# ELEC96033: Deep Learning

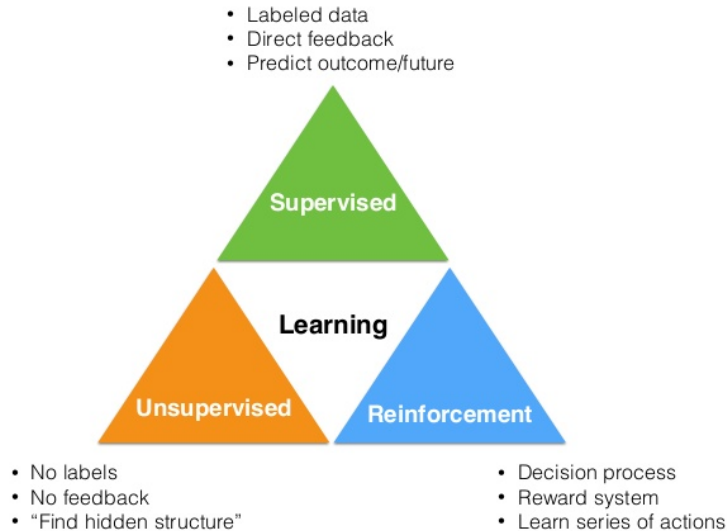
Carlo Ciliberto and Krystian Mikolajczyk

Department of Electrical and Electronic Engineering
Imperial College London

# Deep Learning - Part 2

- Representation Learning
- Generative Models
- Reinforcement Learning
- Hyperparameter Optimization and Meta-Learning

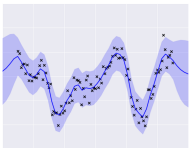
# A Taxonomy of Machine Learning



## Supervised Learning

**Goal:** learn a relation between inputs and outputs $f : \mathcal{X} \to \mathcal{Y}$.

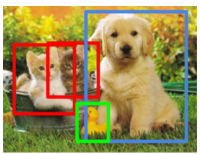
given a a number of examples $(x_i, y_i)_{i=1}^n$



**Regression**

*"cat"*

**Classification**



**Localization**

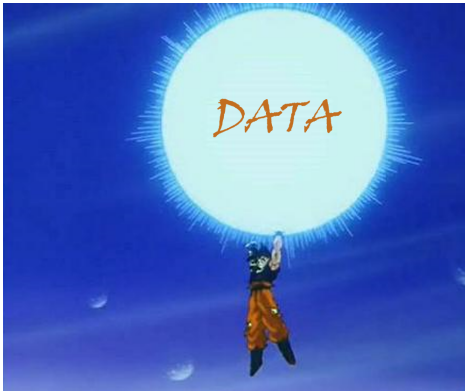
**Segmentation**

*"a group of people playing frisbee"*

**Captioning**

**Note:** The larger $n$ (= more training examples), the better...

Typical Approach: "If it does not work, just trow more data at it"




**But** what if we don't have enough (labeled) data?

## Unsupervised Learning to the Rescue?

**Idea:** automatically find relevant patterns in (input) data.

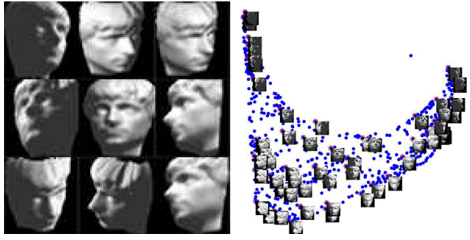
Two main purposes:

- **Simplification:** reduce the (sample) complexity of a supervised learning problem by identifying only useful features in the data and discarding distracting elements.
- **Interpretability:** indentify common qualities (features) between individual elements or groups from huge amounts of data leading to a better understanding of the problem.

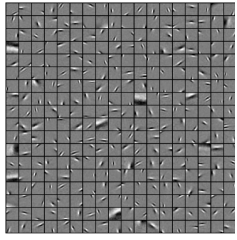
**Benefit:** usually inexpensive to collect large training sets (no need for labeling).

# Semi-supervised Assumption

**Assumption:** Input data needs to encode useful properties about input-output relation.



Pictures of human faces span a very low-dimensional subspace with respect to all possible images.



Natural images are well described by (local) gabor-like patches.

## Unsupervised Learning: A Taxonomy

- Clustering
- Dimensionality Reduction
- Density Estimation
- Feature/Representation Learning

# Unsupervised Learning: A Taxonomy

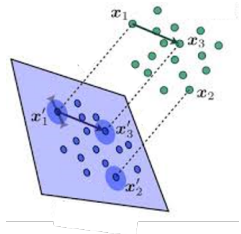
- **Clustering**
- Dimensionality Reduction
- Density Estimation
- Feature/Representation Learning

**Goal:** group points according to their "common properties"...

# Unsupervised Learning: A Taxonomy

- Clustering
- **Dimensionality Reduction**
- Density Estimation
- Feature/Representation Learning

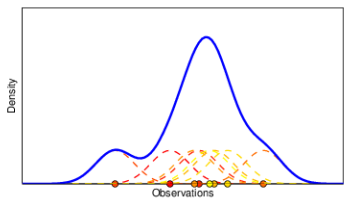
**Goal:** map points to a low dimensional space by keeping thir relations (almost) unchanged.



# Unsupervised Learning: A Taxonomy

- Clustering
- Dimensionality Reduction
- **Density Estimation**
- Feature/Representation Learning

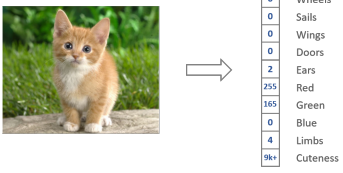
**Goal:** (i) approximate the distribution from which the observed data has been sampled.
(ii) Possibly "generate" new examples (more in next class).



# Unsupervised Learning: A Taxonomy

- Clustering
- Dimensionality Reduction
- Density Estimation
- **Feature/Representation Learning**

**Goal:** Encode data within a concise but characteristic representation that can be used to solve other tasks (e.g. classification, regression, etc.)



## Prelude: Sample Complexity, Hypotheses Spaces and All that...

Let's go back to supervised learning for a second...

**Goal:** find the best function $f : \mathcal{X} \to \mathcal{Y}$ minimizing the **expected risk**

$$R(f) = \mathbb{E}\ \ell(f(x), y) = \int_{X \times Y} \ell(f(x), y)\ d\rho(x, y)$$

given only a finite number of (training) examples $S = \{(x_1, y_1) \dots (x_n, y_n)\}$ sampled from $\rho$.

- $\rho$ is an unknown data distribution.
- $\ell$ is a loss function measuring prediction errors.

**Typical Approach?** Empirical Risk Minimization (ERM).

## Empirical Risk Minimization

**Idea:** parametrize the candidate solutions as $f(x) = w^\top \phi(x)$

- $w \in \mathcal{H}_\phi$ is a vector of parameters in a Feature Space $\mathcal{H}_\phi$.
- $\phi : \mathcal{X} \to \mathcal{H}_\phi$ is a Feature Representation.

Then, minimize the Empirical Risk

$$\hat{w} = \operatorname*{argmin}_{w \in \mathcal{H}_\phi} \frac{1}{n} \sum_{i=1}^{n} \ell(w^\top \phi(x_i), y_i)$$

## Sample Complexity

One way to measure the quality of ERM (with representation $\phi$) is the **sample complexity**.

Here we consider a small variant:

**Definiton.** Given $\delta \in [0,1]$ and $\varepsilon > 0$, the **sample complexity** of ERM with representation $\phi$ on the learning problem with data distribution $\rho$, is defined as the minimum number $n_{\delta,\varepsilon}(\rho, \phi)$ of training points such that

$$\mathbb{P}\left( R(\hat{w}) - \inf_{f:\mathcal{X}\to\mathcal{Y}} R(f) < \varepsilon \right) \geqslant 1 - \delta$$

**In other words:** $n_{\delta,\varepsilon}(\rho, \phi)$ is the minimum number of points needed to guarantee the performance of ERM to be $\varepsilon$-close to the best possible one (with confidence $1 - \delta$).

## Sample Complexity and Data Representation

Ideally, the smaller the sample complexity $n(\rho, \phi)$, the better. However...

**Theorem (No Free Lunch).** For any representation $\phi$, we have $\sup_\rho n(\rho, \phi) = +\infty$!

**TL;DR.** There is not a single feature representation that can solve all problems.

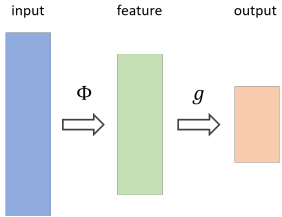
**Question.** What if, instead of keeping $\phi$ fixed, we adapted it to the unknown distribution $\rho$?
...maybe using only (inexpensive) unlabeled training input points?

Data Representation Learning: for each $\rho$ we try to learn $\phi_\rho$

# Autoencoders

# Representation Learning

We want to learn $f(x) = g(\phi(x))$.



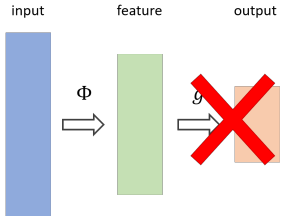

- $\phi$ feature representation,
- $g$ label predictor.

**End-to-end learning:** learn $\phi$ and $g$ jointly.

- Standard approach in supervised learning.
- Works well if we have lots of labeled data.

## Representation Learning

We want to learn $f(x) = g(\phi(x))$.




- $\phi$ feature representation,
- $g$ label predictor.

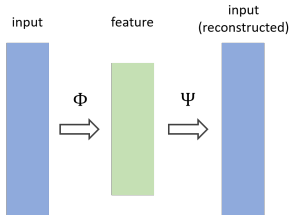
**Note:** $g$ needs output labels to be learned (which are expensive). But what about $\phi$?

*Can we learn $\phi$* **without** *using output labels?*

*(but only input examples?)*

# Representation Learning: AutoEncoders

**Idea:** use the reconstruction error on input data as a form of "self"-supervision.




Reconstruction error: $\|x - \psi(\phi(x))\|^2$.

- $\phi$ feature representation/**encoding**,
- $\psi$ **decoding**.

**Benefits:**

- Unlabeled input data is (often) unexpensive: we can get many self-labelled examples!
- Learned $\phi$ can capture more general properties of the data: can be used for different tasks!

## Training Autoencoders

In practice: given a dataset of $n$ points $(x_i)_{i=1}^n$, we want to solve

$$\min_{\phi,\psi} \frac{1}{n} \sum_{i=1}^{n} \|x_i - \psi(\phi(x_i))\|^2$$

**Problem.** we have a trivial solution: $\psi \circ \phi =$ identity... why?

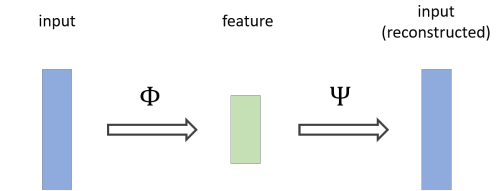
Because, the unconstrained $\phi$ and $\psi$ can "learn" all possible variability in training data (in some sense overfitting it).

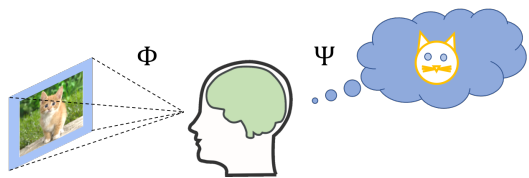
Ideally, we would like to capture only the key aspects in data but ignore noise/nuances. This idea is in line with the Efficient Coding Hypothesis from the neuroscience of vision...

# Efficient Coding Hypothesis & Parsimonious Representations

**Efficient Coding Hypothesis** (Barlow '61))

"*... the Efficient Coding Hypothesis holds that the purpose of early visual processing is to produce an efficient representation of the incoming visual signal.*" (Simoncelli '03)



## "Linear" Autoencoders

In general we will consider Autoencoders where $\phi$ and $\psi$ are deep nets.

**(Simplification)** However, to understand what is going on, let's try to first have a look a what shallow, linear Autoencoders (i.e. one hidden layer, no activation function) do...

$\phi : \mathbb{R}^d \to \mathbb{R}^r$ and $\psi : \mathbb{R}^r \to \mathbb{R}^d$ correspond to matrices $F, P \in \mathbb{R}^{r \times d}$, such that

$$\psi(\phi(x)) = P^\top F x$$

Note that the matrix $P^\top F$ is $d \times d$ of rank at most $r \leqslant d$.
(and if $r < d$, then $P^\top F \neq I$ cannot be the identity)

## "Linear" Autoencoders (Cont.)

The problem of training a linear Autoencoder becomes,

$$\min_{\psi,\phi} \frac{1}{n}\sum_{i=1}^{n} \|x_i - \psi(\phi(x_i))\|^2 = \min_{P,F\in\mathbb{R}^{d\times r}} \frac{1}{n}\sum_{i=1}^{n} \left\|x_i - P^\top F x_i\right\|^2 = \min_{P,F\in\mathbb{R}^{d\times r}} \frac{1}{n}\left\|X - P^\top F X\right\|_F^2$$

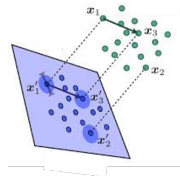
- $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$ the matrix concatenating the input data
- $\|M\|_F^2 = \mathrm{Tr}(MM^\top)$ squared Frobenius norm of a matrix $M$ (sum of squared entries).

**Theorem:**

The solution $\hat{P}^\top \hat{F}$ encodes the first $r$ principal components of the data: we are doing PCA!

## Linear Autoencoders & Principal Component Analysis

**Step 1:** $P^\top F$ is learning an orthogonal projector onto a linear subspace of dimension $r$.




To see this:

1. Set the derivative of objective functional with respect to $F$ equal to 0:

$$0 = \nabla_F \left\| X - P^\top F X \right\|^2 = 2P(X - P^\top F X)X^\top = 2(P - PP^\top F)XX^\top$$

2. Then $P - PP^\top F = 0$ is solved by $F = (P^\top P)^\dagger P = (P^\top)^\dagger$ the pseudoinverse of $P^\top$.

Hence: $P^\top F = P^\top (P^\top)^\dagger = \Pi$ is an orthogonal projector (as desired)!

## Linear Autoencoders & Principal Component Analysis

**Step 2:** $\Pi = P^\top F$ is the projector associated to the first $r$ principal components of $X$.

1. Note that for a projector, $\Pi^\top \Pi = \Pi$. Then, we have $(I - \Pi)^\top (I - \Pi) = I - \Pi$.
2. Therefore
$$\left\| X - P^\top F X \right\|^2 = \|(I - \Pi)X\|^2 = \mathrm{Tr}(X^\top (I - \Pi)^\top (I - \Pi) X) = \mathrm{Tr}(X^\top (I - \Pi) X).$$
3. Recall that an orthogonal projector of rank $r$ can be written as $\Pi = UU^\top$, with $U \in \mathbb{R}^{d \times r}$ a matrix with orthonormal columns, namely $U^\top U = I \in \mathbb{R}^{r \times r}$.
4. Hence: $\mathrm{Tr}(X^\top (I - \Pi) X) = \mathrm{Tr}(X^\top X) - \mathrm{Tr}(X^\top U U^\top X) = \mathrm{Tr}(X^\top X) - \mathrm{Tr}(U^\top X X^\top U)$.

## Linear Autoencoders & Principal Component Analysis

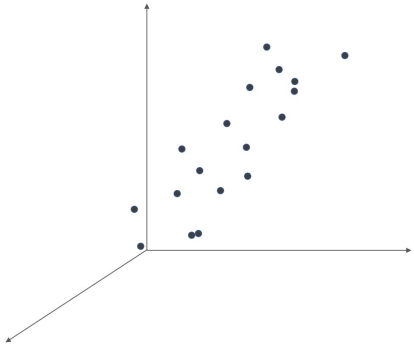
Therefore, the problem of training an autoencoder can be re-formualated as

$$\min_{F,P\in\mathbb{R}^{r\times d}} \left\|X - P^\top F\right\|^2 = \min_{\substack{U\in\mathbb{R}^{d\times r}\\ U^\top U=I}} \mathrm{Tr}(X^\top X) - \mathrm{Tr}(U^\top XX^\top U) = \max_{\substack{U\in\mathbb{R}^{d\times r}\\ U^\top U=I}} \mathrm{Tr}(U^\top XX^\top U)$$

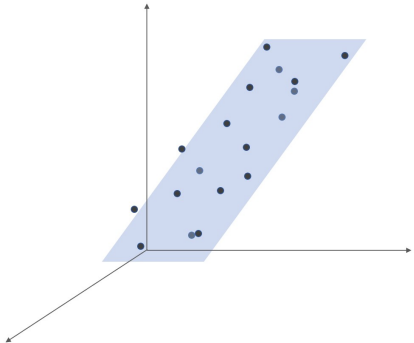
- $\mathrm{Tr}(U^\top XX^\top U) = \sum_{j=1}^{r} u_j^\top XX^\top u_j$, with $u_j$ the $j$-th column of $U$.
- Recall that $\max_{\|u\|=1} u^\top XX^\top u$ is solved by the first principal component of $X$.

Hence, $\min_{F,P} \left\|X - P^\top F\right\|^2 = \max_{U^\top U=I} \sum_{j=1}^{r} u_j^\top XX^\top u_j$ = find the first $r$ principal components of $X$!

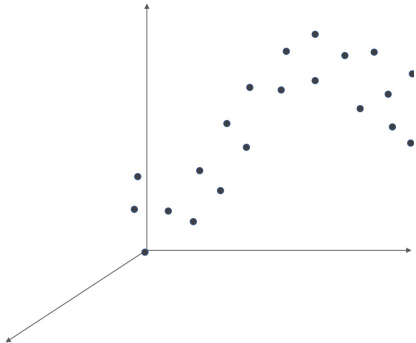
# Geometric Interpretation of Linear Autoencoders/PCA



# Geometric Interpretation of Linear Autoencoders/PCA

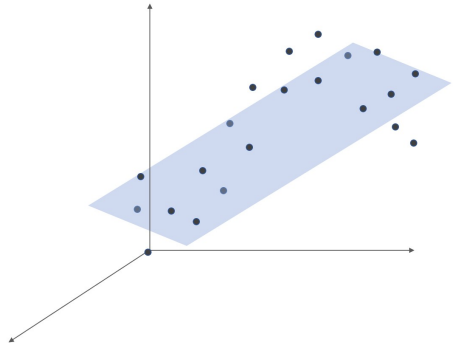


# Geometric Interpretation of Linear Autoencoders/PCA



**Question:** what if the data do not lie on a plane?

# Geometric Interpretation of Linear Autoencoders/PCA



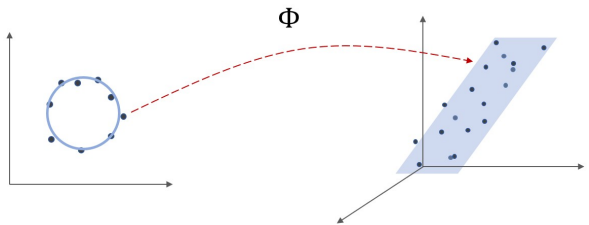
**Question:** what if the data do not lie on a plane?

## "Nonlinear" Features

Rather than working directly with $x_1, \ldots, x_n$, we can consider a feature transformation $\phi$ that captures the non-linear relations in the data...

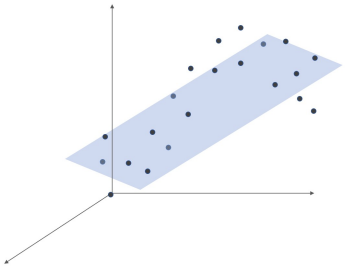
...if we know such nonlinear relations beforehand (e.g. quadratic, cubic, etc.) .

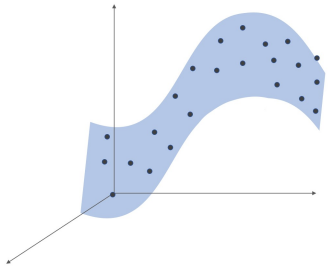
Then we perform linear PCA on the "encoded" dataset $\phi(x_1), \ldots, \phi(x_n)$.



# Nonlinear PCA

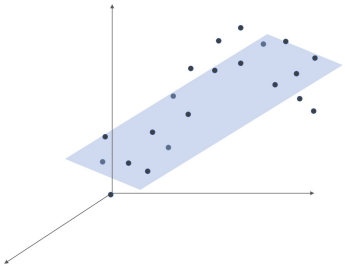
So, if we know $\phi$, we can learn the correct (nonlinear) principal directions of the data.

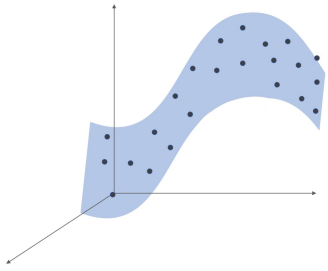




## Nonlinear PCA

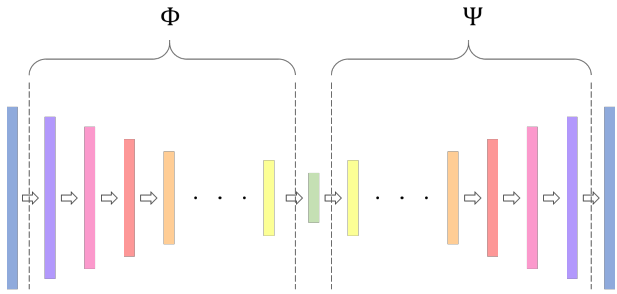
So, if we know $\phi$, we can learn the correct (nonlinear) principal directions in the data.





What if we don't know $\phi$? **We learn it of course!** (with a deep, nonlinear Autoencoder!)

# Deep Nonlinear Autoencoders




**Intuition:** We are learning jointly:

- The key (possibly nonlinear) features/relations in the data,
- The principal "directions" in the resulting feature space.

# Autoencoders: Models

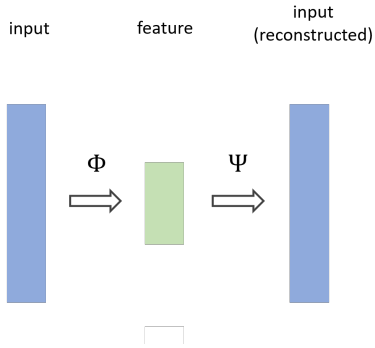
## Autoencoders

Many different options for Autoencoders:

- Undercomplete
- Overcomplete + Regularized
- Denoising
- Contractive
- Convolutional
- ...

## Undercomplete Autoencoders

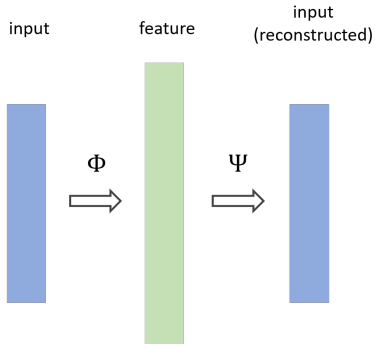
Feature representation layer dimension significantly smaller than input layer.




- We obtain a concise description of the input features (a sort of "semantic" compression).
- Also useful for: dimensionality reduction, data visualization, clustering, ...

## Overcomplete Autoencoders

Feature representation layer dimension significantly larger than input layer.




- Allows for a rich representation (e.g. neurons learn a "dictionary" of patterns)...
- ...but needs regularization!

## Regularized Autoencoders

Add a penalty $\Omega(\psi, \phi)$ (often just $\Omega(\phi)$) to the learning problem

$$\min_{\psi,\phi} \frac{1}{n} \sum_{i=1}^{n} \|x_i - \psi(\phi(x_i))\|^2 + \Omega(\psi, \phi)$$

**Intuition:** $\Omega$ to encourages the weights of the networks $\phi$ and $\psi$ to be:

- small,
- sparse,
- symmetric,
- ...

## Weight Decay

Standard regularization approach: $\ell_2$ regularization. Let $\phi$ and $\psi$ be neural networks

parmetrized by weights $W = \{W^{(1)}, \ldots, W^{(L)}\}$. Then, $\Omega(\phi, \psi) = \Omega(W)$ with

$$\Omega(W) = \lambda \sum_{l=1}^{L} \left\| W^{(l)} \right\|^2 = \lambda \sum_{l=1}^{L} \sum_{i=0}^{d^{(l-1)}} \sum_{j=1}^{d^{(l)}} (w_{ij}^{(l)})^2$$

- Weight decay favours Autoencoders with small weights,
- We can choose a different $\lambda_l$ for each layer (e.g. depending on the layer dimension)

# Sparse Autoencoders

**Idea:** Learn an overcomplete representation of neurons that are tuned only to specific patterns in the data $\longrightarrow$ we would like to observe sparse activation patterns!

**Connection** with the behavior of simple cells in the mammalian primary visual cortex.

## Sparse Autoencoders (Cont.)

**Idea:** Model neurons as a Bernoulli distributions that "fire" with probability $\rho \in [0, 1]$.

- Evaluate the empirical average activation of a neuron in the representation layer.

$$\hat{\rho}_j = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i)^{(j)}$$

  for any $j = 1, \ldots, r$ the number of neurons in the representation layer.

- Penalize with respect to the Kullback-Leibler divergence between $\rho$ and $\hat{\rho}_j$

$$\mathrm{KL}(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + \rho \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

## Sparse Autoencoders (Cont.)

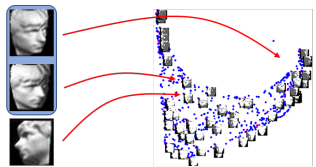
The regularizer becomes

$$\Omega(\phi, \psi) = \sum_{j=1}^{r} \mathsf{KL}(\rho \| \hat{\rho}_j)$$

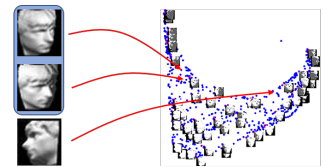
**Remark:** given the interaction between neural activation and the regularizer $\Omega$, the choice of the activation function (e.g. ReLU, sigmoid, tanh, ...) is critcal.

# Contractive Autoencoders

**Idea:** We would like the feature map $\phi$ to be "stable" to small perturbations in the input.



Unstable
(similar input -> different representation)



Stable
(similar input -> similar representation)

**Wishlist.** We would like a representation that is:

- Locally invariant to small changes around training points,
- Able to dicover the low-dimensional manifold structure in the data (if any).

## Contractive Autoencoders (Cont.)

Local variations of the feature representation around a point $x$ are encoded by the derivatives of $\phi$: the larger the derivative, the less stable the feature map.

**Penalty:** we can choose the regularizer

$$\Omega(\phi) = \sum_{i=1}^{n} \|J_\phi(x_i)\|_F^2 \qquad \text{with} \qquad J_\phi(x) = \begin{bmatrix} \frac{\partial \phi_1}{\partial x_1}(x) & \cdots & \frac{\partial \phi_1}{\partial x_d} \\ \vdots & & \vdots \\ \frac{\partial \phi_r}{\partial x_1}(x) & \cdots & \frac{\partial \phi_r}{\partial x_d} \end{bmatrix}$$

- $J_\phi(x)$ is the Jacobian of $\phi$ in $x$,
- $\|\cdot\|_F^2$ the squared Frobenius norm ($=$ sum of the squared entries of a matrix).

## Contractive Autoencoders (Cont.)

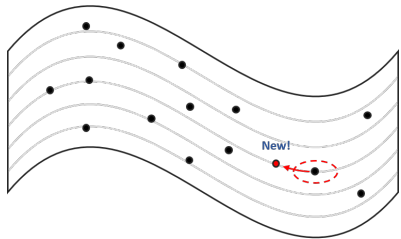
A map $\phi$ whose Jacobian has all singular values strictly less than 1 is called a *contraction*.

An Autoencoder trained using the penalty $\Omega(\phi) = \sum_{i=1}^{n} \|J_\phi(x_i)\|_F^2$ is called a **Contractive Autoencoder** since $\Omega$ encourages feature representations that are (local) contractions.

**Practical Note:** activation functions that are twice differentiable are preferable/needed (e.g. sigmoid, tanh... not ReLU).

# A Peek into Next Class: Generating Data with Contractive Autoencoders

- The encoding $\phi(\cdot)$ captures and defines a "data manifold".
- The Jacobian $J_\phi$ describes the directions along which such manifold varies the most. (and, more importantly, "illegal" directions).




**Question:** can we move on this manifold to generate new "realistic" points?

## A Peek into Next Class: Generating Data with Autoencoders (Cont.)

Let's generate $\varepsilon$ a random perturbation (e.g. $\varepsilon \sim \mathcal{N}(0, \sigma I)$) and move along the direction

$$\hat{x}_1 = x_0 + J_0^\top \varepsilon \qquad \text{with} \qquad J_0 = J_\phi(x_0).$$

The procedure che be repeated, generating a sequence $x_0, \ldots, x_T$ of points moving away from the original $x_0$ along the manifold identified by $\phi$.

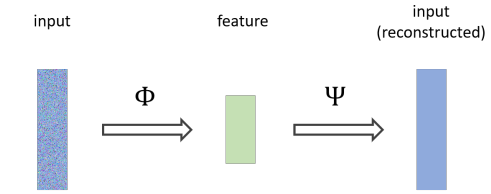
**Note:** actually, at every step we move along the plane tangent to the manifold. In order to "project" back to the manifold we can use the reconstruction map $\psi$:

$$x_{t+1} = \psi\Big( \phi\big(x_t + J_t^\top \varepsilon_t\big) \Big) \approx \psi\Big( \underbrace{\phi(x_t) + J_t\, J_t^\top \varepsilon_t}_{1^{\text{st}} \text{ order Taylor expansion of } \phi} \Big)$$

# Dealing with Noise

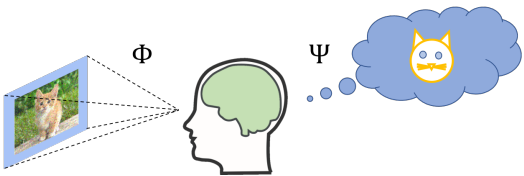
**Setting:** noise can affect data.

An efficient coding should automatically "ignore" it.



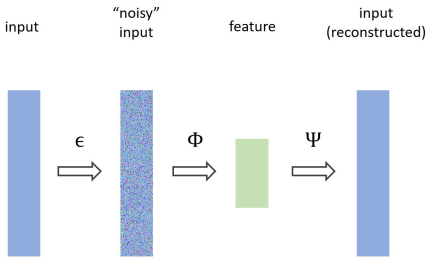

Two main approaches:

- Denoising Autoencoders.
- Robust Autoencoders.



## Dealing with Noise: Denoising Autoencoders

**Approach:** given a data point $x$, "corrupt" it with some noise $\varepsilon$, obtanining for instance $\hat{x} = x + \varepsilon$ (or $\hat{x} = x\varepsilon$). Then learn how to "denoise" $\hat{x}$ and reconstruct $x$.




**Denoising Autoencoders** aim to be robust with respect to non-smooth perturbation of input data (such as **noise**).

# Dealing with Noise: Denoising Autoencoders (Cont.)

Consider the case of additive noise. Training a denoising Autoencoder amounts to solve

$$\min_{\phi,\psi} \frac{1}{n} \sum_{i=1}^{n} \|x_i - \psi(\phi(x_i + \varepsilon_i))\|^2$$

The noise vector $\varepsilon$ can be sampled according to the task. Examples:

- (Additive) Gaussian,
- (Multiplicative) Bernoulli (e.g. suppressing individual pixels in images).
- Any noise, really...

# Denoising Autoencoders: a Data Augumentation Perspective

- For each $x_i$, we can sample many perturbations $\varepsilon_i^{(1)}, \ldots, \varepsilon_i^{(m)}$ "generate" even more training points!

- Noise is meant in a very abstract sense. For instance if we want to be **invariant** to specific input transformations (e.g. rotations, translations, etc.), we can perturb $x_i$ accordingly.





## Dealing with Noise: Robust Autoencoders

A different approach to build robust representations is to replace the quadratic error with a loss function that is more robust to outliers.

$$\min_{\psi,\phi} \frac{1}{n} \sum_{i=1}^{n} \|x_i - \psi(\phi(x_i))\|^2 \qquad \longrightarrow \qquad \min_{\psi,\phi} \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, \psi(\phi(x_i)))$$

Examples:

- Absolute value loss $\ell(x, \psi(\phi(x))) = \sum_{k=1}^{d} |x^{(k)} - \psi(\phi(x))^{(k)}|$.
- "Correntropy" $\ell(x, \psi(\phi(x))) = \sum_{k=1}^{d} \exp(-(x^{(k)} - \psi(\phi(x))^{(k)})^2/\sigma)$. With $\sigma > 0$.
- Huber, Cauchy, German-McLure, Fair, $L2 - L1$...

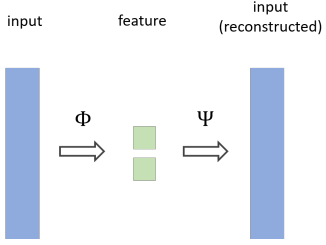
with $x^{(k)}$ denoting the $k$-th entry of $x \in \mathbb{R}^d$.

# Autoencoders: Applications

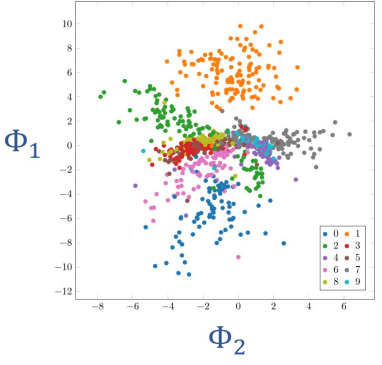
# Applications: Data Visualization

**Idea:** set the feature representation layer to have only 2 or 3 dimensions.

Then we can visualize each data point in this new representation!



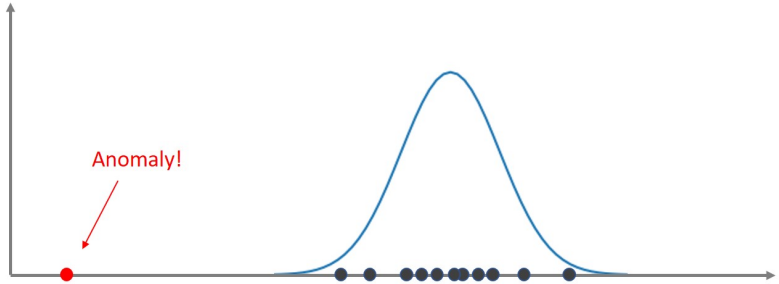

MNIST Digits




**Useful for:** interpretation.

## Applications: Anomaly Detection

**Task:** given a set of points $x_1, \ldots, x_n$ sampled from a distribution $\rho$, we want to determine whether a new point $x$ is sampled from the same distribution or not (= an anomaly).



## Applications: Anomaly Detection

**Idea:** since Autoencoders implicitly learn the manifold on which $\rho$ is "supported"

$\implies$ whenever a point $x$ is not sampled from $\rho$ it will have a high reconstruction error.

**Approach:**

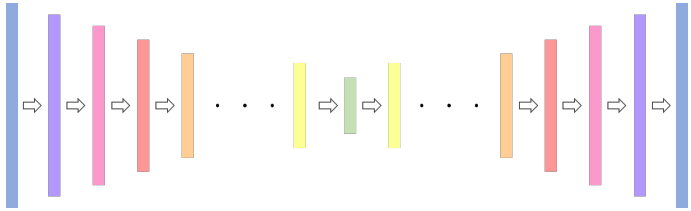
1. Train an Autoencoder on $x_1, \ldots, x_n$.
2. Given a new point $x$, if its reconstruction error

$$\|x - \psi(\phi(x))\| > \tau$$

is greater than a threshold $\tau > 0$, mark it as an anomaly.

## Weight Initialization & Stacking

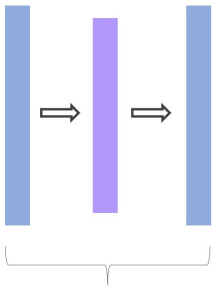
**Practical concern:** Deep Autoencoders are very sensitive to initialization...



**Simple & Effective strategy.** Stacking + Fine-tuning: we initialize each intermediate layer as a shallow Autoencoder using the previous representation layer as input. Then, we fine-tune the whole network.

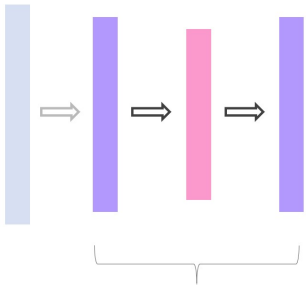
## Weight Initialization & Stacking

**Stacking:** we initialize each intermediate layer as a shallow Autoencoder using the previous representation layer as input.



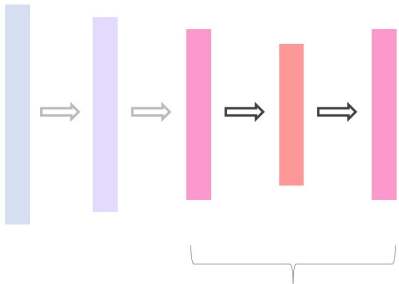
## Weight Initialization & Stacking

**Stacking:** we initialize each intermediate layer as a shallow Autoencoder using the previous representation layer as input.



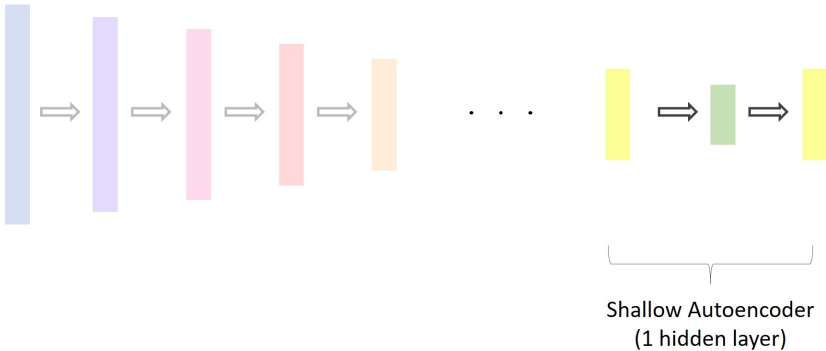
## Weight Initialization & Stacking

**Stacking:** we initialize each intermediate layer as a shallow Autoencoder using the previous representation layer as input.



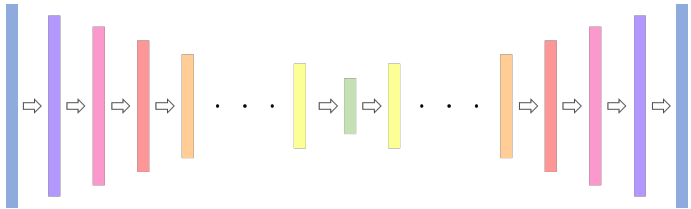
# Weight Initialization & Stacking

**Stacking:** we initialize each intermediate layer as a shallow Autoencoder using the previous representation layer as input.



## Weight Initialization & Stacking

**Fine-tuning:** starting from the weights learned independently from each layer, we fine-tune the whole network.



**Note:** a possible approach to speed up the entire process is to initialize $\phi$ and $\psi$ symmetrically (and possibly add some noise to avoid perfect symmetry).

## Summary: Autoencoders Main Design Choices

**Dimensionality:**

- Undercomplete,
- Overcomplete.

**Noise Tolerance:**

- Denoising Autoencoder,
- Robust Loss functions.

**Network Structure of $\phi$ and $\psi$:**

- Fully connected,
- Convolutional/Deconvolutional,
- Recurrent (e.g. RNN, LSTM).

**Regularization:**

- Weight decay,
- Sparse Autoencoders,
- Contractive Autoencoders.

## Next Class: Density Estimation

We have seen how Autoencoders implicitly capture properties of the input distribution.

**Question:** can we sample new points from it?

This is the goal of density estimation approaches:

- Variational Autoencoders,
- Generative Adversarial Networks,
- Energy Models,
- ...