Deep Architecture

Monean

WHAT IS DEEP LEARNING

- Deep learning, an approach to AI based on enabling computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts.

Several artificial intelligence projects have sought to hard-code knowledge about the world in formal languages.

The difficulties faced by systems relying on hard-coded knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data.

This capability is known as machine learning.

What we call a learning machine or more generally learner is the agent that executes the learning procedure, that takes training data as input and yields a change in the agent

The performance of these simple machine learning algorithms depends heavily on the representation of the data they are given.

Data can be represented in different ways, but some representations make it easier for machine learning algorithms to capture the knowledge they provide.

Binary encoding

In many cases, the compact binary representation is
a poor choice for learning algorithms,

eg: 3,encoded as binary 00000011
4,encoded as binary 00000100)
have no digits in common

while two values that are very different
(like binary 10000001 = 129 and binary 00000001 = 1) only differ by one digit.

representation learning

Representation learning algorithms can either be supervised, unsupervised, or a combination of both (semi-supervised).

Supervised learning requires examples that include both an input and a target output, the latter being generally interpreted as what we would have liked the learner to produce as output, given that input. Such examples are called labeled examples because the target output often comes from a human providing that “right answer”.

representation learning

Unsupervised learning allows a learner to capture statistical dependencies present in unlabeled data, while semi-supervised learning combines labeled examples and unlabeled examples.

Machine Learning

A popular definition of learning in the context of computer programs is "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T , as measured by P, improves with experience E" (Mitchell, 1997).

Manifold Learning

- The manifold learning states that probability is concentrated around regions called manifolds, i.e., that most configurations are unlikely and that probable configurations are neighbors of other probable configurations. (Cayton, 2005; Narayanan and Mitter, 2010)
- The manifold hypothesis also states that small changes (e.g. translating an input image) tend to leave unchanged categorical variables (e.g., object identity) and that there are much fewer such local degrees of freedom (manifold dimensions) than the overall input dimension (the number of observed variables).

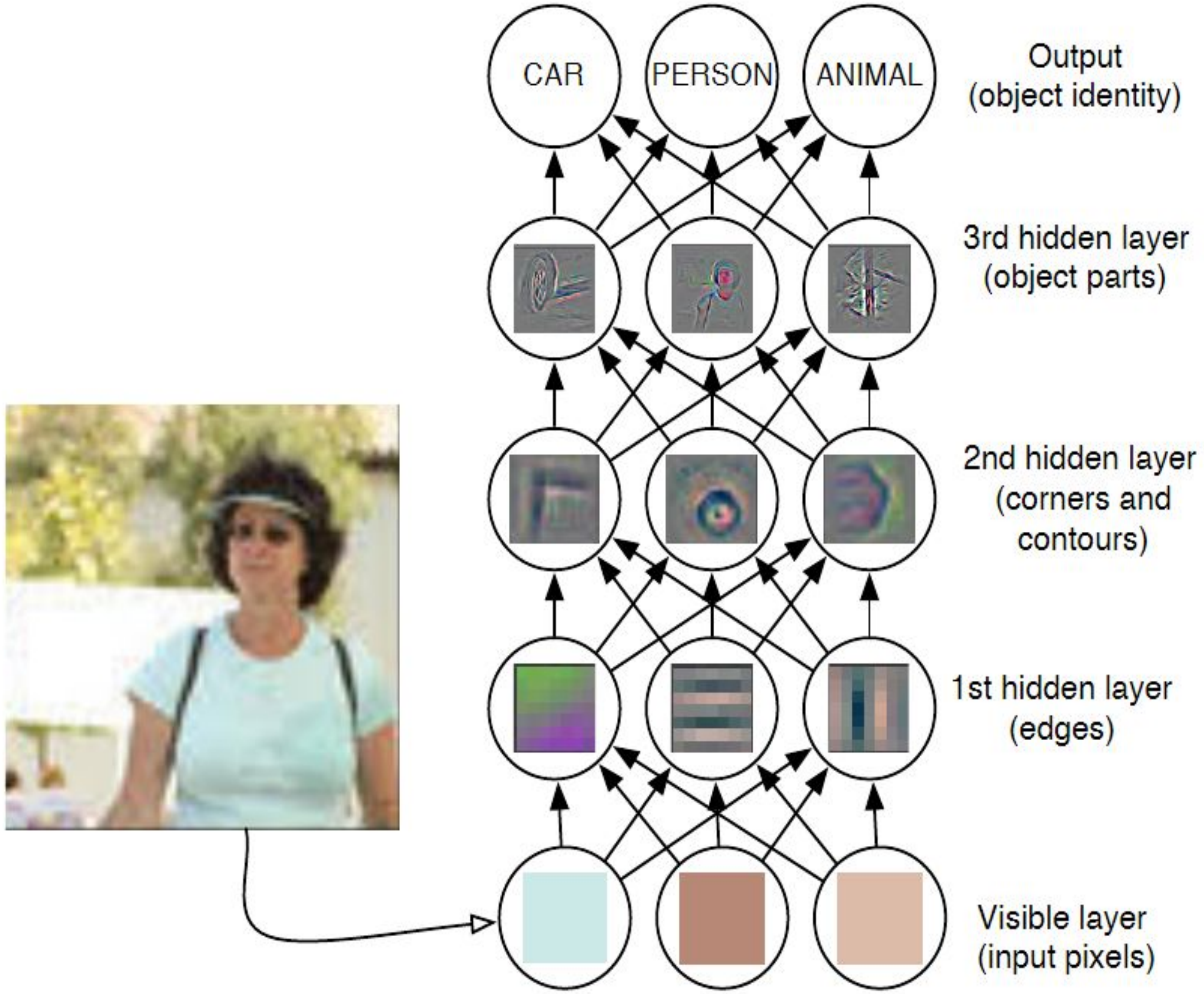
These ideas turn out to be very important to understand the basic concept of representation associated with deep learning algorithms, which may be understood as a way to specify a coordinate system along these manifolds, as well as telling to which manifold the example belongs.

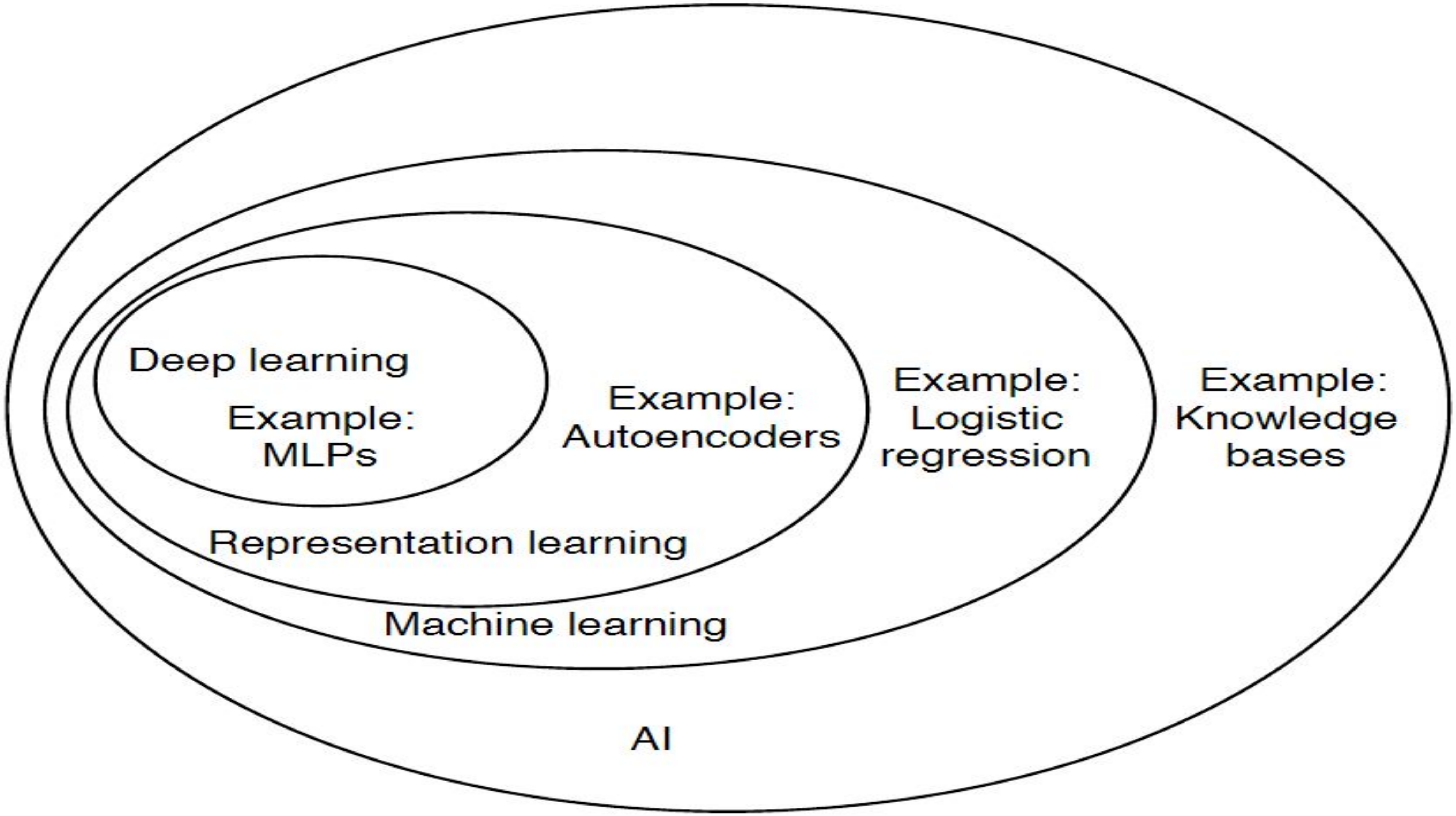
Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations.

"Depth" is not a mathematically rigorous term in this context; there is no formal definition of deep learning. All approaches to deep learning share the idea of nested representations of data, but different approaches view depth in different ways.

For some approaches, the depth of the system is the depth of the flowchart describing the computations needed to produce the final representation. Other approaches consider depth to be the depth of the graph describing how concepts are related to each other.

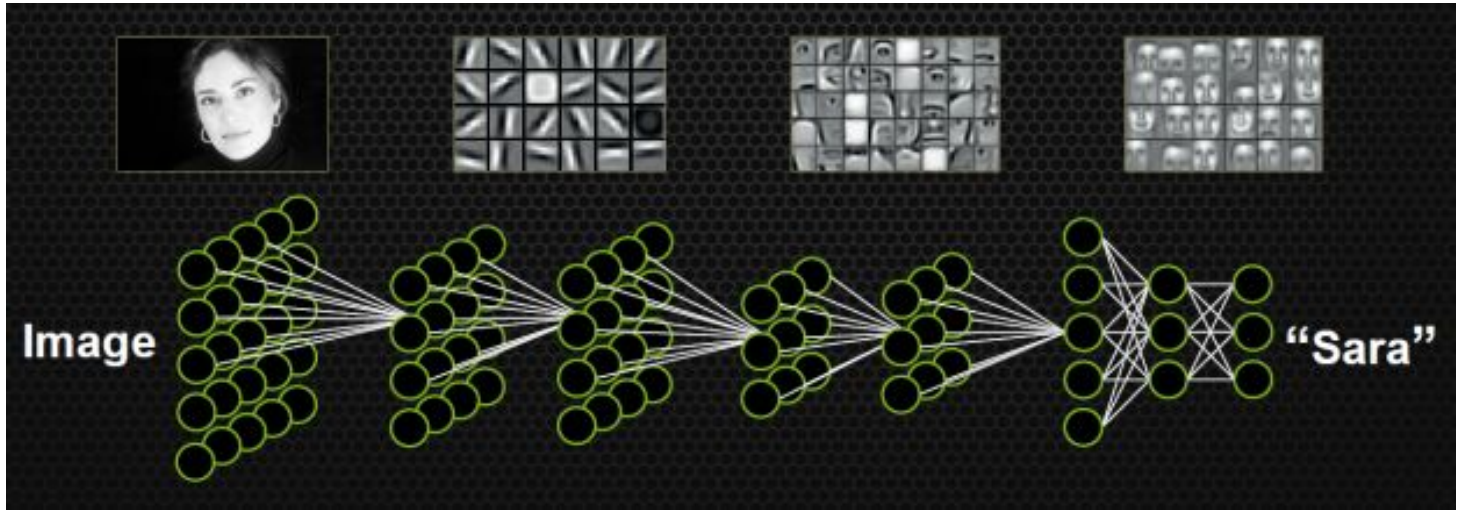
Deep learning resolves this di fficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model. The input is presented at the visible layer. Then a series of hidden layers extracts increasingly abstract features from the image.


CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visible layer (input pixels)


Deep learning
Example:
MLPs
Example:
Autoencoders
Example:
Logistic
regression
Example:
Knowledge
bases
Representation learning
Machine learning
AI

Convolution Networks

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication.



Convolution Operation

Suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements.

$$s(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da$$

w(a)	: weighting function
a	: the age of a measurement

$$s(t) = (x * w)(t)$$

Discrete Convolution

$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

Two dimensional discrete convolution :

$$s[i,j] = (I * K)[i,j] = \sum_{m}\sum_{n} I[m,n]\underbrace{K[i-m,j-n]}_{\text{kernel}}$$

$$\Vert$$

$$s[i,j] = (I * K)[i,j] = \sum_{m}\sum_{n} I[i-m,j-n]K[m,n]$$

Motivation

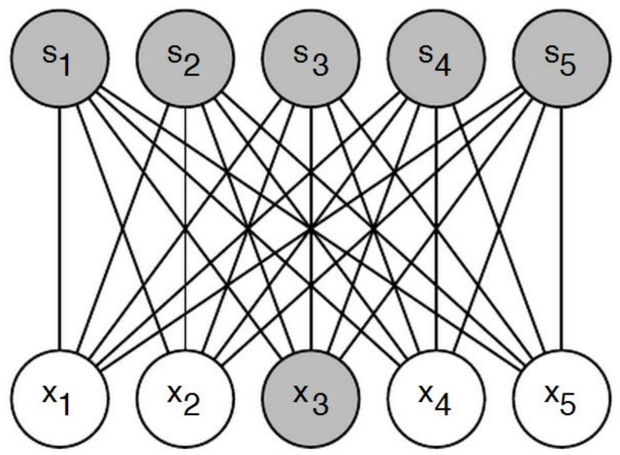
Convolution leverages three important ideas that can help improve a machine learning system: **sparse interactions**, **parameter sharing**, and **equivariant representations**

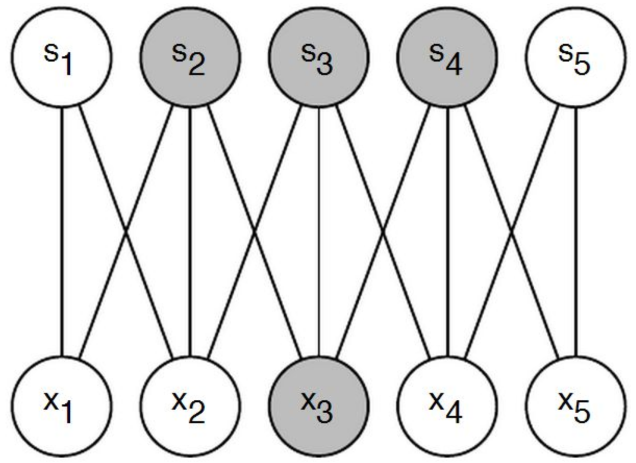
Traditional neural network layers use a **matrix multiplication** to describe the interaction between each input unit and each output unit. This means every output unit interacts with every input unit.

Convolutional networks, however, typically have **sparse interactions** (also referred to as sparse connectivity or sparse weights). This is accomplished by making **the kernel smaller** than the input.

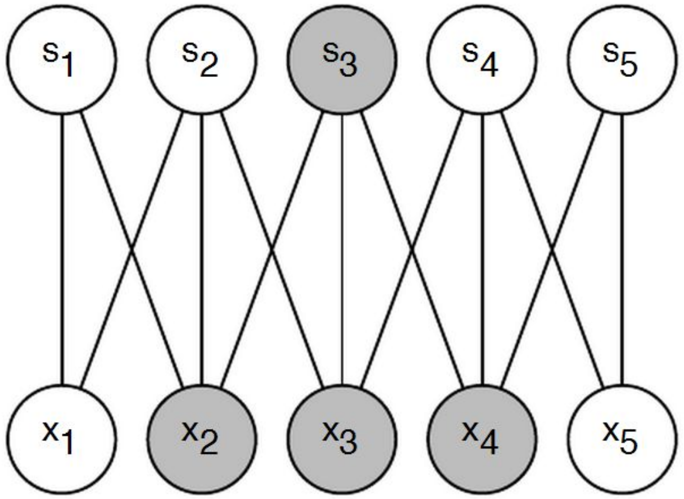
eg. processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features

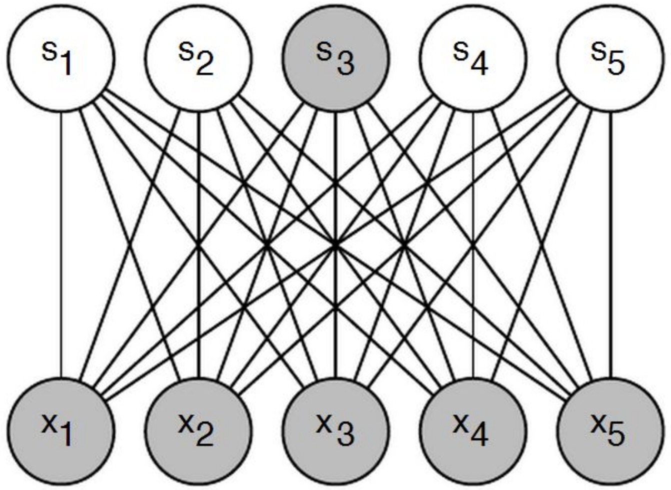
Sparse Connectivity





Sparse connectivity, viewed from above: Highlight one input unit, x3, and also highlight the output units in s that are affected by this unit. **(Left)** When s is formed by **convolution with a kernel of width 3**, only three outputs are aff ected by x3. **(Right)** When s is formed by **matrix multiplication**, connectivity is no longer sparse, so all of the outputs are affected by x3.





Sparse connectivity, viewed from above: We highlight one output unit, s3, and also highlight the input units in x that affect this unit. These units are known as the **receptive field** of s3. **(Left)** When s is formed by convolution with a kernel of width 3, only three inputs affect s3. **(Right)** When s is formed by **matrix multiplication**, connectivity is no longer sparse, so all of the inputs affect s3.

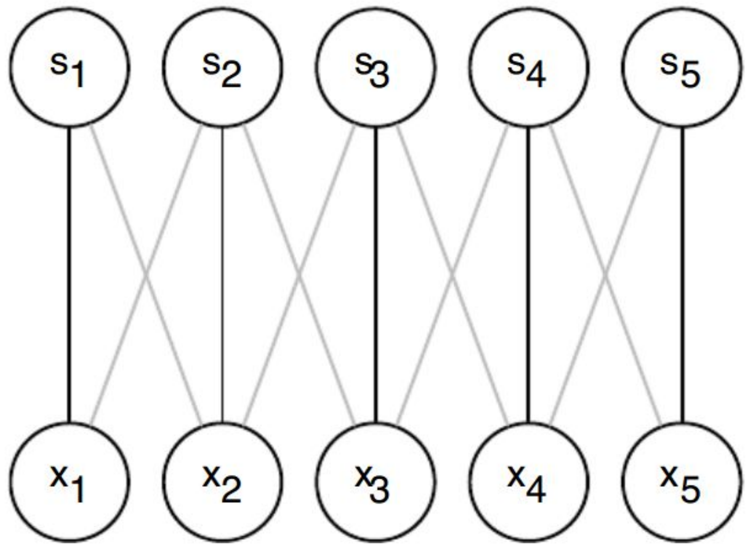
Parameter Sharing

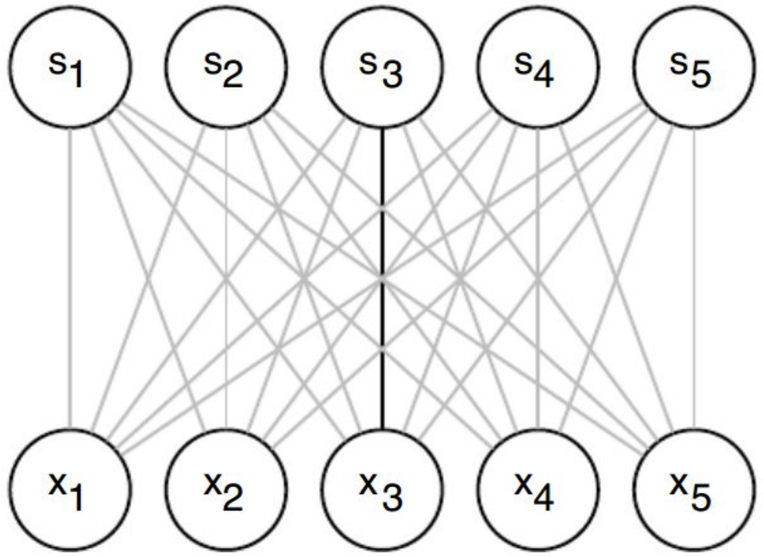
Parameter Sharing : Using the same parameter for more than one function in a model.

In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input, and then never revisited.

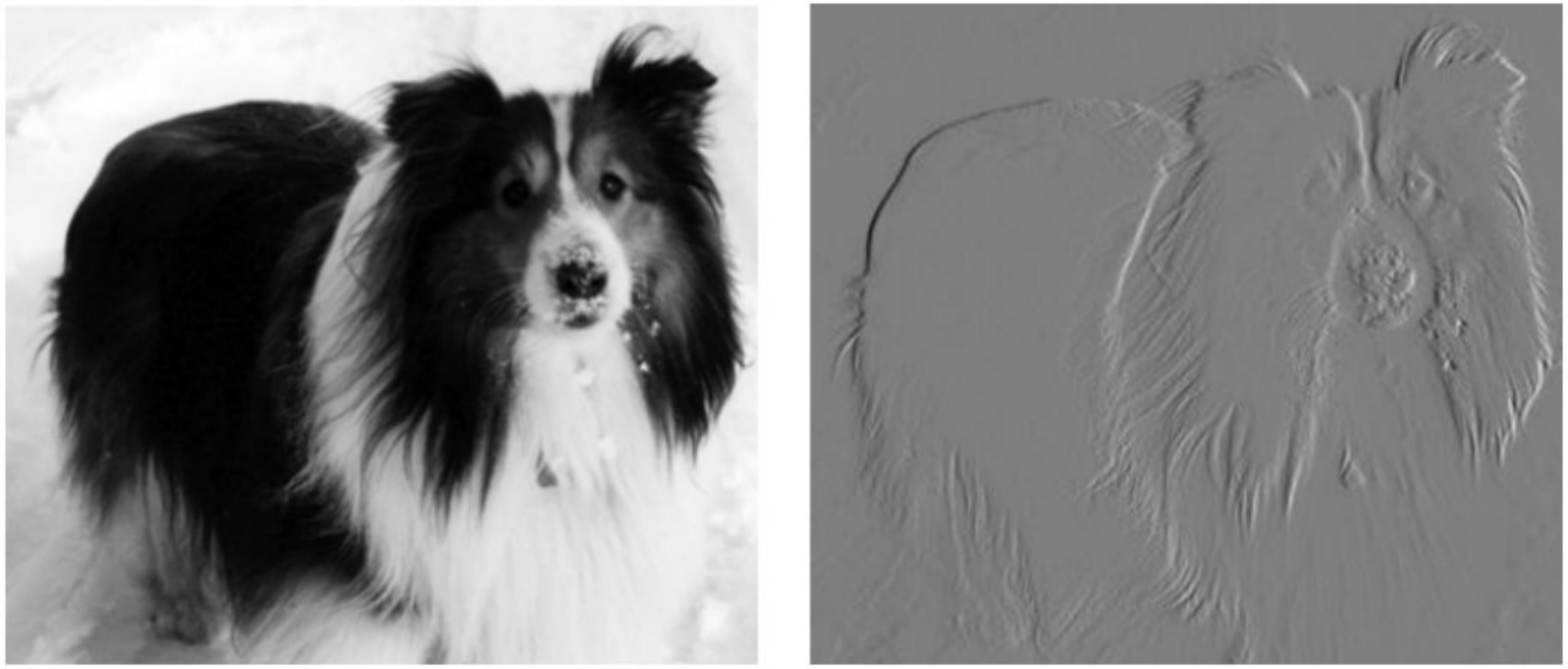
As a synonym for parameter sharing, one can say that a network has **tied weights**, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural net, each member of the kernel is used at every position of the input

Parameter sharing:





Parameter sharing: We highlight the connections that use a particular parameter in two different models. (Left) We highlight uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations. (Right) We highlight the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.



Efficiency of edge detection. The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small, local region across the entire input.

Equivariant Representations

Figure above shows how sparse connectivity and parameter sharing can dramatically improve the efficiency of a linear function for detecting edges in an image.

In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation.

Equivariant

Definition:

To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function f(x) is equivariant to a function g if :

$$f(g(x)) = g(f(x))$$

if we let g be any function that translate the input, i.e., shifts it

Then the convolution function is equivariant to g. For example, define g(x) such that for all i, $g(x)[i] = x[i - 1]$. This shifts every element of x one unit to the right. If we apply this transformation to x, then apply convolution, the result will be the same as if we applied convolution to x, then applied the transformation to the output.

Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.

For example, when processing images, it is useful to detect edges in the first layer of a convolutional network, and an edge looks the same regardless of where it appears in the image.

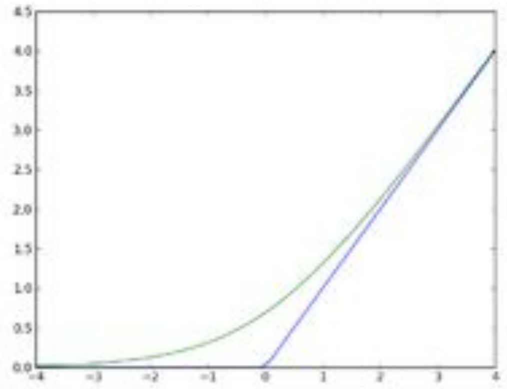
Note that convolution is not equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations

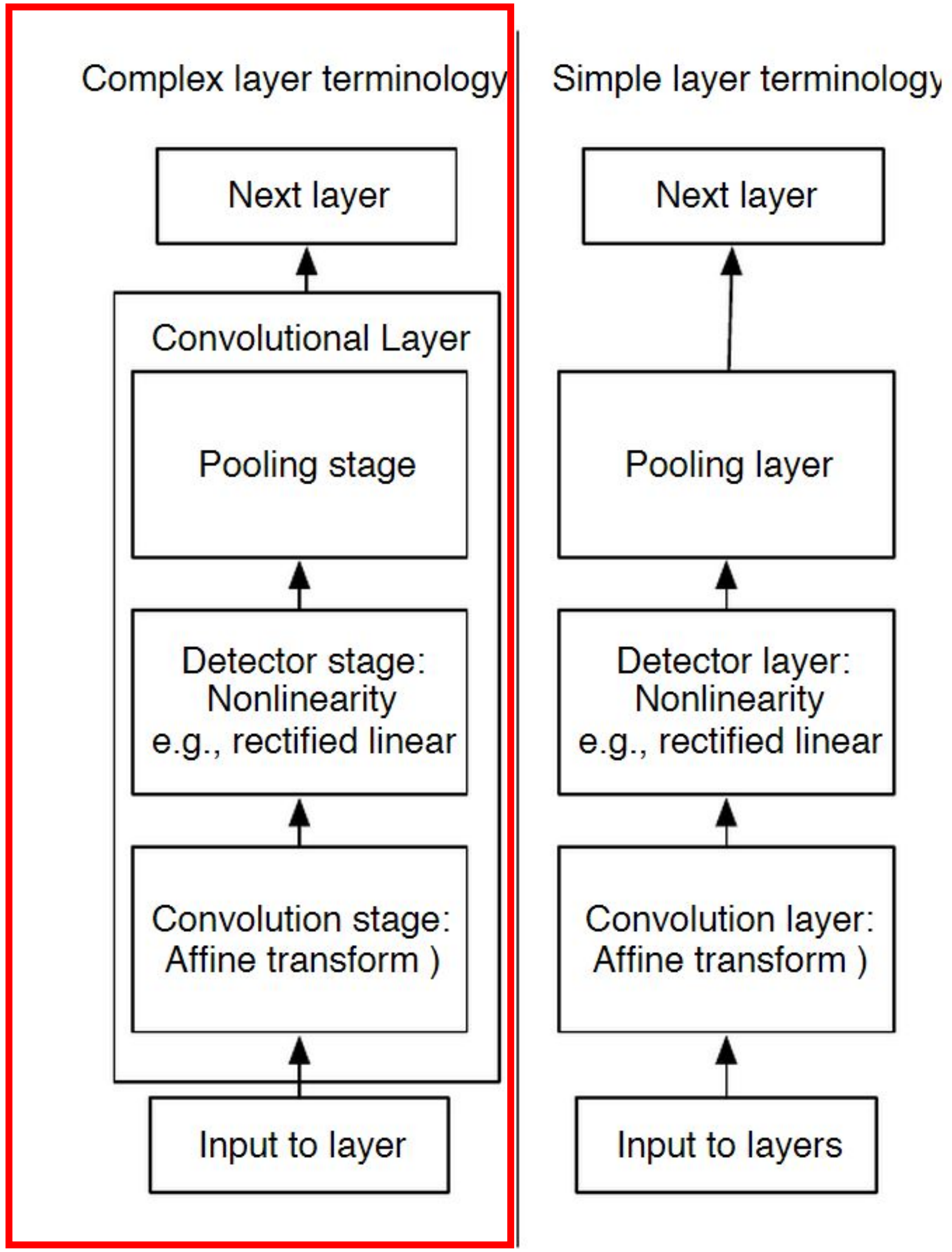
Pooling

A typical layer of a convolutional network consists of three stages

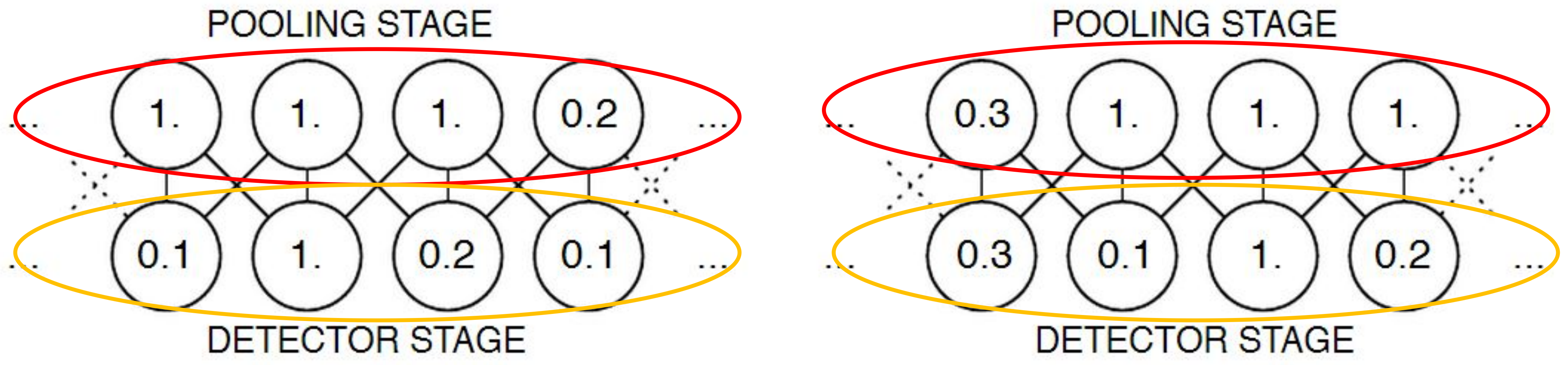
rectifier is an activation function defined as

$$f(x) = \max(0, x)$$



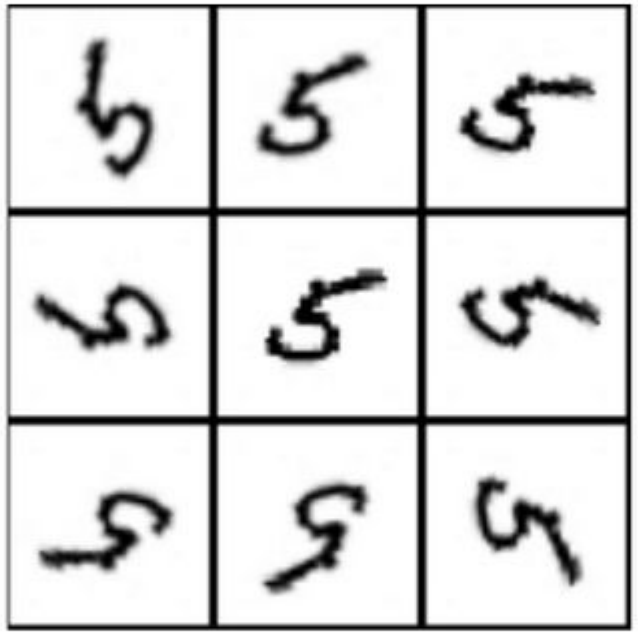

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform)
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform)
Input to layers

Pooling helps to make the representation become invariant to small translations of the input. This means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.



Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

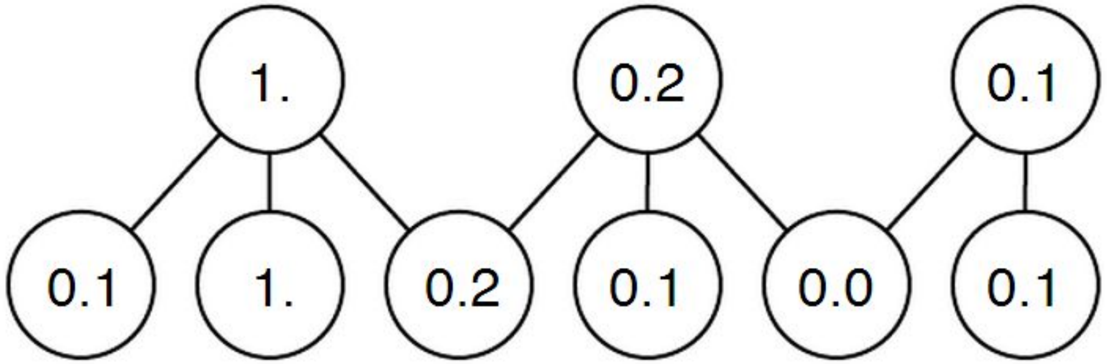
For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face.



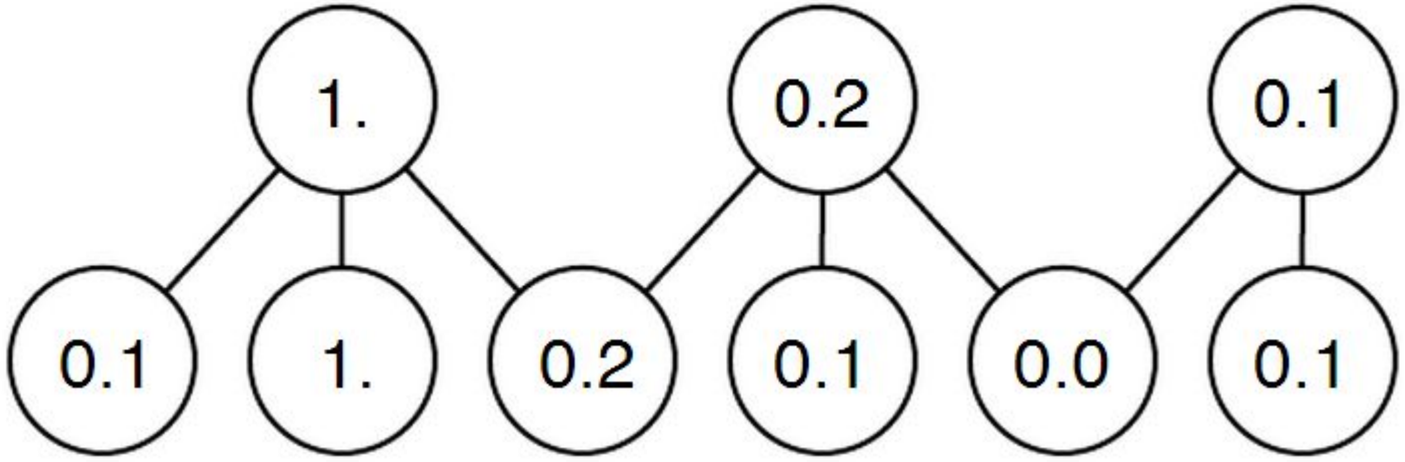
Example of learned invariances: If each of these fi lters drive units that appear in the same max-pooling region, then the pooling unit will detect "5"s in any rotation. By learning to have each filter be a diff erent rotation of the "5" template, this pooling unit has learned to be invariant to rotation. This is in contrast to translation invariance, which is usually achieved by hard-coding the net to pool over shifted versions of a single learned filter.

Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced k pixels apart rather than 1 pixel apart.

Downsampling



downsampling



Pooling with downsampling. Here we use max-pooling with a pool width of 3 and a stride between pools of 2. This reduces the representation size by a factor of 2, which reduces the computational and statistical burden on the next layer. Note that the final pool has a smaller size, but must be included if we do not want to ignore some of the detector units.

downsampling

This improves the computational efficiency of the network because the next layer has roughly k times fewer inputs to process. When the number of parameters in the next layer is a function of its input size (such as when the next layer is fully connected and based on matrix multiplication) this reduction in the input size can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.

Manifold perspective

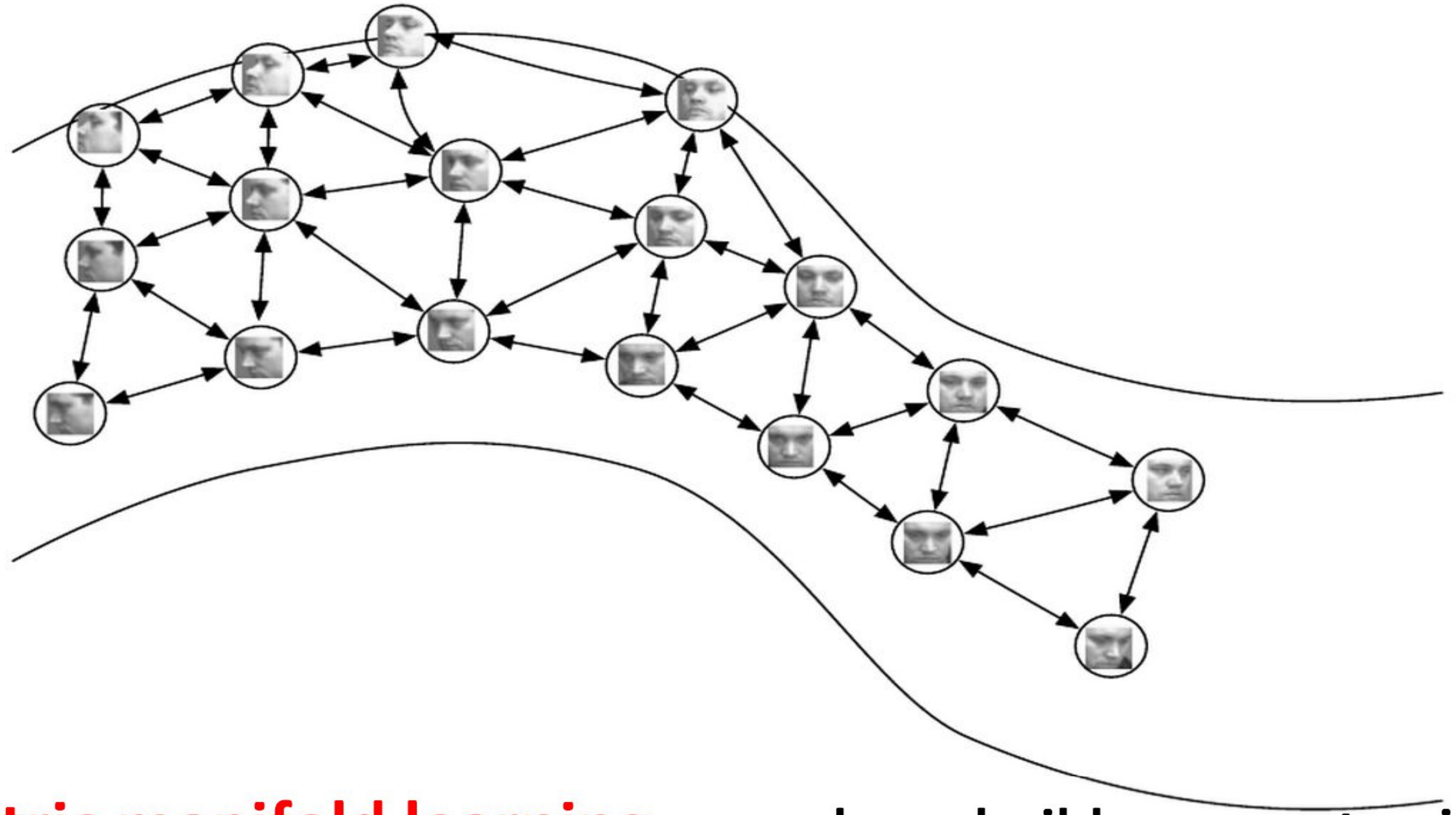
Manifold learning is an approach to machine learning that is capitalizing on the manifold hypothesis (Cayton, 2005; Narayanan and Mitter, 2010): the data generating distribution is assumed to concentrate near regions of low dimensionality.

The notion of manifold in mathematics refers to continuous spaces that locally resemble Euclidean space, and the term we should be using is really submanifold, which corresponds to a subset which has a manifold structure. The use of the term manifold in machine learning is much looser than its use in mathematics

- the data may not be strictly on the manifold, but only near it,
- the dimensionality may not be the same everywhere,
- the notion actually referred to in machine learning naturally extends to discrete spaces.

manifold hypothesis: when a configuration is probable it is generally surrounded (at least in some directions) by other probable configurations.

e.g. : If a configuration of pixels looks like a natural image, then there are tiny changes one can make to the image (like translating everything by 0.1 pixel to the left) which yield another natural-looking image.



Non-parametric manifold learning procedures build a nearest neighbor graph whose nodes are training examples and arcs connect nearest neighbors. Various procedures can thus obtain the tangent plane associated with a neighborhood of the graph, and a coordinate system that associates each training example with a real-valued vector position, or embedding. It is possible to generalize such a representation to new examples by a form of interpolation. So long as the number of examples is large enough to cover the curvature and twists of the manifold, these approaches work well. Images from the QMUL Multiview Face Dataset (Gong et al., 2000)

Manifold learning OR PCA ?

有关于数据挖掘小组的读书会

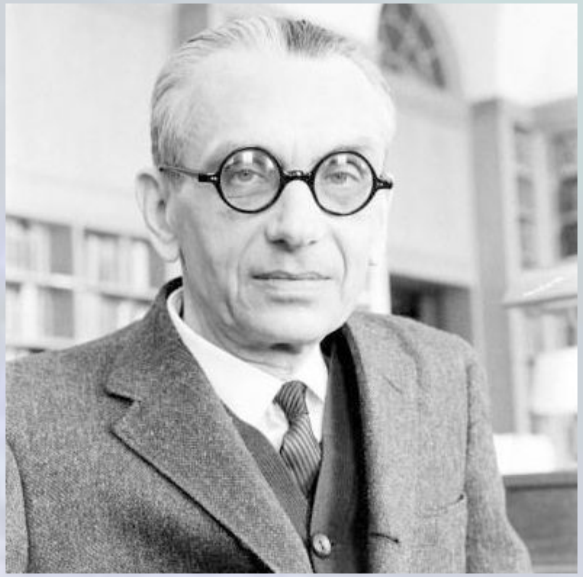
Less is more, learning to be great

无尽的圈~~~

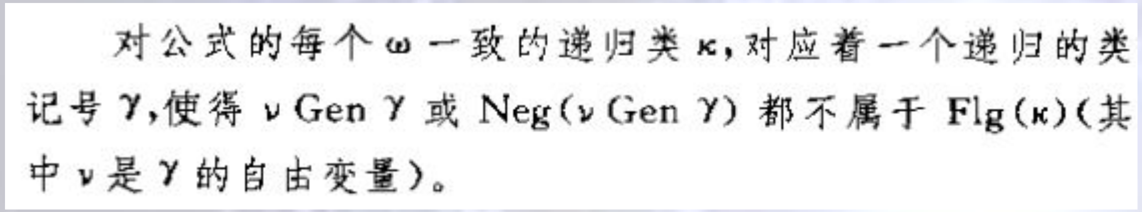
艾皮曼尼蒂斯：“说谎者悖论”

所有的克里特岛人都是说谎者

库特·哥德尔

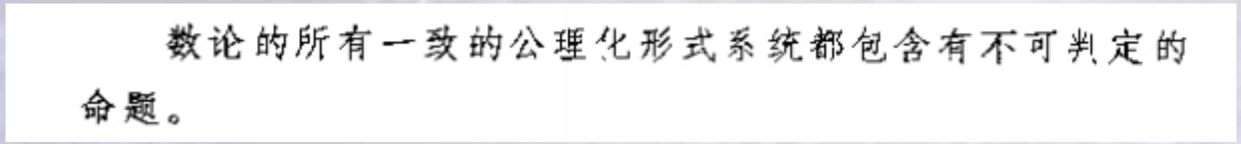


哥德尔不完全性定理：



对公式的每个 ω 一致的递归类 κ，对应着一个递归的类记号 γ，使得 ν Gen γ 或 Neg(ν Gen γ) 都不属于 Flg(κ)(其中 ν 是 γ 的自由变量)。

||



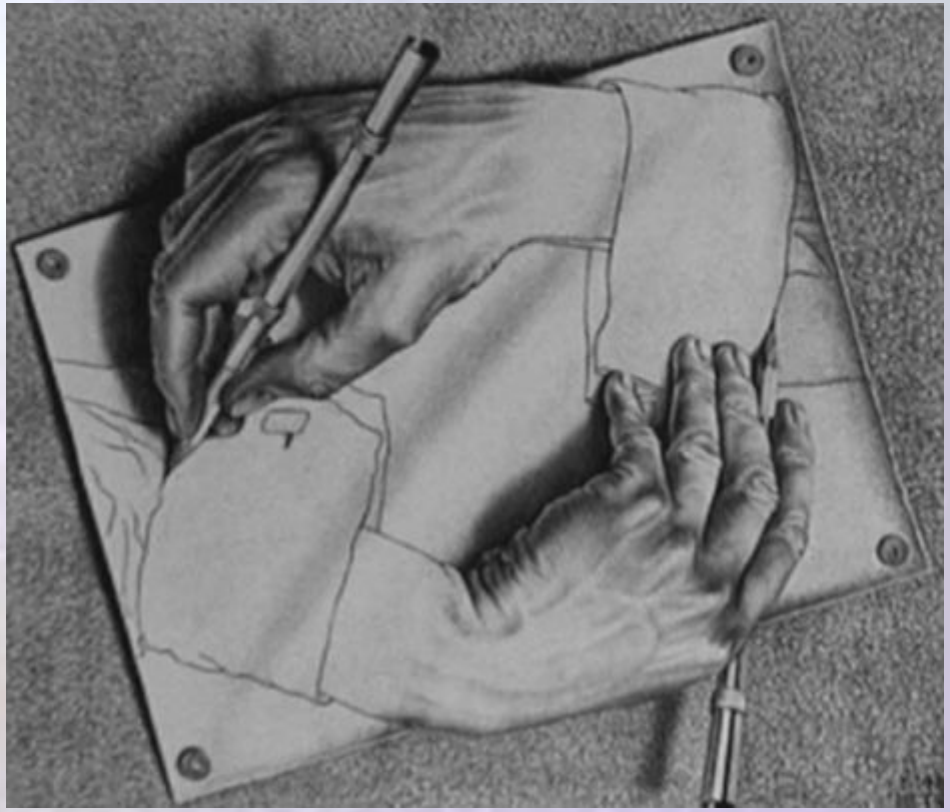
数论的所有一致的公理化形式系统都包含有不可判定的命题。

无论涉及什么样的公理系统，可证性总是比真理性要弱。。。

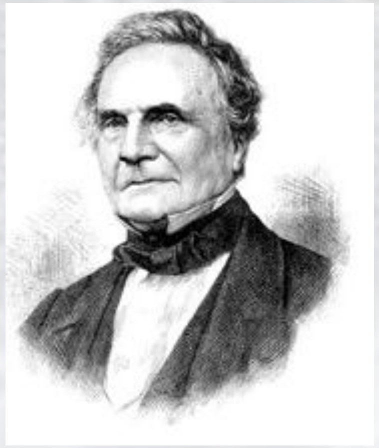
悖论的祸根——自指（怪圈）

下面这个句子是假的

上面这个句子是真的



查尔斯▪巴比奇与“分析机”

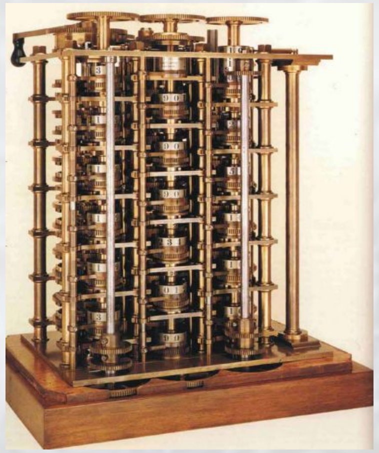


雅卡提花机：一种卡片控制的提花机，能够织出惊人的复杂图案

发现哥德尔定理在计算机中的对应阿兰 ▪图灵

图灵的发现：即便是在可以设想出来的性能最好的计算机中，也存在有不可避免的漏洞

讽刺的是，在这些怪诞的局限性被发现的同时，人们有不断制造出性能越来越好的计算机，以至于远远超出了制造者们的预见力。



五十年代初期，机械化智能似乎已经指日可待了

在创造最终的真正的思维机器时，没跨越一个障碍都要产生一个新的障碍。我们所苦苦追寻的目标的隐蔽是否有着其中本身的深刻内涵？

那么究竟非智能与智能之间的界限在哪里呢?

智能的基本能力:
1.对于情境有很灵活的反应
2.充分利用机遇
3.弄懂含糊不清或彼此矛盾的信息
4.认识到一个情境中什么是重要的因素，什么是次要的
5.在存在差异的情境之间能发现他们的相似处
6.从那些由相似之处联系在一起的事物中找出差别
7.用旧的概念综合出新的概念，把它们用新的方法组合起来
8.提出全新的概念

人工智能的奇异之处在于试图将一长串严格形式化的规则放在一起，用这些规则教给不灵活的机器如何变得灵活

形式系统与现实

对于那些基于现实的一部分的形式系统中有些看起来模仿的很好。在这种模仿中，它的定理与有关的那部分现实中的真理同构。

现实世界的一切都可以形式化吗？

比如说，事物由组成事物的基本粒子构成，在三维空间里面的运动假如都符合某种运动定律。那些物理法则，告诉人们如何根据给定的时刻从给出的所有粒子的位置和速度，得出属于“下一个瞬间”的一组新的位置和速度。

形式与非形式的对比

Thank you !!!!!!