars Using Physical Parameters, J. Martìnez-Palomera, J. S. Bloom, E. S. Abrahams (2022)



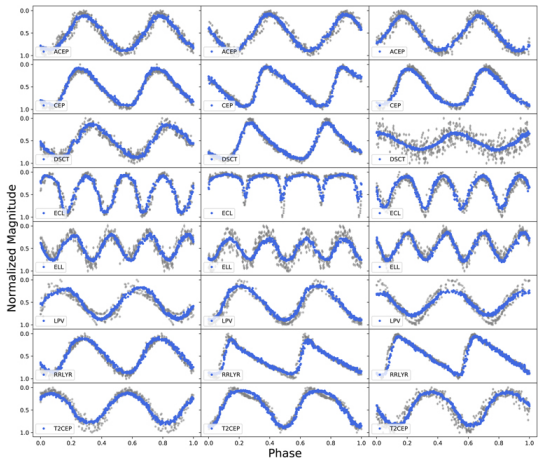
Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Variational Autoencoders can serve as **Generative Models**

new data can be generated by sampling from distributions in latent space, reconstructed by the decoder

astronomy example:
Deep Generative Modeling of Periodic Variable Stars Using Physical Parameters, J. Martìnez-Palomera, J. S. Bloom, E. S. Abrahams (2022)



generative modeling is an important application of variational autoencoders

Variational Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Summary:

- probabilistic models allow to generate data
- intractable density: optimize variational lower bound instead
- trained via backpropagation by using reparameterization

pros:

- principled approach to generative models
- latent space representation can be useful for other tasks: feature representation

cons:

- only maximizes lower bound of likelihood
- samples in standard models often of lower quality compared to Generative Adversarial Networks (GANs)

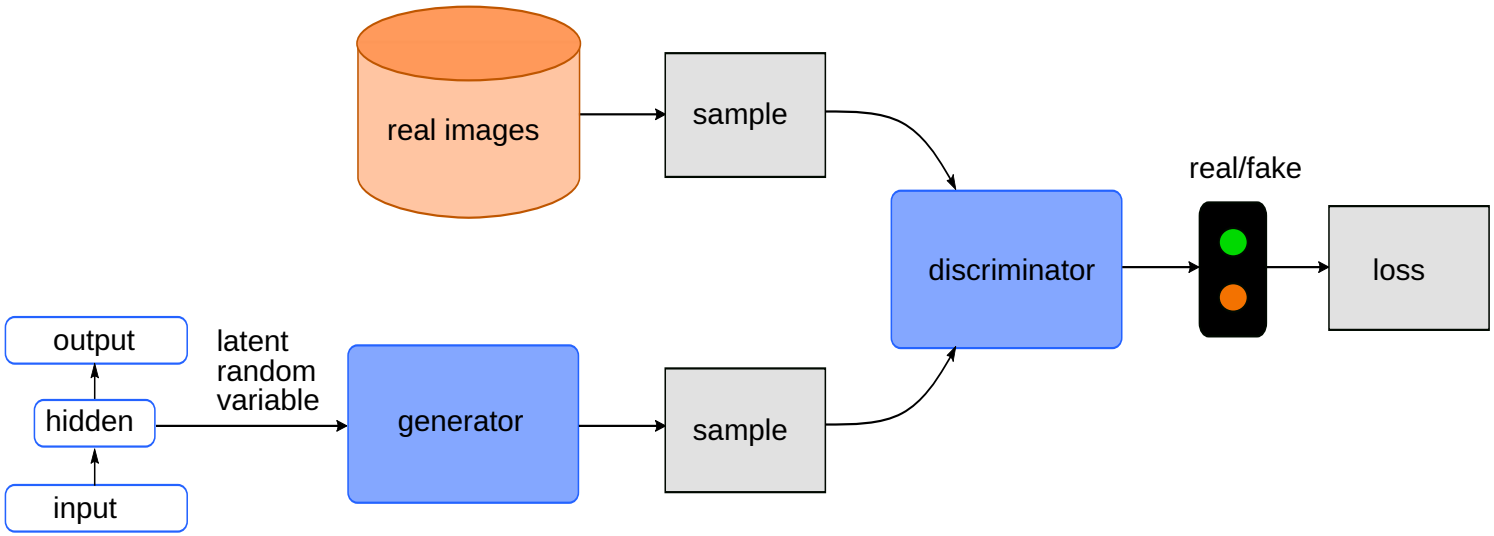
Generative Adversarial Network (GAN)

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Generative Adversarial Networks are also generative models (like VAEs). They model a game between two players:

- **Generator** creates samples intended to come from a training distribution
- **Discriminator** attempts to discern the "real" (original) training samples from the "fake" (generated) ones

Discriminator trains as a binary classifier, generator trains to fool the discriminator



Generative Adversarial Network (GAN)

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

training:

let $D(x)$ be the discriminator parameterized by $\theta^{(D)}$
goal: find $\theta^{(D)}$ minimizing $E^{(D)}$ $(\theta^{(D)}, \theta^{(G)})$

Generative Adversarial Network (GAN)

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

training:

let $D(x)$ be the discriminator parameterized by $\theta^{(D)}$
goal: find $\theta^{(D)}$ minimizing $E^{(D)}$ $(\theta^{(D)}, \theta^{(G)})$

let $G(z)$ be the generator parameterized by $\theta^{(G)}$
goal: find $\theta^{(G)}$ minimizing $E^{(G)}$ $(\theta^{(D)}, \theta^{(G)})$

Generative Adversarial Network (GAN)

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

training:

let $D(x)$ be the discriminator parameterized by $\theta^{(D)}$
goal: find $\theta^{(D)}$ minimizing $E^{(D)}\ (\theta^{(D)}, \theta^{(G)})$

let $G(z)$ be the generator parameterized by $\theta^{(G)}$
goal: find $\theta^{(G)}$ minimizing $E^{(G)}\ (\theta^{(D)}, \theta^{(G)})$

A **Nash equilibrium** of this game is $(\theta^{(D)}, \theta^{(G)})$ such that each $\theta^{(i)}$, $i \in \{D, G\}$ yields a local minimum of its corresponding E.

In game theory, the Nash equilibrium is the most common way to define the solution of a non-cooperative game of two (or more) players. In a Nash equilibrium, each player is assumed to know the strategies of the other players, and no one has anything to gain by changing only one's own strategy.

Generative Adversarial Network (GAN)

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

training:

let $D(x)$ be the discriminator parameterized by $\theta^{(D)}$
goal: find $\theta^{(D)}$ minimizing $E^{(D)}$ $(\theta^{(D)}, \theta^{(G)})$

let $G(z)$ be the generator parameterized by $\theta^{(G)}$
goal: find $\theta^{(G)}$ minimizing $E^{(G)}$ $(\theta^{(D)}, \theta^{(G)})$

A **Nash equilibrium** of this game is $(\theta^{(D)}, \theta^{(G)})$ such that each $\theta^{(i)}$, $i \in \{D, G\}$ yields a local minimum of its corresponding E.

Then **each training step** consists of:

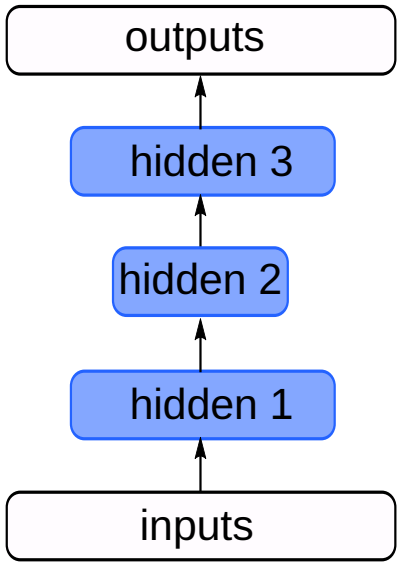
- draw a minibatch of x values from dataset
- draw a minibatch of z values from prior (e.g. $\mathcal{N}(\mathbf{0}, I)$)
- simultaneously update $\theta^{(G)}$ to reduce $E^{(G)}$ and $\theta^{(D)}$ to reduce $E^{(D)}$.

For $J^{(D)}$ common to use cross-entropy where label is 1 for real and 0 for fake data.
Since the generator wants to trick the discriminator, one can use $J^{(G)} = -J^{(D)}$.

Stacked Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Stacked autoencoders have multiple hidden layers by essentially embedding an autoencoder inside of an autoencoder. This allows us to build essentially **deep autoencoders**.



The outer autoencoder has another autoencoder in its hidden layer (blue nodes).

Benefit: share parameters to reduce their number by exploiting symmetry: $W_4 = W_1^T$ and $W_3 = W_2^T$

Stacked Autoencoders

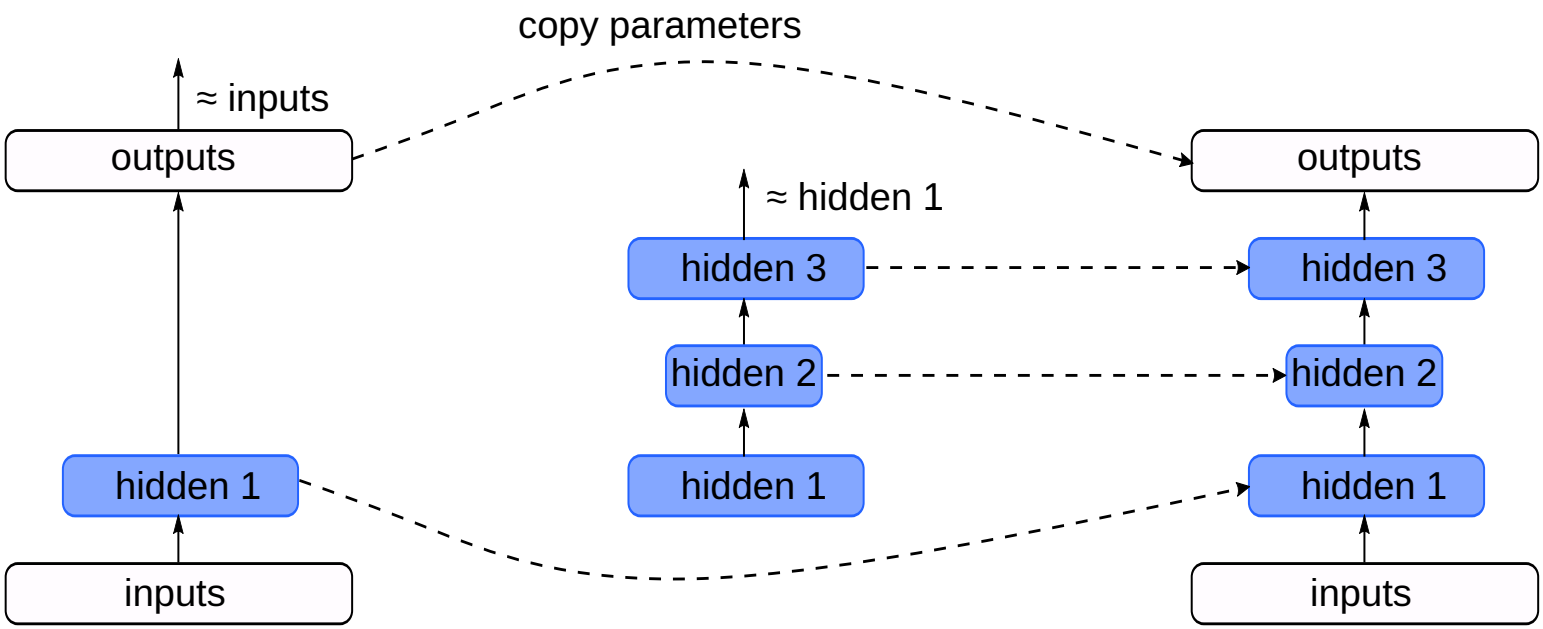
Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Incremental training in stacked autoencoders can simplify training by starting with single hidden layer H_1.

Then train a second autoencoder to mimic the output of H_1.

Insert this into the first network.

Using output from H_1 as training set for Phase 2.



Phase 1:
train the first autoencoder

Phase 2:
train the second autoencoder

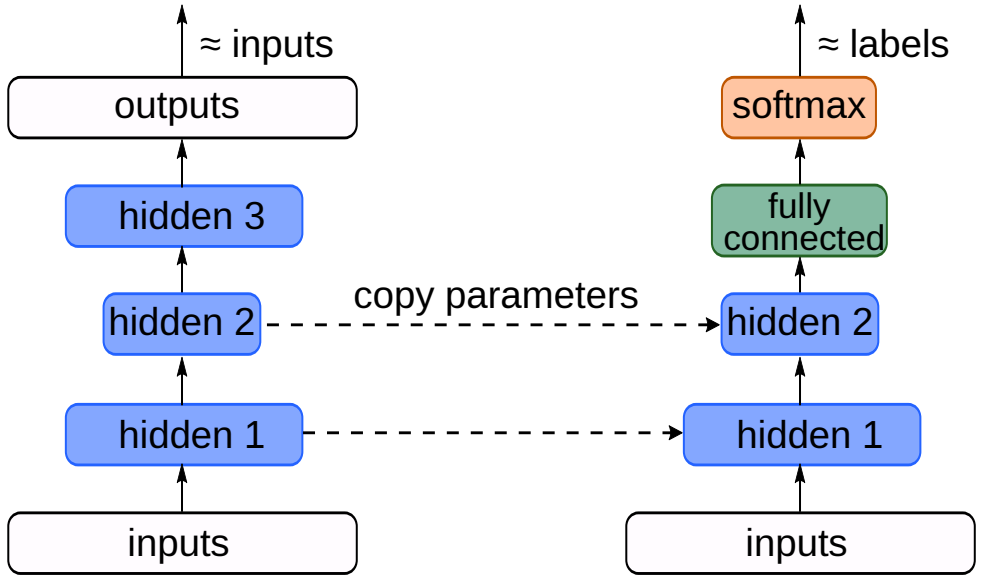
Phase 3:
stack the autoencoders

Stacked Autoencoders

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Semi-supervised training in stacked autoencoders can **pre-train** network with unlabeled data

$\Rightarrow$ learn useful features and then train logic of dense layer with labeled data



Phase 1:
train the autoencoder using all the data

Phase 2:
train the classifier on the labeled data

Applications

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

typical applications of autoencoders include:

Transform noisy data into clean data using **denoising autoencoders**

Add color to grayscale images (image colorization) using **automatic colorization autoencoders**

Increase the resolution of images to improve the details (**super-resolution**)

Compress images to save memory (**image compression**

Reduce the dimensionality of data (**dimensionality reduction**

Extract the most important features of the input data (**feature extraction**)

Generate new data with slight variations using **generative variational autoencoders**

Applications

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

typical applications of autoencoders include:

Transform noisy data into clean data using **denoising autoencoders**

Add color to grayscale images (image colorization) using **automatic colorization autoencoders**

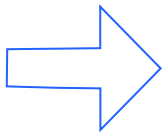
Increase the resolution of images to improve the details (**super-resolution**)

Compress images to save memory (**image compression**

Reduce the dimensionality of data (**dimensionality reduction**

Extract the most important features of the input data (**feature extraction**)

Generate new data with slight variations using **generative variational autoencoders**

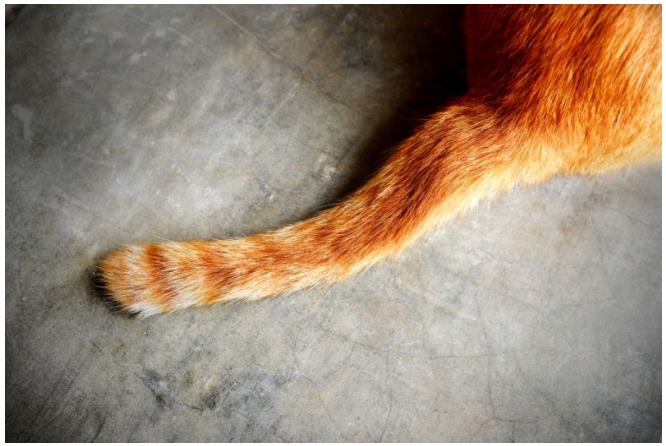


all of this can be used as a part of a larger data-processing pipeline

Applications

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Context Encoders mimic human ability to **guess hidden parts** of an image:



Applications

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

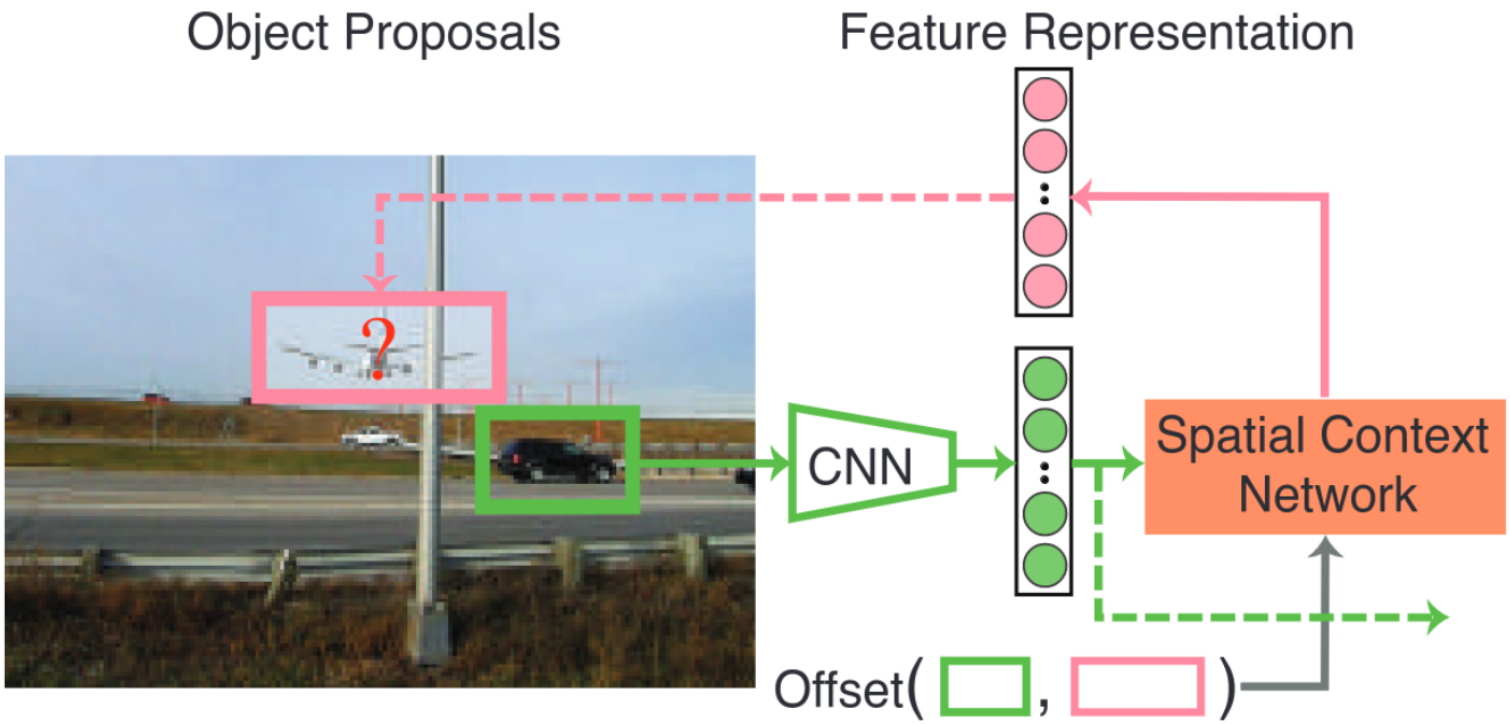


Illustration of the spatial context network by Wu et al (2019). A CNN to compute feature representation of the green patch is fine-tuned to predict feature representation of the red patch using the spatial context module. Pairs of patches used to train the network are obtained from object proposal mechanisms. Once the network is trained, the green CNN can be used as a generic feature extractor for other tasks (dotted green line).

Summary

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

An autoencoder consists of the following building blocks:

Encoder: This is usually a non-linear function that transforms the input, x, into a lower-dimensional latent vector that is denoted by $z = f(x)$.

Latent vector: This is also referred to as latent representation or latent code. It represents the most important features of the input data in a lower-dimensional form and is the output provided by the encoder that will be the input for the decoder. The latent vector dimension can be bigger than x (overcomplete autoencoder), but in most cases, the dimension of the latent vector is much smaller than x (undercomplete autoencoder).

Decoder: This is a non-linear function that takes the latent vector, z, as the input and outputs a compressed representation (recovered input) of the input data. This process, mathematically denoted by $g(z) = \tilde{x}$, is known as decoding. The decoder can only approximate the input because the latent vector is low-dimensional, thus $\tilde{x} \approx x$.

Loss function: A loss function measures the dissimilarity between input x and output $\tilde{x}$. It must be differentiable (for backpropagation). During training, our goal is to make the recovered input, $\tilde{x}$, as close as possible to the original input, x, which we achieve by minimizing the loss function.

Outlook

Recap
Motivation
Autoencoders
Training
Autoencoder Architectures
Applications

Reinforcement learning is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones

⇨ learning from mistakes