# Deep Learning
# Autoencoder

Charlotte Pelletier
*Univ. Bretagne Sud – IRISA Vannes*

December 14 2021

# Content

**The scenario**:
We have acces to the data $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_m\}$; but not the labels $\{y_1, \cdots, y_m\}$.

**The scenario**:
We have acces to the data $\mathcal{D} = \{\mathbf{x}_1, \cdots, \mathbf{x}_m\}$; but not the labels $\{y_1, \cdots, y_m\}$.

**Why tackle such a task?**

1. Extracting interesting information from data
    - clustering: group similar observations together (*e.g.*, $k$-Means)
    - dimensionality reduction, for example for visualization
    - discovering interesting trend
    - data compression (for example MP4, JPEG)
2. Learning better representations for another supervised task
3. Detecting anomalies by learning a likelihood function
4. Generating new samples similar to past observations

# Unsupervised learning

For complex data (text, image, sound, ...), there are plenty of hidden latent structures we hope to capture:

- **image data**: find low dimensional semantic representations, independent sources of variation;
- **text data**: find fixed size, dense semantic representation of data.

For example, **latent space** might be used to help build more efficient human labelling interfaces. $\Rightarrow$ The goal in this case it to reduce labelling cost *via* active learning.

# Unsupervised learning

For complex data (text, image, sound, ...), there are plenty of hidden latent structures we hope to capture:

- **image data**: find low dimensional semantic representations, independent sources of variation;
- **text data**: find fixed size, dense semantic representation of data.

For example, **latent space** might be used to help build more efficient human labelling interfaces. $\Rightarrow$ The goal in this case it to reduce labelling cost *via* active learning.

## Unsupervised representation learning

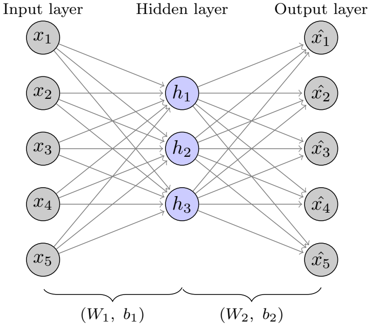
- Force our representations to better model input distribution
  - not just extracting features for classification
  - asking the model to be good at representing the data and not overfitting to a particular task
  - potentially allowing for better generalization
- Use for initialization of supervised task, especially when we have a lot of unlabeled data and much less labeled examples

# Content

# Autoencoder

An autoencoder is a special type of neural network, which is trained to learn the identity function. It is composed of

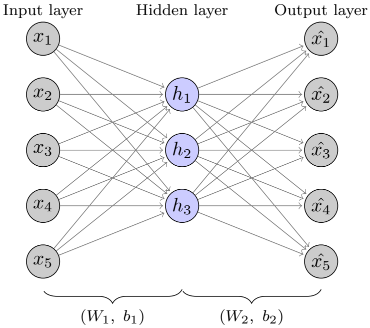
1. an **encoder**, which encodes the input $\mathbf{x}$ into a representation $\mathbf{h}$:
   $\mathbf{h} = g(W_1\mathbf{x} + b_1)$
2. a **decoder**, which decodes the input from the hidden representation $\mathbf{h}$:
   $\hat{\mathbf{x}} = f(W_2\mathbf{h} + b_2)$



## Autoencoder

An autoencoder is a special type of neural network, which is trained to learn the identity function. It is composed of

1. an **encoder**, which encodes the input $\mathbf{x}$ into a representation $\mathbf{h}$:
   $\mathbf{h} = g(W_1\mathbf{x} + b_1)$
2. a **decoder**, which decodes the input from the hidden representation $\mathbf{h}$:
   $\hat{\mathbf{x}} = f(W_2\mathbf{h} + b_2)$



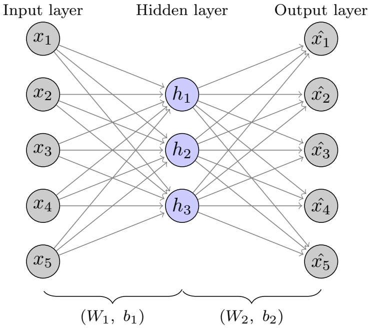

The model is trained to minimize a certain loss function (also called the **reconstruction error**) that will ensure that $\hat{\mathbf{x}}$ is close to $\mathbf{x}$:

$$\ell(\mathbf{x}, \hat{\mathbf{x}}) = ||\hat{\mathbf{x}}^2 - \mathbf{x}||_2^2$$

# Autoencoder

An autoencoder is a special type of neural network, which is trained to learn the identity function. It is composed of

1. an **encoder**, which encodes the input $\mathbf{x}$ into a representation $\mathbf{h}$: $\mathbf{h} = g(W_1\mathbf{x} + b_1)$
2. a **decoder**, which decodes the input from the hidden representation $\mathbf{h}$: $\hat{\mathbf{x}} = f(W_2\mathbf{h} + b_2)$



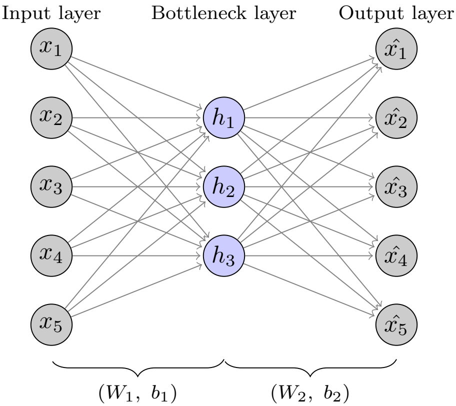

The model is trained to minimize a certain loss function (also called the **reconstruction error**) that will ensure that $\hat{\mathbf{x}}$ is close to $\mathbf{x}$:

$$\ell(\mathbf{x}, \hat{\mathbf{x}}) = ||\hat{\mathbf{x}}^2 - \mathbf{x}||_2^2$$

**Note**: if $f, g$ are linear activation functions and the loss is the squarred loss, this is equivalent to the Principal Component Analysis (PCA).

# Autoencoder

The hidden layer is also called a **bottleneck** layer. It forces a **compressed** knowledge of the original input.

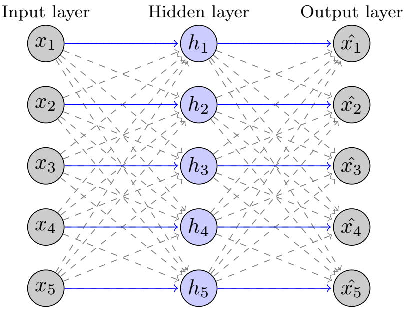


In practice,

- **undercomplete autoencoders**: if $dim(\mathbf{h}) < dim(\mathbf{x})$ and we are able to reconstruct $\mathbf{x}$ perfectly from $\mathbf{h}$, it means that $\mathbf{h}$ is a loss-free encoding of $\mathbf{x}$. It captures all the important charateristics of $\mathbf{x}$.

In practice,

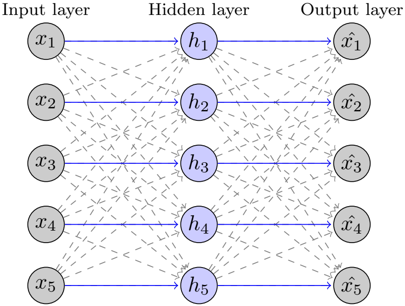
- **undercomplete autoencoders**: if $dim(\mathbf{h}) < dim(\mathbf{x})$ and we are able to reconstruct $\mathbf{x}$ perfectly from $\mathbf{h}$, it means that $\mathbf{h}$ is a loss-free encoding of $\mathbf{x}$. It captures all the important charateristics of $\mathbf{x}$.
- **complete autoencoders**: if $dim(\mathbf{h}) \geq dim(\mathbf{x})$, then the autoencoder could learn a trivial encoding by simply copying $\mathbf{x}$ into $\mathbf{h}$ and copying $\mathbf{h}$ into $\hat{\mathbf{x}}$
$\Rightarrow$ This identity encoding is useless as it does not really tell us anything about the important characteristics of the data.



## Autoencoder

In practice,

- **undercomplete autoencoders**: if $dim(\mathbf{h}) < dim(\mathbf{x})$ and we are able to reconstruct $\mathbf{x}$ perfectly from $\mathbf{h}$, it means that $\mathbf{h}$ is a loss-free encoding of $\mathbf{x}$. It captures all the important charateristics of $\mathbf{x}$.
- **complete autoencoders**: if $dim(\mathbf{h}) \geq dim(\mathbf{x})$, then the autoencoder could learn a trivial encoding by simply copying $\mathbf{x}$ into $\mathbf{h}$ and copying $\mathbf{h}$ into $\hat{\mathbf{x}}$
  $\Rightarrow$ This identity encoding is useless as it does not really tell us anything about the important characteristics of the data.
  We will see that it can be regularized to enforce a **sparsity** constraint.



Some applications:

- Dimensionality reduction (when using an undercomplete autoencoders)
- Feature extraction by using the encoding learned by the encoder.
- Denoising

- Compression
- Image colorization

# Applications

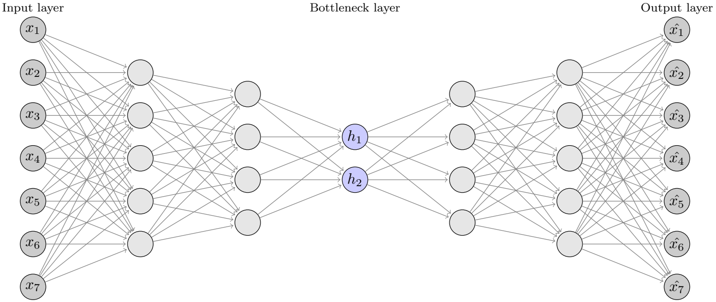
Some applications:

- Watermak removal


- Image generation

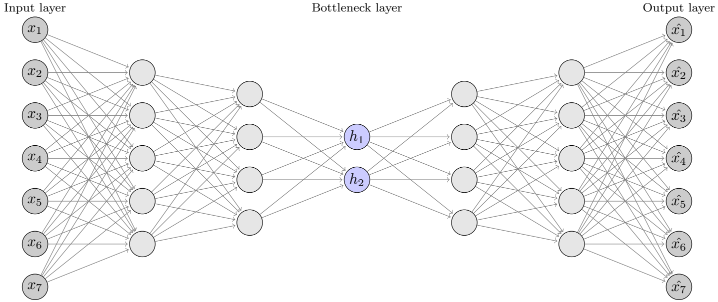
# Deep Autoencoder

- Deep nonlinear autoencoder learn to project the data, not onto a subspace, but onto a nonlinear **manifold**
- This manifold is the image of the decoder.
- This is equivalent to a non-linear dimensionality reduction.



# Deep Autoencoder

- Deep nonlinear autoencoder learn to project the data, not onto a subspace, but onto a nonlinear **manifold**
- This manifold is the image of the decoder.
- This is equivalent to a non-linear dimensionality reduction.




**Note**: The term *deep autoencoder* is sometimes used interchangeably with *stacked autoencoder*, which is another type of architecture, where we train a first autoencoder, minimised the loss function, then feed the hidden layer as the input of a next autoencoder, and so on. It usually helps the training at the cost of an increase in the training time.

# Regularized Autoencoder

The “ideal” autoencoder model balances the following:

1. sensitive to the inputs enough to accurately build a reconstruction,
2. insensitive enough to the inputs that the model does not simply memorize or overfit the training data.

# Regularized Autoencoder

The "ideal" autoencoder model balances the following:

1. sensitive to the inputs enough to accurately build a reconstruction,
2. insensitive enough to the inputs that the model does not simply memorize or overfit the training data.

This trade-off requires the model to maintain only the variations in the data required to reconstruct the input without holding on to redundancies.

# Regularized Autoencoder

The "ideal" autoencoder model balances the following:

1. sensitive to the inputs enough to accurately build a reconstruction,
2. insensitive enough to the inputs that the model does not simply memorize or overfit the training data.

This trade-off requires the model to maintain only the variations in the data required to reconstruct the input without holding on to redundancies.

This is done by having a loss function with two terms:

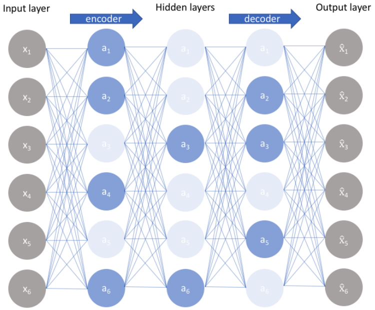
1. the first term, the **reconstruciton loss** $\ell(\mathbf{x}_i, \hat{\mathbf{x}}_i)$, encourages the model to be sensitive to the inputs,
2. the second term, the **regularized term** $\mathcal{R}$, discourages memorization and overfitting by adding a penalization.

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i) + \lambda \mathcal{R},$$

where $\lambda$ is the regularizer trade-off hyperparameter.

## Sparse Autoencoder

- It uses a complete autoencoder where there is no reduction in the number of nodes in the the bottleneck layer.
- It adds a regularization term to penalize activations. The network is encouraged to learn an encoding and decoding which relies on only a small number of activations.
- It is forced to selectively activate regions of the network depending on the input data.




$\Rightarrow$ The network's capacity to memorize the input data is limited while the network's capability to extract features from the data is not restrained.

**Source**: `https://www.jeremyjordan.me/autoencoders/`

How to impose this sparsity constraint?

1. **L1 regularization**: it penalizes the absolute values of the vector of activations $a$ in a layer $l$:

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m} \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i) + \lambda \sum_i |a_i^{(l)}|$$

# Sparse Autoencoder

How to impose this sparsity constraint?

1. **L1 regularization**: it penalizes the absolute values of the vector of activations $a$ in a layer $l$:

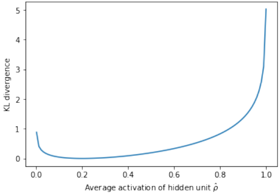
$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m} \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i) + \lambda \sum_i |a_i^{(l)}|$$

2. **Kullback-Leibler (KL) divergence**: it is a measure of the difference between two probability distributions.

We constrain the average activation of a neuron over a collection of training data.

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m} \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i) + \lambda KL(\rho || \hat{\rho}_j),$$

where $\hat{\rho}_j$ is the average activation of a neuron over a collection of samples $\hat{\rho}_j = \frac{1}{m}\sum_i a_i^{(l)}(x)$ for a specific neuron $j$ in layer $l$ (observed distribution); and $\rho$ is the sparsity parameter (the ideal distribution).



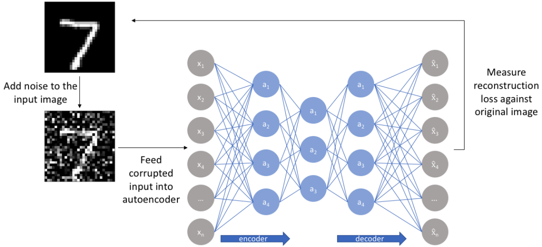

**Sour**

`https://www.jeremyjordan.me/autoencoders/` $-\rho = 2$

# Denoising Autoencoder

In a denoising autoencoder, the input is different from the output:

- **corrupted** input = image + noise ⇒ improves model's generalization
- output: denoised image




**Source**: `https://www.jeremyjordan.me/autoencoders/`

# Contractive Autoencoder

- Similar inputs should have a similar encoding.
- We want thus to maintain a similar encoded state for small changes of the input ⇒ having small derivatives of the hidden layer activations with respect to the input:

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m} \ell(\mathbf{x}_i, \hat{\mathbf{x}}_i) + \lambda \sum_{i,j} ||\nabla_{x_i} a_j^{(l)}(x_i)||_F^2,$$

where $\nabla_{x_i} a_j^{(l)}(x_i)$ defines the gradient field of the hidden layer activations (in layer $l$) with respect to an input $x_i$.