# Deep Learning 1

## Introductory lecture

Siniša Šegvić

University of Zagreb

Faculty of Electrical Engineering and Computing

# Contents

- **Motivation** for deep learning:
  - machine learning and artificial intelligence
  - composite data (consist of parts)
- About **the course**:
  - main topics
  - structure of the course, scoring, literature
- Basics of **machine learning**:
  - basic concepts and techniques
  - examples of algorithms
- Overview of contemporary challenges and benefits

# Motivation: machine learning

- **Machine learning**: express an algorithm by showing some examples (avoid handcrafted rules):
  - one of the central problems of artificial intelligence
- **Artificial intelligence**: studies creation of machines that (appear as if they) can think
  - tasks that are easy for humans but very difficult for computers
- Example: write program to find a cow:
  - intelligent behavior is easier to learn than to construct.

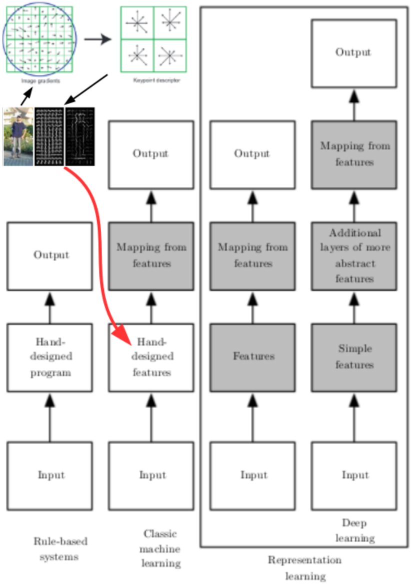

# Motivation: artificial intelligence

Relationship between AI and ML

- early approaches: ~~learning~~
- classical approaches: shallow models, handcrafted features
- representation learning: first the features and then the model
- deep learning: learn everything at once from end to end
  - the model is a sequence of non-linear transformations

**Learning** becomes increasingly important!

**Learning representations** too!




[goodfellow16]

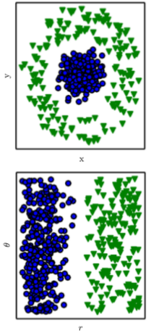
# Motivation: data representation

Computational complexity often depends on data representation:

- for a given task, some representations more suitable than others
- MCMLXXI + XIX vs 1971 + 19?
- polar representation allows a shallow (=linear!) model ↘

Features are often not easily handcrafted

- Greeks and Romans failed to invent a positional number system in more than 1000 years
- that considerably hindered the math development
  - MCMLXXI + XXIX = ?
  - MXXIV : LXIV = ?
- ⇒ learned representations extremely interesting!



[goodfellow16]

# Motivation: example

How to recognize images of bison from images of oxen?

[image-net.org]

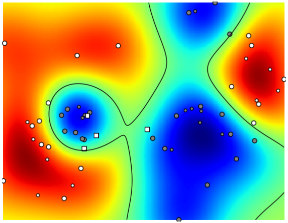
Computational complexity strongly depends on data representation

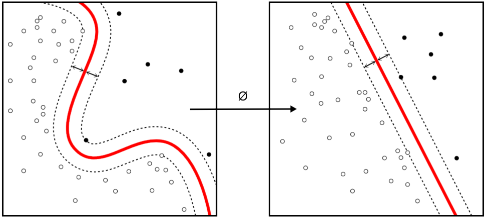
- the task would be easy if some magic algorithm converted the images into binary features: [fur?, hump?, wilderness?, ...]
- most bison would be: [Yes, Yes, Yes, ...]
- most oxen would be: [No, No, No, ...]

Best approach: **jointly** learn the representation and the classifier!

# Motivation: shallow models

- Dominant machine learning approach 1990-2006: handcrafted features and shallow classifiers with convex loss
  - SVM, logistic regression, generalized linear models.
- At the time, advantages of shallow models were clear:
  - learning convergence guaranteed and fast
  - kernel trick provides enormous representational capacity
  - competitive recognition accuracy in practice






[Wikipedia]

- However, these approaches can not distinguish cows from bison...

# Motivation: deep learning

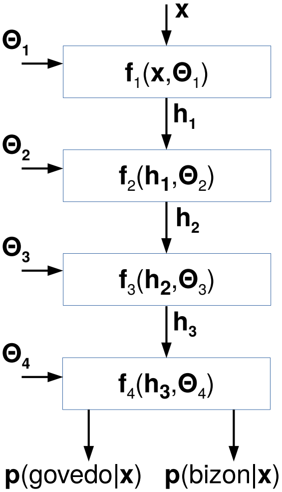
Deep model: a sequence of learned nonlinear transformations

Why was deep learning unpopular?

- no guarantee of learning success
  - non-convex loss $\rightarrow$ local minima
- could not exceed the state of the art

Why has deep learning become successful?

- new modeling and learning techniques
- large datasets (n=$10^6$, $10^9$)
- high processing power (TFLOPS)
  - cuda, cuDNN, OpenBLAS, OpenMP




Applications: understanding of images language and speech, bioinformatics, etc

# Motivation: no free lunch

Deep models did not "work" properly because we did not have:

- enough computational power to afford convergence
- enough data to learn the required capacity
- techniques that promote convergence

However, this does not explain why deep models would generalize better than other high-capacity models (kSVM, trees, ...)

- we cannot design learning algorithms independently from the real data: no free lunch theorem [domingos12cacm]

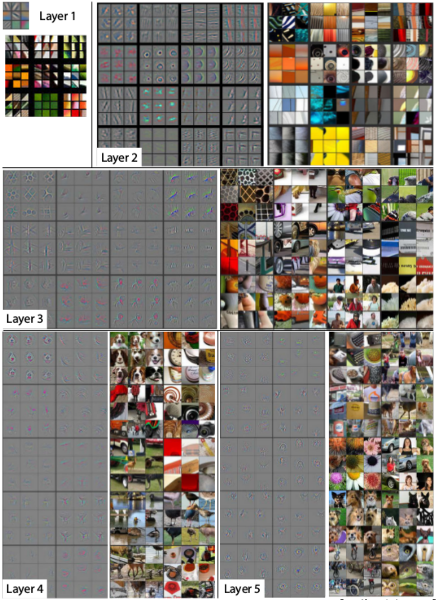
Performance on unseen data depends on the algorithm bias:

- if the bias matches the data - the model will generalize well
- logistic regression: excellent choice for linearly separable data

# Motivation: composite data

**Composite structure**: inherent bias of deep models

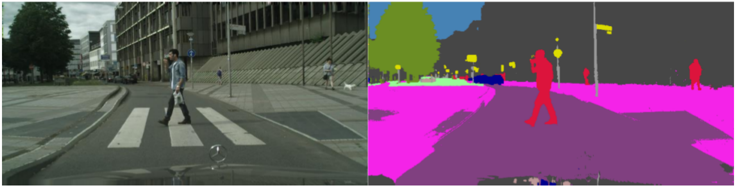
- data representation at level n built from representations at the level n-1
- that property is a good fit in many difficult tasks
  - eg. a motorcycle has wheels, wheels have rims, rims have spokes
  - letters make syllables, syllables - words, words - sentences..




[zeiler14eccv]

# Motivation: deep learning - applications

- Applications:
  - computer vision: image classification, object localizaction, content-based image retrieval, semantička segmentacija



[kreso17iccvw]

  - natural language processing: sentiment analysis, automatic translation, question answering...
  - information retrieval: learning to rank
  - speech recognition

# Motivation: another example

- Let the upper images be negatives and the lower ones - positives:


- The task is to localize objects in test images:

[krapac16gcpr]

# About the course: plan

Topic overview

- Discriminative deep learning:
  - fully connected models
    (lab exercise one)
  - discriminative convolutional models
    (lab exercise two)
  - optimization techniques
  - regularization
- recurrent models
  (lab exercise three)
- metric embeddings
  (lab exercise four)

# About the course: literature

- Deep Learning; Ian Goodfellow, Yoshua Bengio and Aaron Courville; MIT Press; 2016.
- Neural Networks and Deep Learning; Michael Nielsen; Determination press; 2015.
- PyTorch tutorials and documentation
  - `https://pytorch.org/tutorials/`
  - `https://pytorch.org/docs/stable/index.html`

# About the course: prior knowledge

Required prior knowledge to follow the course::

- linear algebra (§2.1 - §2.11) and probability theory (§3.1 - §3.11)
- analysis of multivariate vector-valued functions (§4.3.1)
- basic concepts and techniques of machine learning (§5.1 - §5.11)
- basics of the Python programming language

Lab exercise #0: instrument for checking/acquiring prior knowledge

- you can (should!) start immediately
- if you do not solve the lab #0 - the lab #1 will be too difficult
- all laboratory exercises will be included in the exams

# About the course: lab

There are 4 lab exercises:

0. vector algebra, shalow models, Python, numpy
1. fully connected models
2. basic convolutional models
3. recurrent models
4. metric embeddings

Guidelines:

- the exercises should be solved at home
- they should be submitted before the final exam
  - let us know when you are ready!
- hardware ad software requirements: Python, Numpy, Scipy, Matplotlib and PyTorch

# About the course: lab (2)

Lab exercises are the core component of the course:

- direct 20% points $(4 \times 5)$
  - you must have at least 1/2 of these before taking the final exam
- at least additional 20% points at the exams

There are no points for solving (or not solving) the lab exercise #0.

# About the course: Details

**Activities**: lectures, exercises (Python), exams

**Approximate calendar**:

| | |
|---|---|
| end of October: | L1 |
| mid November: | L2 |
| end of November: | ME |
| start of February: | L3 |
| mid February: | L4 |
| end of February: | FE |
| start of March: | Full |

**Continuous scoring**:

| | |
|---|---|
| lab: | 20 |
| mid-term exam: | 40 |
| final exam: | 40 |
| condition: | 1/2 of the lab points |

**Full exam**:

condition: 1/2 of the lab points

We award **bonus points** for: useful suggestions, seminars, proposals of new problems and exercises

- send e-mail to apply for a seminar before the mid-term exam

**Scoring.** 2: 50%, 3: 63%, 4: 76%, 5: 89%.

## About the course: lecturers and assistants

- lectures:
  - Siniša Šegvić
  - Petra Bevandić
  - Josip Šarić
- labexercises:
  - Petra Bevandić
  - Josip Šarić
  - Marin Kačan, Iva Sović, Ivan Sabolić, Anja Delić, Ivan Martinović.

# Machine learning: basic concepts

Machine learning studies data-processing algorithms that improve with experience

A machine learning algorithm is a meta-algorithm that involves two sub-algorithms:

- the model: a data processing algorithm that will be deployed
  - must supply a **performance** metric
    - eg. classification accuracy on the **test set** $\{\mathbf{x}_i, y_i\}$
    - eg. number of wrongly classified examples
  - we distinguish **empirical** and **generalization** performance
- the optimizer: fits the model to the data (eg. gradijent descent)
  - requires a **training dataset** $\{\mathbf{x}_i, y_i\}$
    - should be sampled from the same distribution as the test set!
  - requires an optimization objective (loss)
    - eg. negative log-likelihood of parameters on the training set

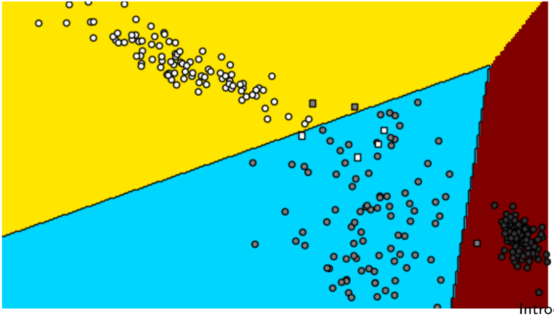
# Machine learning: definitions

- In machine learning, a parametric (meta-)algorithm is defined with:
  - model: a data-processing algorithm with free parameters
  - loss: formalized (anti-)goodness of fit of the model parameters
  - optimizer: finds parameters that make the loss acceptable (not necessarily minimal).
- Regarding the quality of learning data, we distinguish:
  - supervised learning: we have a desired output in each learning example
    - typical tasks: classification, regression
  - unsupervised learning: we have only the data
    - typical tasks: density estimation, data generation.
  - reinforcement learning: we receive the feedback after a number of model evaluations.

# Machine learning: example

Logistic regression: supervised multi-class classification

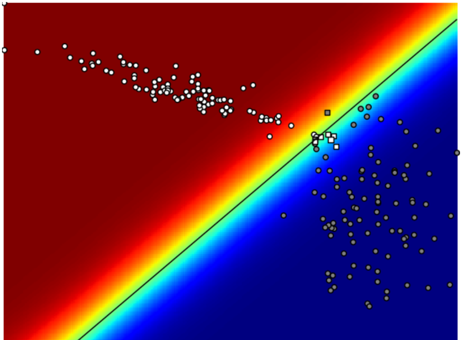
- the model returns the posterior distribution over the training taxonomy:
  $P(Y \mid \mathbf{x}) = \mathrm{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$, $\mathrm{softmax}(\mathbf{s}) = [e^{s_j} / \sum_k e^{s_k}]^\top$.
- loss (negative log-likelihood on the training set):
  $\mathcal{L}(\mathbf{W}, \mathbf{b} \mid \mathbf{Y}, \mathbf{X}) = -\sum_i \log P(Y = y_i \mid \mathbf{x}_i)$
- optimizer (gradient descent): $\mathbf{b}_{i+1} = \mathbf{b}_i - \delta \cdot \nabla_{\mathbf{b}_i}\mathcal{L} = \mathbf{b}_i - \delta \cdot (\frac{\partial \mathcal{L}}{\partial \mathbf{b}_i})^\top$



# Machine learning: example 2

Support vector machine (supervised binary classification):

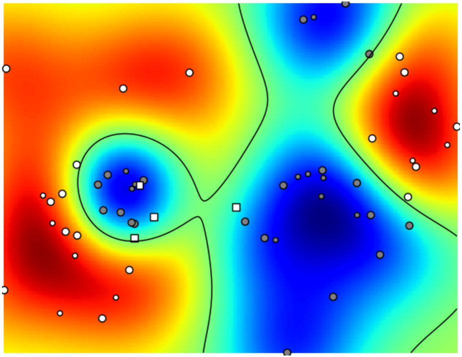
- the model (binary classifier): $f(\mathbf{x}) = \begin{cases} c_0 & \text{if } \mathbf{w}^\top \mathbf{x} + b < 0 \\ c_1 & \text{if } \mathbf{w}^\top \mathbf{x} + b > 0 \end{cases}$
- the loss (total margin violation plus regularization):
  $\mathcal{L}(\mathbf{w}, b \mid \mathbf{Y}, \mathbf{X}) = \lambda \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 + (-1)^{[\![y_i = c_1]\!]} (\mathbf{w}^\top \mathbf{x}_i + b)\right)$
- optimizer: quadratic programming or gradient descent



# Machine learning: example 3

SVM with support vectors $\mathbf{x}_i$ and kernel function k:

- model: $f(\mathbf{x}) = \begin{cases} c_0 & \text{if } \sum_i \alpha_i \cdot k(\mathbf{x}_i, \mathbf{x}) + b < 0 \\ c_1 & \text{if } \sum_i \alpha_i \cdot k(\mathbf{x}_i, \mathbf{x}) + b > 0 \end{cases}$
- loss (regularization + margin violation): $\mathcal{L}(\alpha, \mathbf{b} \mid \mathbf{Y}, \mathbf{X}) = h(\alpha\alpha^\top) + \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 + (-1)^{[\![y_i = c_1]\!]}(\alpha_i \cdot k(\mathbf{x}_i, \mathbf{x}) + b)\right)$
- optimization: quadratic programming or gradient descent



# Machine learning: indirect optimization

Peculiarity of machine learning: we optimize performance indirectly

- the optimization method can not "see" the loss on the test set
  - but it presumes that the empirical distribution corresponds to the generative distribution
  - ie. the learning and testing data are generated by the same random process
- the loss often cannot be equated with the empirical error
  - typically because the error formulation is not differentiable
  - in this case the loss is a proxy for the empirical error
  - a well-defined replacement loss can improve the generalization even after the empirical error drops to zero!

# Machine learning: capacity

Capacity as the basic property of the model:

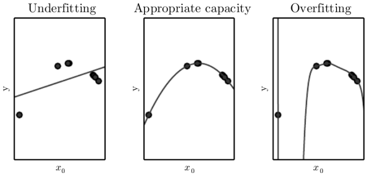
- describes the ability to adapt to data
- usually proportional to the number of degrees of freedom
- can be described as the number of examples (VC dimension) which the model can **shatter** (ie. explain with arbitrary labels)

Low capacity models prone to undertraining:

- there is no set of parameters that can explain the learning set

High-capacity models are prone to overtraining:

- too many sets of parameter can explain the training data.

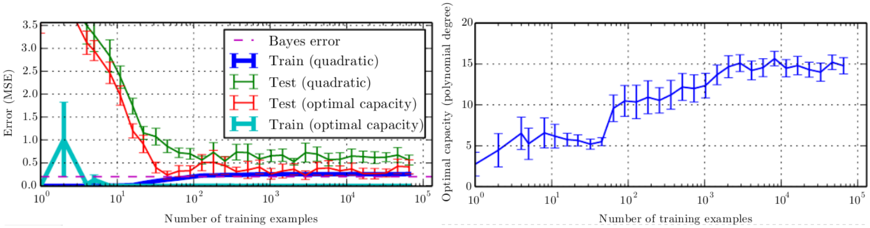



[goodfellow16]

# Machine learning: impact of data

The accuracy of the learned model depends on the number of data

- introduce the inevitable (Bayesian) error
    - it occurs due to stochastic data or labeling noise
- the undercapacitated algorithm converges to:
    - empirical error that is noticeable (dark blue)
    - generalization error that is greater than the unavoidable (green)
- a model with excess capacity may be better than model with the same complexity as the noisy generative process (n=5, right)




[goodfellow16]

# Machine learning: lex parsimonae

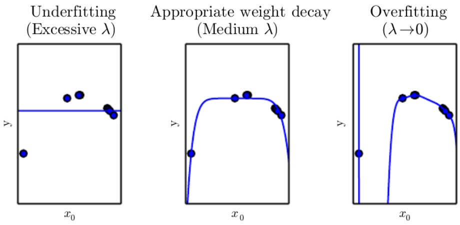
Limiting model capacity is not the only way to match the data complexity

Another way is to keep the high capacity but introduce a preference towards simpler models

If we look at regression, one way to regularize the loss would be:

$$J(\mathbf{w}) = \lambda \mathbf{w}^\top \mathbf{w} + \sum_i (\sum_j w_j {x_i}^j + b - y_i)^2.$$

The algorithm now prefers solutions where the input changes lead to smooth changes in the model output



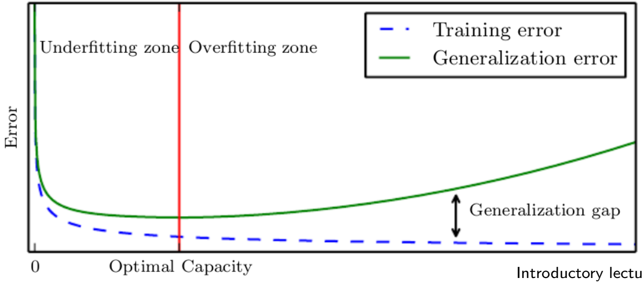

[goodfellow16]

# Machine learning: regularization

Regularization is any modification aimed to improve generalization without reducing the empirical error

Regularization can be applied to all components of learning:

- loss: penalizing the norm of the parameters
- optimizer: early stopping
- data: jittering, label smoothing
- model: reduce capacity, parameter sharing



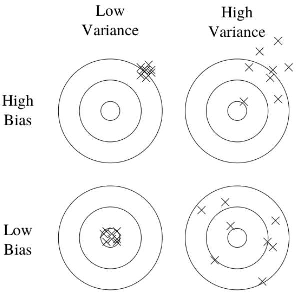
# Machine learning: statistical view

Under-training and overfitting can be clarified by explaining the generalization error with bias and variance

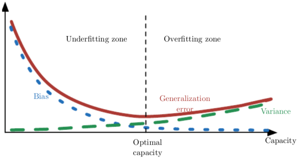
Bias: a built-in tendency towards some solutions.

Variance: variation due to different training data.

Regularization reduces the variance and increases the bias.




[domingos16cacm]




[goodfellow16]

Regularization should be adjusted to the data:

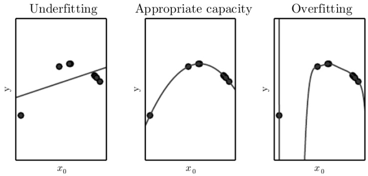
- decrease the variance while avoiding inappropriate bias

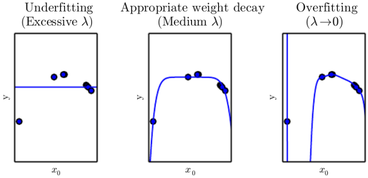
# Machine learning: hyperparameters

Hiperparameters regulate the algorithm behaviour, but they are not affected by optimization.

Examples: model complexity, factor of the parameter norm, optimization step, number of epochs...

Hyperparameters are often adjusted through exhaustive or random search on the validation subset




[goodfellow16]

# Machine learning: loss

The most intuitive loss is the mean square error:

$$J(\theta) = \sum_i (\text{model}(\mathbf{x}_i) - y_i)^2$$

This loss is not suitable for probabilistic classification because it ignores that the model returns a distribution

- eg. $\mathrm{d}^2_{L2}$([1,0,0], [0.2, 0.4, 0.4]) < $\mathrm{d}^2_{L2}$([1,0,0], [0.2, 0.0, 0.8])

A more consistent formulation of the classification loss is negative log-likelihood of model parameters:

$$J(\theta) = -\frac{1}{N} \sum_i \log \mathrm{P}_{\text{model}}(y_i | \mathbf{x}_i, \theta).$$

It can be shown that the negative log-likelihood is a special case of cross-entropy (the equivalent of KL divergence).

If we model the regression deviation with a Gaussian distribution, negative log-likelihood becomes **squared loss**.

# Machine learning: SGD

One of the currently most popular optimization methods in machine learning is stochastic gradient descent

Gradient descent by negative log-likelihood must calculate:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i^N \nabla_\theta L(\mathbf{x}_i, y_i, \theta).$$

Gradient descent may become slow when $N \cdot \dim(\theta) \sim 10^{12}$

We solve the problem by separating the data into batches:

$$\nabla_\theta J(\theta) = \frac{1}{N'}\sum_i^{N'} \nabla_\theta L(\mathbf{x}_i, y_i, \theta).$$

- the optimization step is performed in time O(N'·dim($\theta$)), N'<<N
- groups are randomly formed after each epoch (stochastics!)
- in large models (dim($\theta$) $\sim 10^6$): faster than higher order methods
- also used for large shallow models!
- standard kSVM is $\sim O(N^2)$ in space, and $\sim O(N^3)$ in time.

# Towards deep learning: challenge

Problems at AI-level consider complex data (D=dim$(\mathbf{x}_i) \sim 10^5$)

Therefore, the number of all possible data is at least $O(2^D)$

If the bias of the classifier does not match the bias of the data, we must have a representative in each hyper-cube of the data space

- in the pessimistic case we need $O(2^D)$ training examples!
- this is a form of the curse of dimensionality

[goodfellow16]

# Towards deep learning: classical answer

Classical approaches assume smoothness (or local constancy):

*the model should not change very much within a small region.*

Such prior is endorsed by k-NN, kernel methods, trees:

- all these approaches require O(n) examples to discriminate O(n) regions in data space
- such approaches cannot work when $n = 2^{10^5}$

We can draw the following conclusions:

- smoothness prior does not hurt, but it can't handle the increase in dimensionality
- we need appropriate kinds of **bias** for **real data.**

# Towards deep learning: composite data

A fundamental assumption of deep models: the data is generated through recursive composition of parts

- a person has a head, a head has a face, a face has eyes

Potential for independent learning of lower level features:

- a person with blue eyes and black hair can contribute to recognizing people with blue eyes and red hair.

We can express deep models with fewer parameters:

- eg. learn $xor_n$ as a sum of minterms: $\mathrm{xor}_n(\mathbf{b}) = \sum_{j=1}^{2^n} w_j \cdot m_j$
  - $\rightarrow 2^n$ binary parameters
- eg. learn xor as a composition of two-way logical functions $f_i$:
  - $\mathrm{xor}_n(\mathbf{b}) = f_1(b_1, f_2(b_2, ...f_{n-1}(b_{n-1}, b_n)...))$
  - $\rightarrow 4n$ parameters

Such approach can counter the curse of dimensionality.

# Towards deep learning: composite data (2)

**Composite structure**: intrinsic bias of deep models

- data representation at level n built from representations at level n-1
- that property of deep models fits the data in many difficult tasks
  - eg. a motorcycle has wheels, wheels have rims, rims have spokes
  - letters make syllables, syllables - words, words - sentences..


[zeiler14eccv]

# Towards deep learning: manifold learning

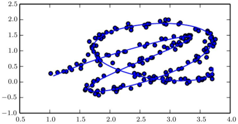
Manifold: a connected subset $\{\mathbf{x}_i\} \in \mathbb{R}^n$ that can locally be approximated with $\{\mathbf{x}'_i\} \in \mathbb{R}^m$, $m << n$ (left!).

Composite data reside at a particular manifold:

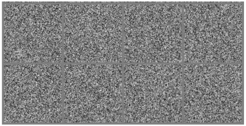
- eg. no people have eyes on their legs
- the model can specialize for dense portions of the data space

An intuition that the manifold assumption holds in practice:

- most of all possible input vectors are not valid data (middle!)
- we can imagine independent **factors of variation** that define local axes of the manifold: brightness, contrast, rotation (right!) etc.




[goodfellow16]



# Towards deep learning: conclusion

Deep models are **biased**: they work best with data that consist of parts

- it makes sense to use them when the data is **composite**
- otherwise, better results could be achieved by shallow models

Deep models are **scalable**, they can work with:

- high-dimensional data (D=$10^5$)
- large training datasets (N=$10^6$)
- huge numbers of parameters (dim($\theta$)=$10^9$)

This makes deep models **a method of choice** in many problems of artificial intelligence.