Laurea Magistrale in Informatica
Dipartimento di Informatica
Università di **To**rino

Representation Learning

Course

Neural Networks and Deep Learning

Professor

Roberto Esposito
roberto.esposito@unito.it

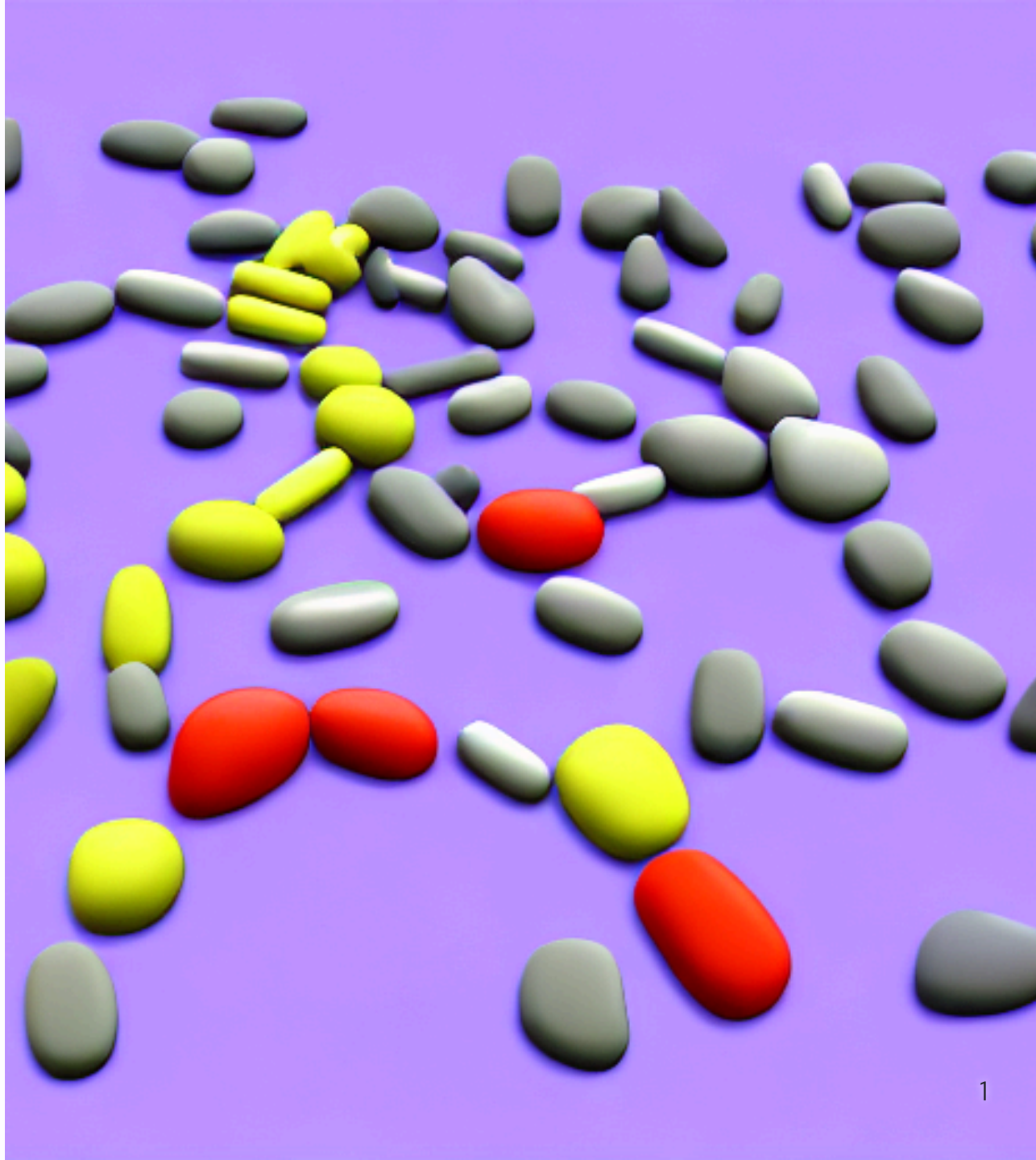


Image dreamed by stable diffusion
Prompt: "abstract image about neural network autencoders"

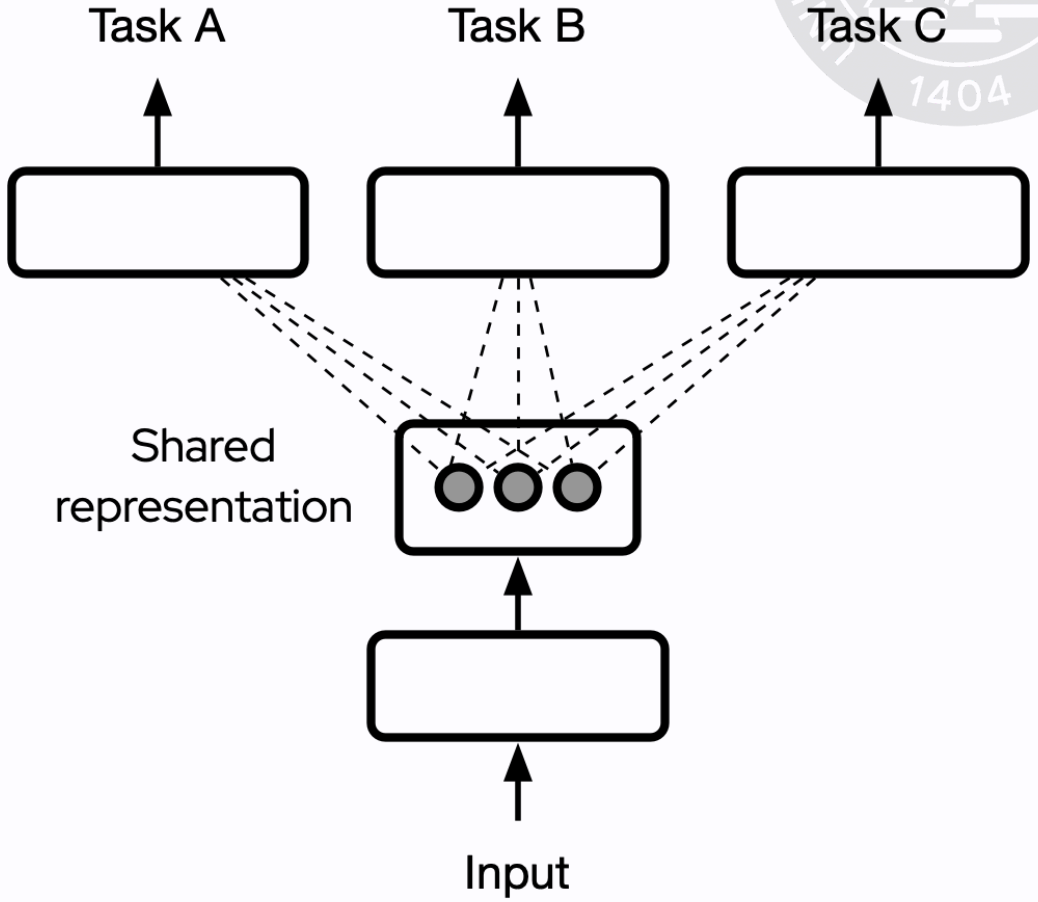
> *The concept of **representation learning** ties together all of the many forms of deep learning. Feedforward and recurrent networks, autoencoders and deep probabilistic models all learn and exploit representations. Learning the best possible representation remains an exciting avenue of research.*

Deep Learning, p. 557

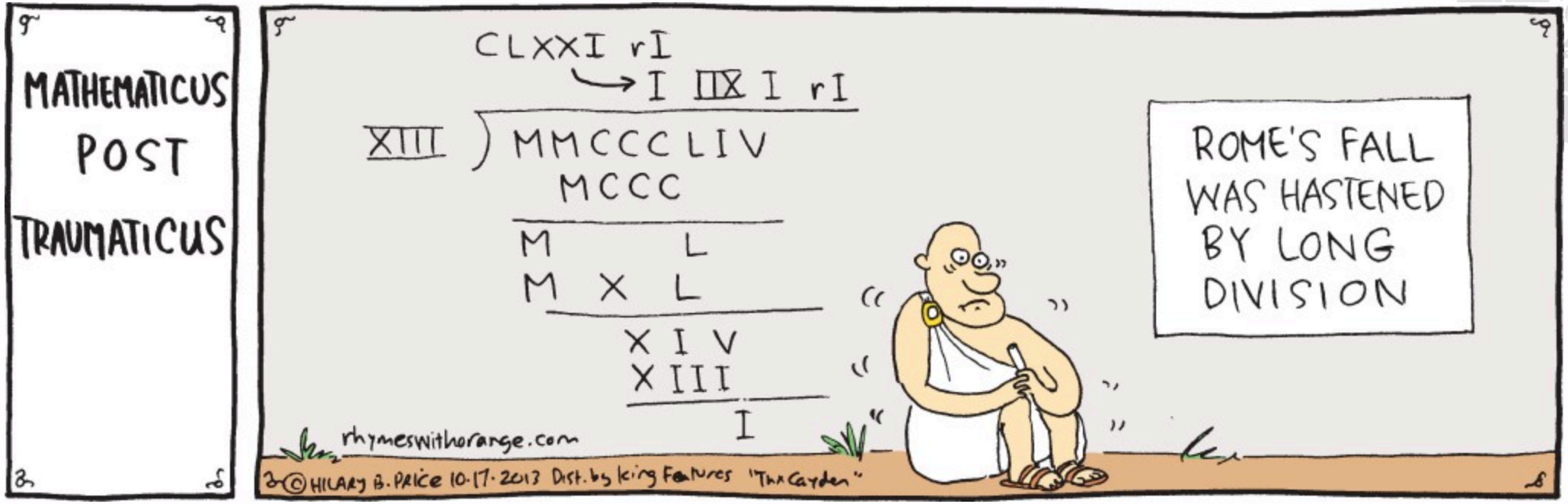
Representation Learning

Representation learning enables the sharing of statistical power across various tasks, enhancing the overall performance.

These shared representations are particularly beneficial when dealing with multiple modalities or domains. They also facilitate knowledge transfer to tasks where few or no examples are available, but a task representation exists.



Representations and Difficulty



Changing representation can make a problem very difficult or very easy:

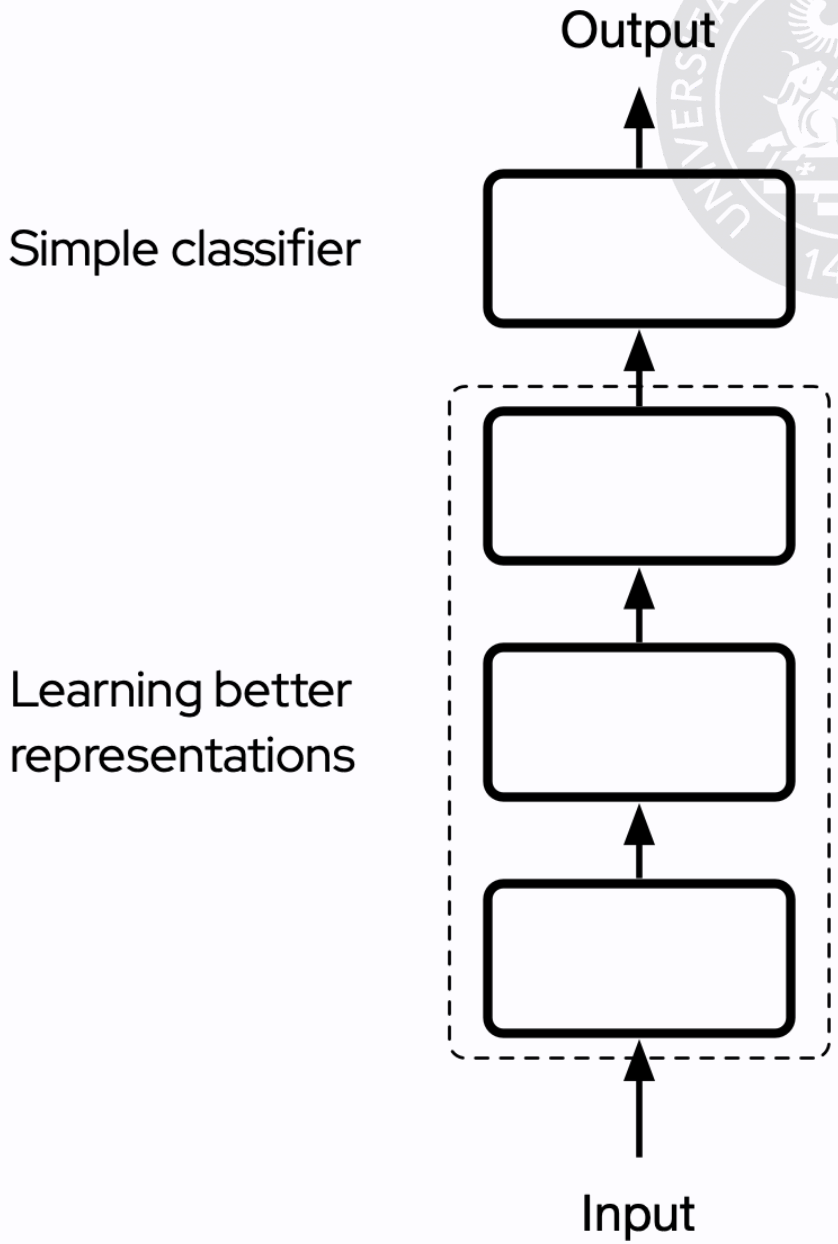
- Dividing two numbers can be easy when they are expressed with a convenient notation (e.g., positional notation) and difficult when they are not (e.g., roman numerals);
- Inserting a number into the correct position is an $O(n)$ operation if the underlying representation is an ordered list, it's $O(\log(n))$ if it is a red-black tree.

Representations and Difficulty

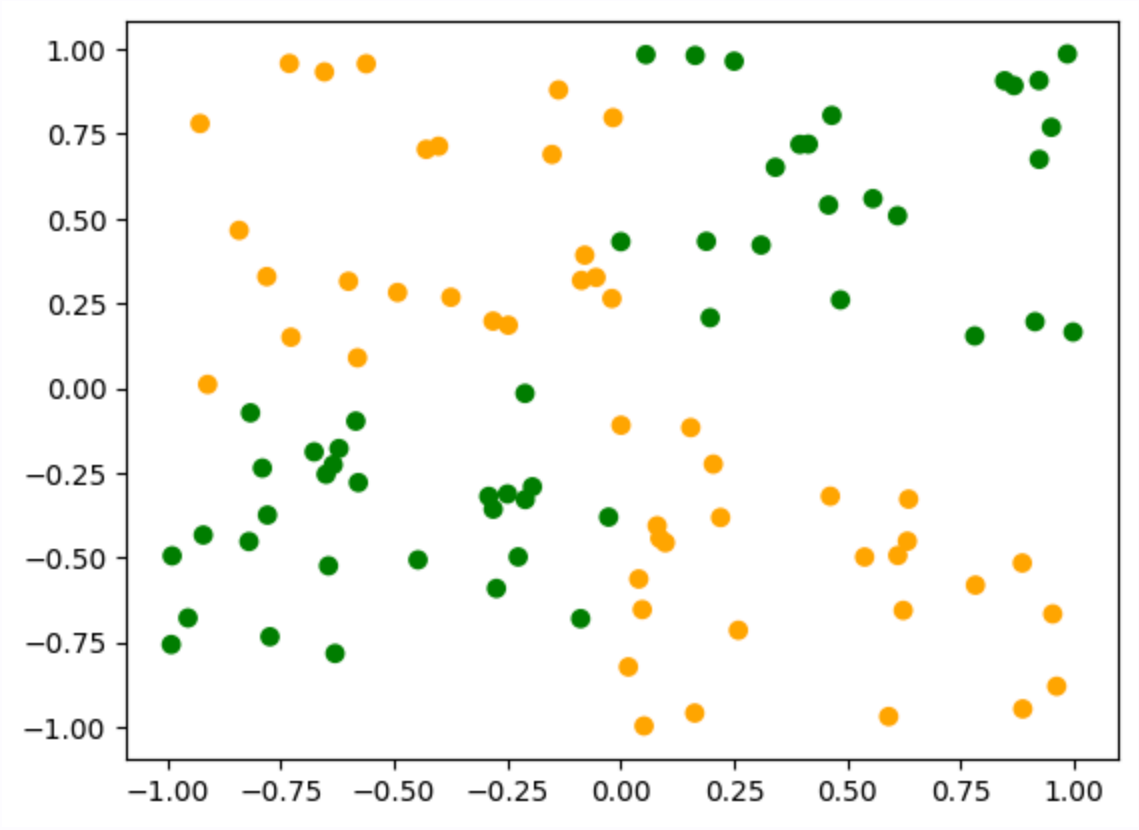
A **good representation** is one that makes a subsequent learning task easy.

Feedforward networks training by a supervised learning algorithm can be thought as performing some kind of representation learning.

The last few layers are typically a simple classifier that can be very accurate because the rest of the network provides it with a good representation.

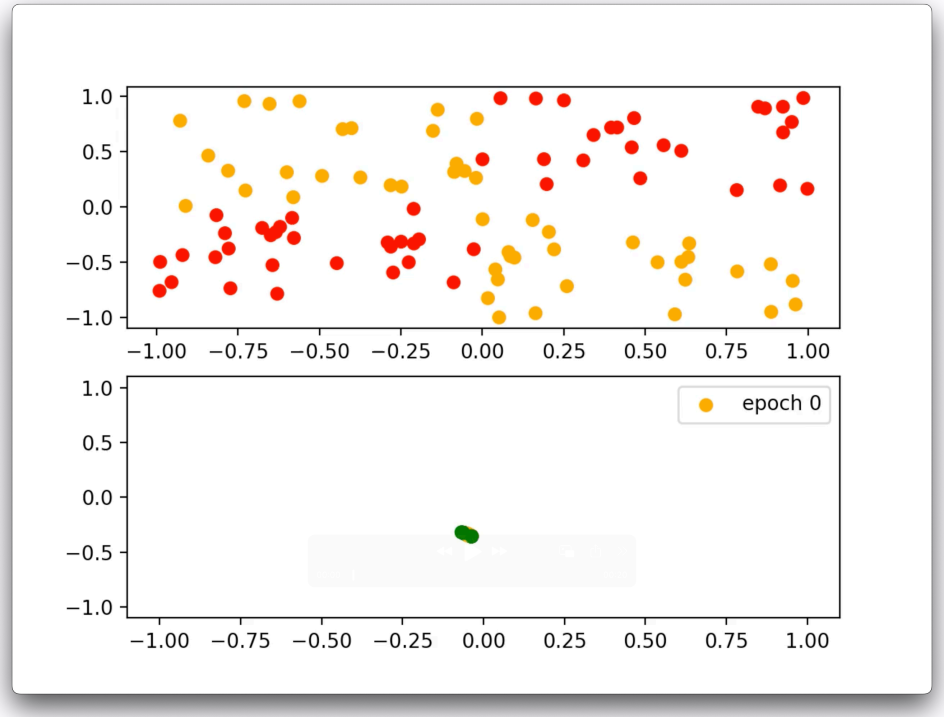


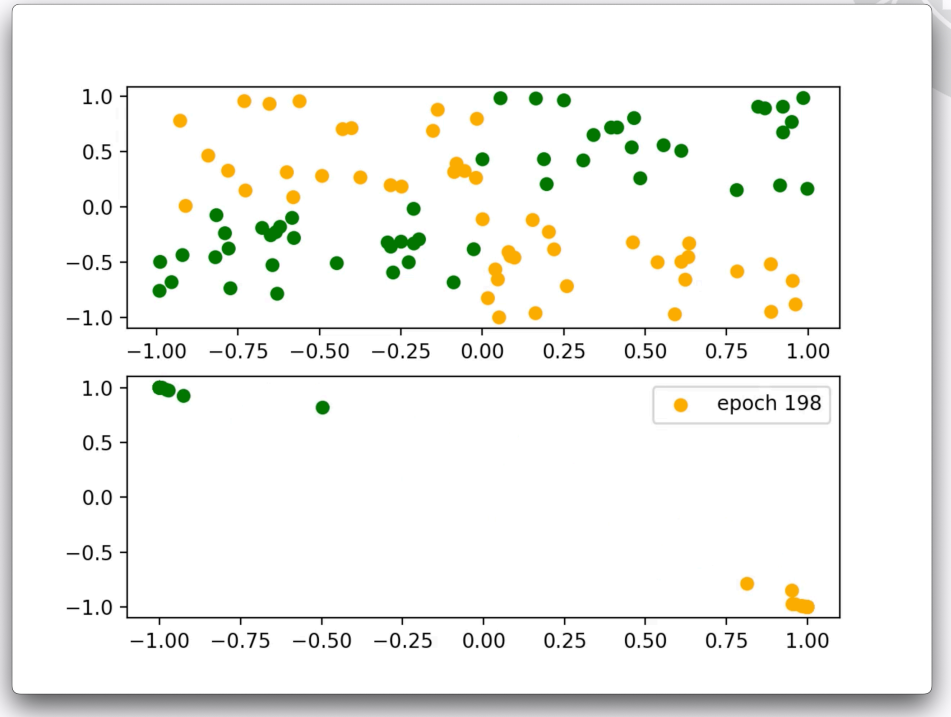
Example on a Toy XOR Dataset




```python
class NN(torch.nn.Module):
    def __init__(self) -> None:
        super().__init__()
        self.fc1 = torch.nn.Linear(2, 15)
        self.fc2 = torch.nn.Linear(15, 8)
        self.fc3 = torch.nn.Linear(8, 5)
        self.fc4 = torch.nn.Linear(5, 2)

    def encode(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        return x

    def repr(self, x):
        x = self.encode(x)
        x = F.htan(self.fc4(x))
        return x

    def forward(self, x):
        x = self.encode(x)
        x = self.fc4(x)
        x = F.softmax(x, dim=1)
        return x
```

Feed forward network learns representations





- Mislabeled examples
- Positive examples
- Negative examples

Animation

Multiple Objectives

It is worth considering that the learning objective can be tailored to force the representation to have some nice properties.

Most representation learning problems face a tradeoff between preserving as much information about the input as possible and attaining nice properties (such as *independence* of the features).

Greedy Layer-Wise Unsupervised Pretraining

Why the name?

- It is a **greedy** algorithm;
- it is **layer-wise**;
- layers are trained in an **unsupervised** way;
- supposed to be **only a first step before** joint **training** kicks in

Greedy Layer-Wise Unsupervised Pretraining

- Instrumental to the revival of deep neural networks.
- a clear example of how **learning a good representation for one task** (unsupervised learning, trying to capture the shape of the distribution) **can sometimes be useful for another task** (supervised learning with the same input domain).

Why is (was) it necessary?

Before modern training techniques came about, deep nets had trouble in propagating information.

Specifically, deep networks had problems with **vanishing** and **exploding** gradients.

Solution: break up the training into the training of smaller networks where such problems are less of an issue.

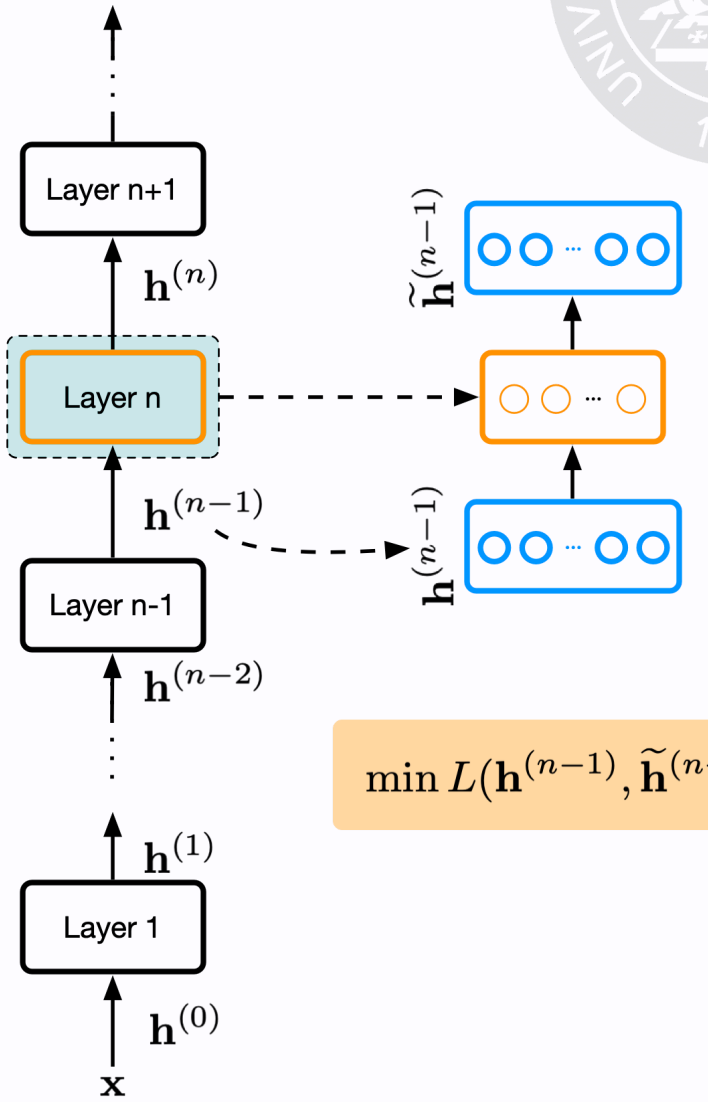
Greedy Layer-Wise Unsupervised Pretraining

Overview

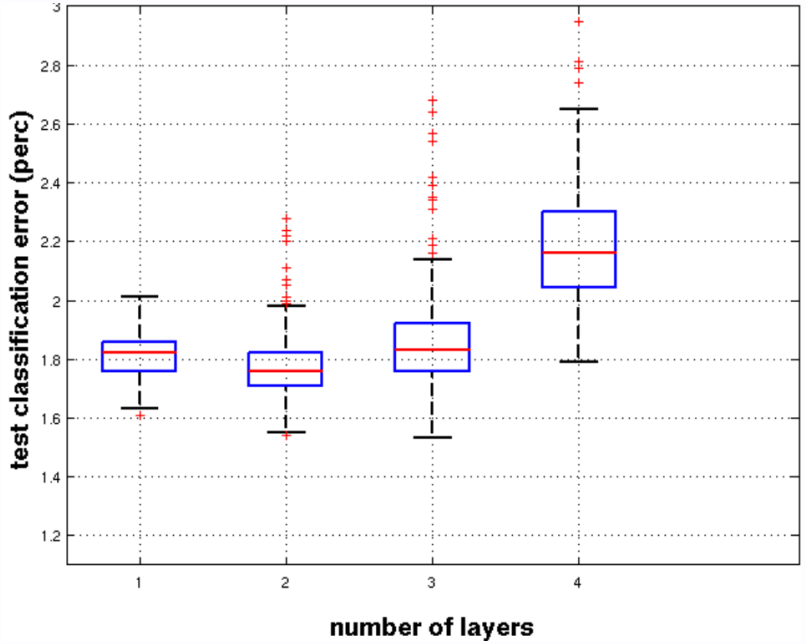
*Each layer is trained using **unsupervised** learning, taking the output of the previous layer and producing as output a new representation of the data, whose distribution is hopefully simpler/better.*

Most often, the procedure relies on a single-layer representation learning algorithm such as:

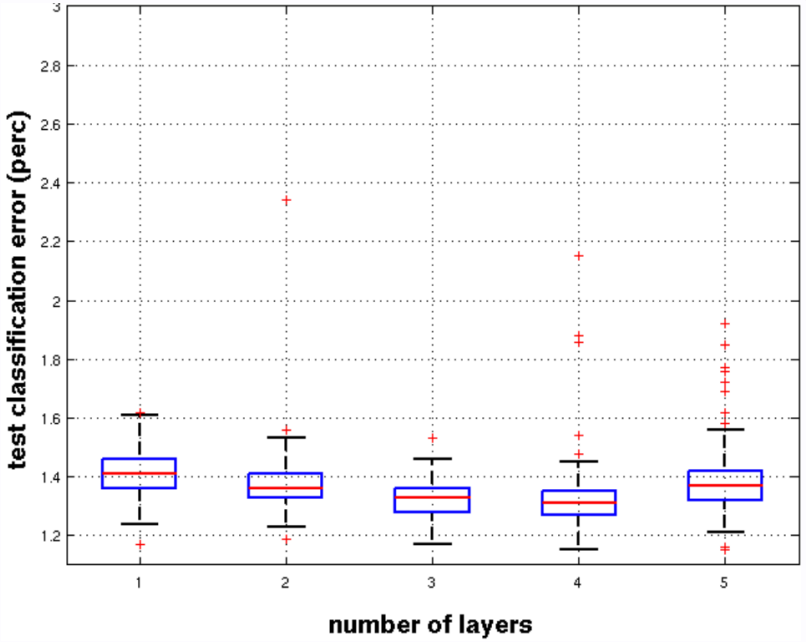
- Restricted Boltzman Machines;
- Single layer autoencoders;
- other models that learn latent representations.



Experiments on MNIST



No pretraining



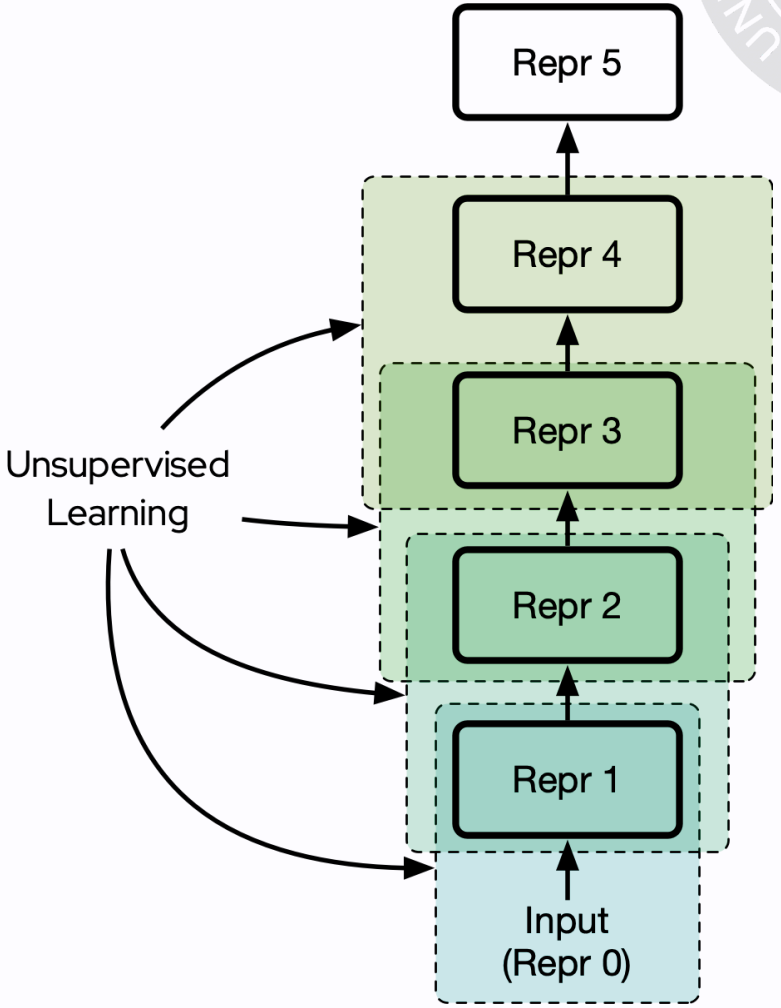
With pretraining

Note: the authors were unable to properly train a network with 5 layers without pretraining.

1: *Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent* , **Why does Unsupervised Pre-training Help Deep Learning?.** 2009.

Greedy Layer-Wise Unsupervised Pretraining

```
def greedy_layer_wise_unsupervised_pretraining(X,y)

    f = lambda z: z          # identity function
    data = X                 # repr 0

    for k in range(m):
        f_k = UnsupervisedLearn(data)
        f = lambda x: f_k(f(x))
        data = f_k(data)     # repr k+1

    if fine_tuning:
        f = FineTune(f, X, y)

    return f
```



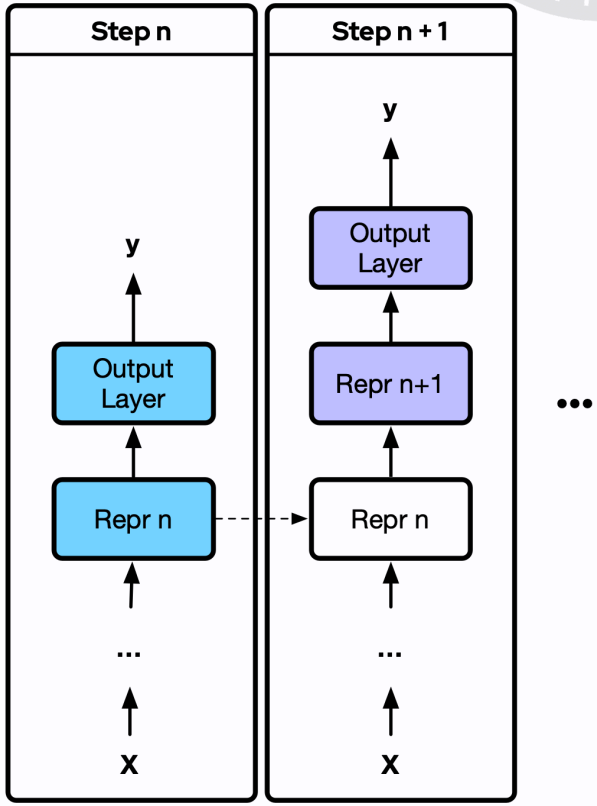
Greedy Layer-Wise Unsupervised Pretraining

This procedure played a major role in 2006 when it was shown that it could be used to find good initialization points for training deep architectures.

Today, we understand training deep architectures is possible and that it does not necessarily depend on greedy pretraining. However, it's crucial to acknowledge that the **this approach was the initial demonstration that such training was indeed achievable**.

Variants

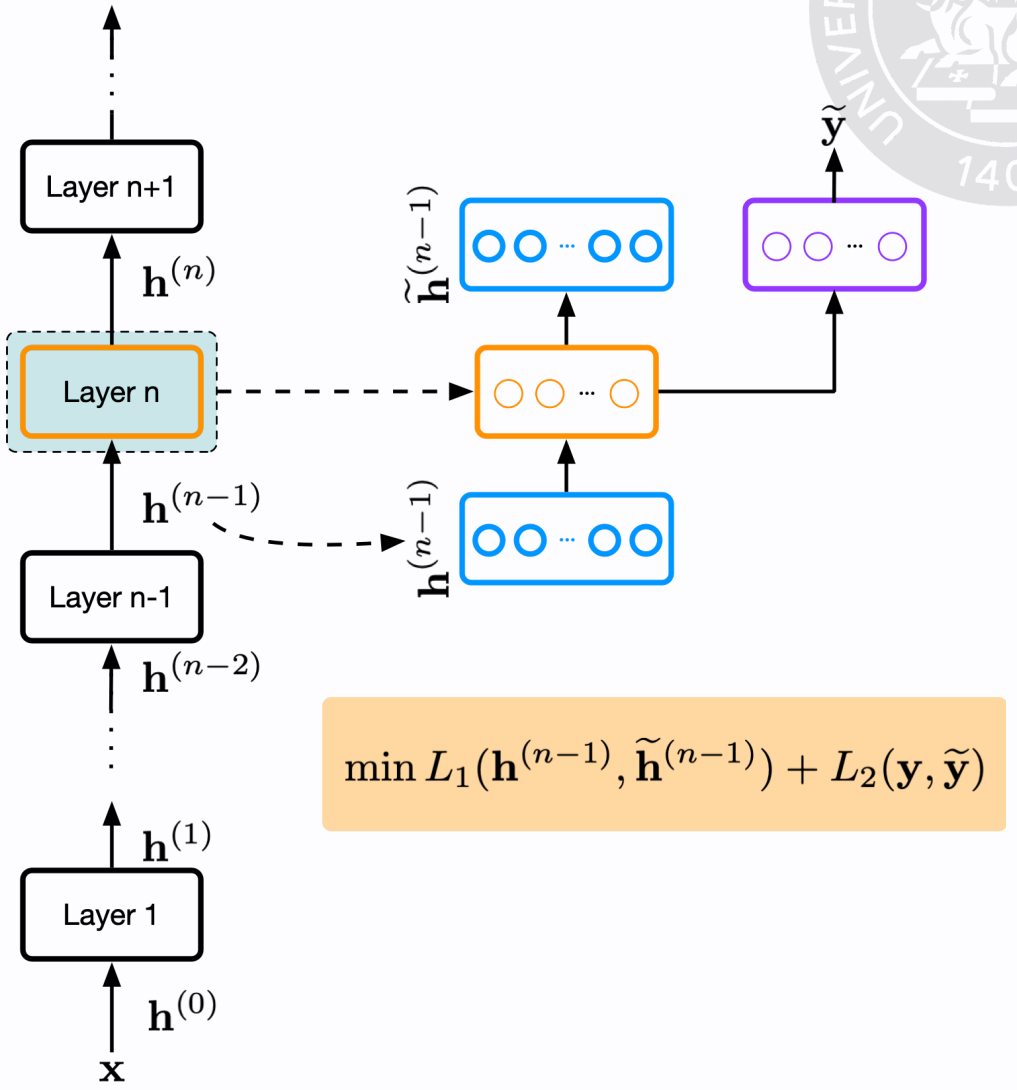
- The algorithm can be also used as an initialization for deep unsupervised models.
- It is also possible to have greedy layer-wise supervised pretraining.



- Simultaneous supervised and unsupervised learning

Rationale

the integration allows incorporating the constraints imposed by the output layer from the outset.



Why Unsupervised Pretraining Work?

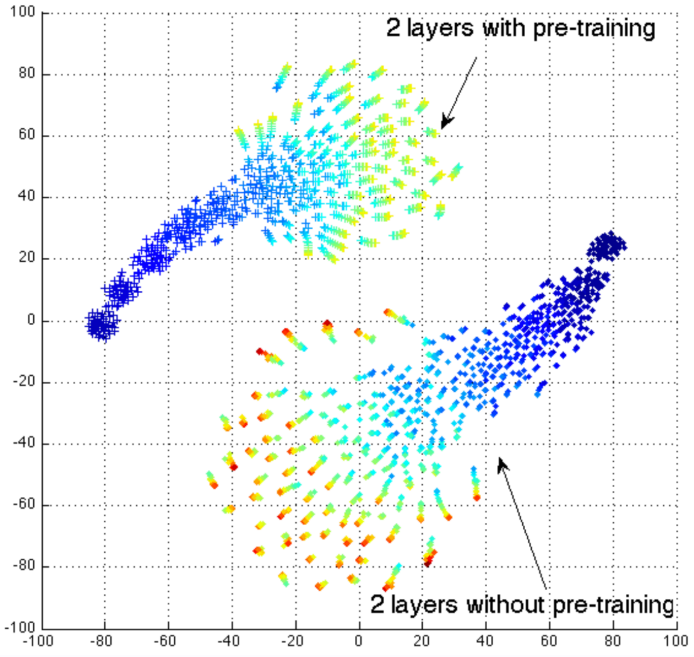
Two main ideas are important to understand why and when unsupervised pretraining work:

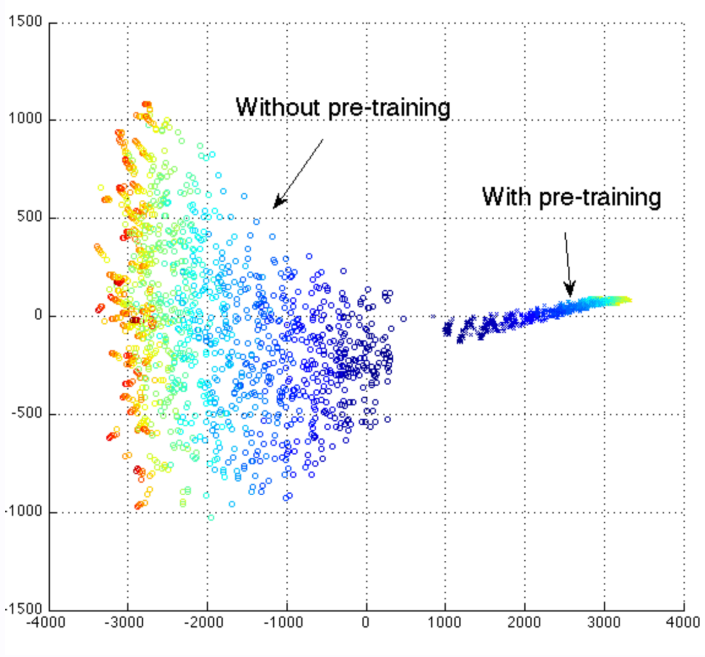
1. the **choice of initial parameters** can have a significant regularizing effect on the model;
2. **learning about the input distribution** can help to learn the mapping from inputs to outputs.

1. Choice of Initial Parameters

It was initially assumed that pretraining would cause the training procedure to approach one local minimum instead of another one.

Plots of two different projections (tSNE and ISOMAP) of the functions representing 50 networks trained with and without pretraining. Colors (from blue to red) indicate a progression in training iterations.

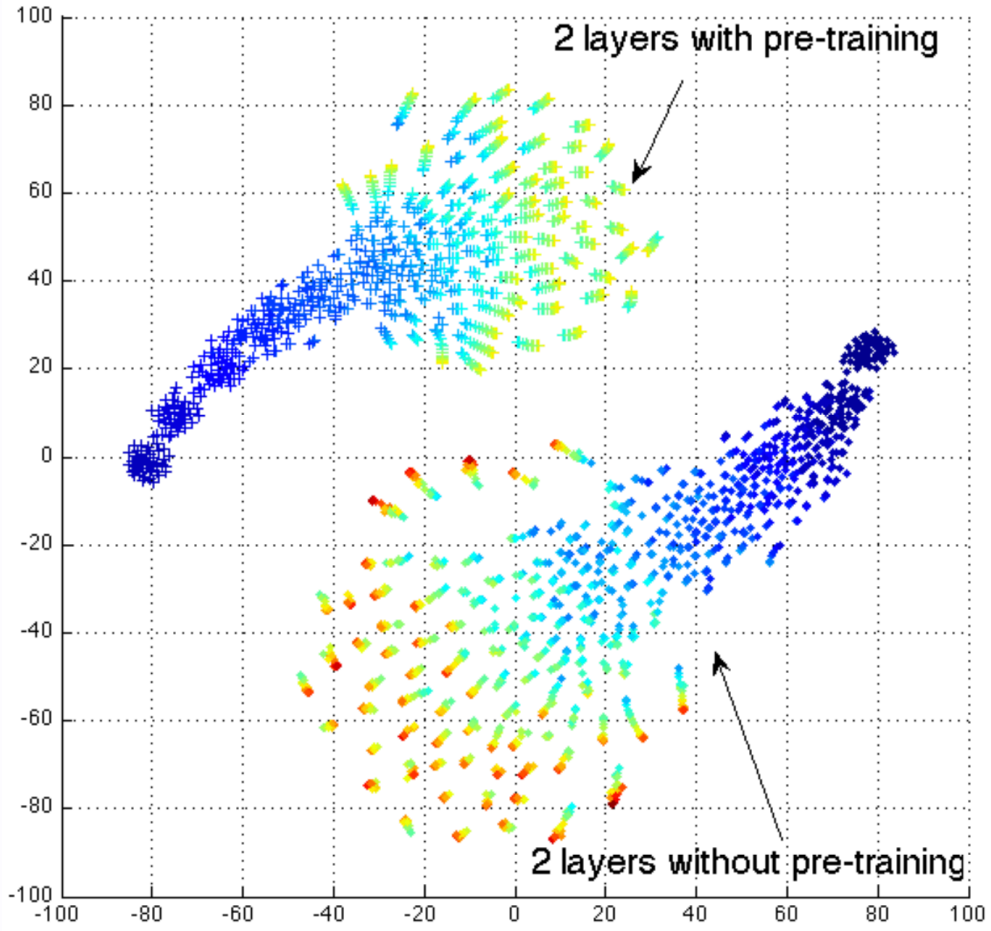




1: *Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent* , **Why does Unsupervised Pre-training Help Deep Learning?.** 2009.

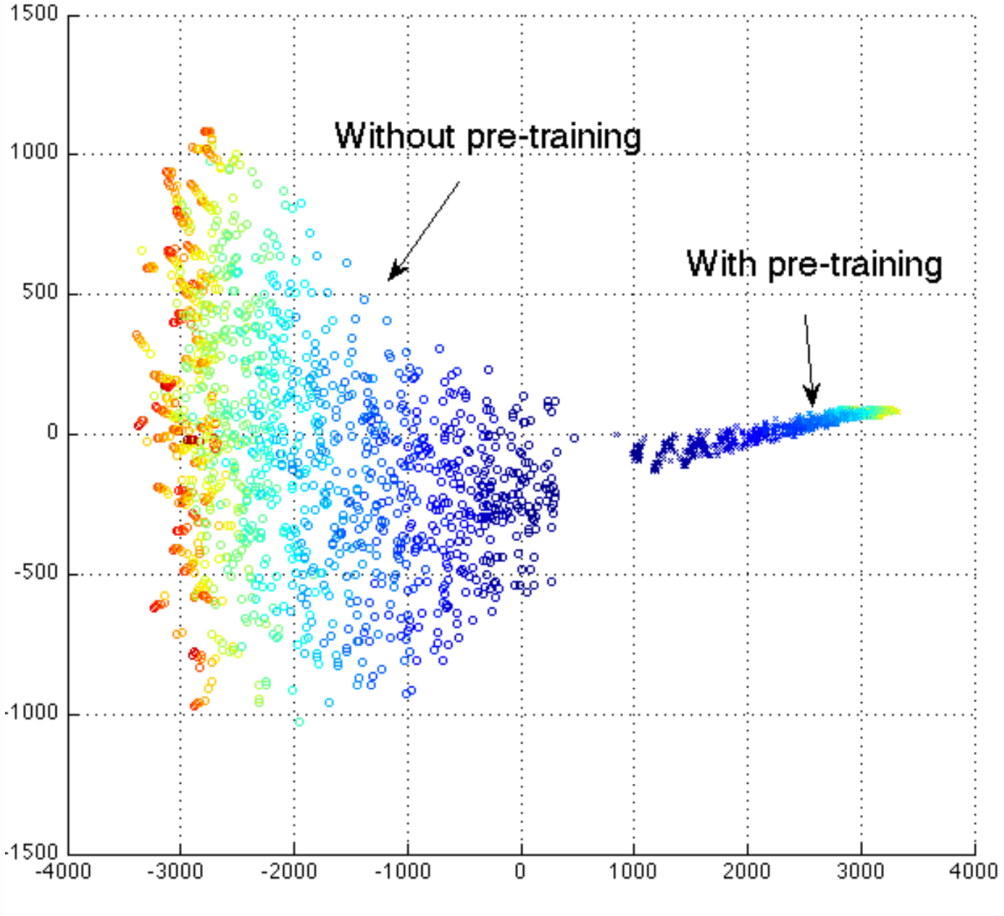
Observations

1. Training is longer without pretraining.
2. The pre-trained and not pre-trained models start and stay in different regions of function space.
3. All trajectories of a given type initially move together and then diverge. This suggests that each trajectory moves into a different apparent local minimum.



Observations

1. The pre-trained models live in a disjoint and much smaller region of space than the not pre-trained models.
2. The pre-trained solutions look all the same, and their self-similarity increases during training, while the opposite is observed without pre-training.



Now local minima are no longer considered a serious problem:

> *... Recent work on understanding the quality of training argues that critical points are more **likely to be saddle points** rather than spurious local minima [...] and that **local minima concentrate near the global optimum** ...*

Mathematics of deep learning [1]

... more likely to be saddle points ...

The idea is that as the number of dimensions (parameters) increases, since **for each dimension there is a direction of ascent and a direction of descent**, the probability of finding a minimum (where all directions are ascent) decreases.

1: *Vidal, Rene and Bruna, Joan and Giryes, Raja and Soatto, Stefano,* **Mathematics of deep learning.** 2017.

2. Learning the Input Distribution

It is widely understood that in many tasks features learned in the unsupervised stage can be useful also in the supervised stage.

Example: *Unsupervised learning on images of cars and motorcycles.*

If we are fortunate the unsupervised task can be learning features such as the shape of the wheels that can be very useful for the supervised task.

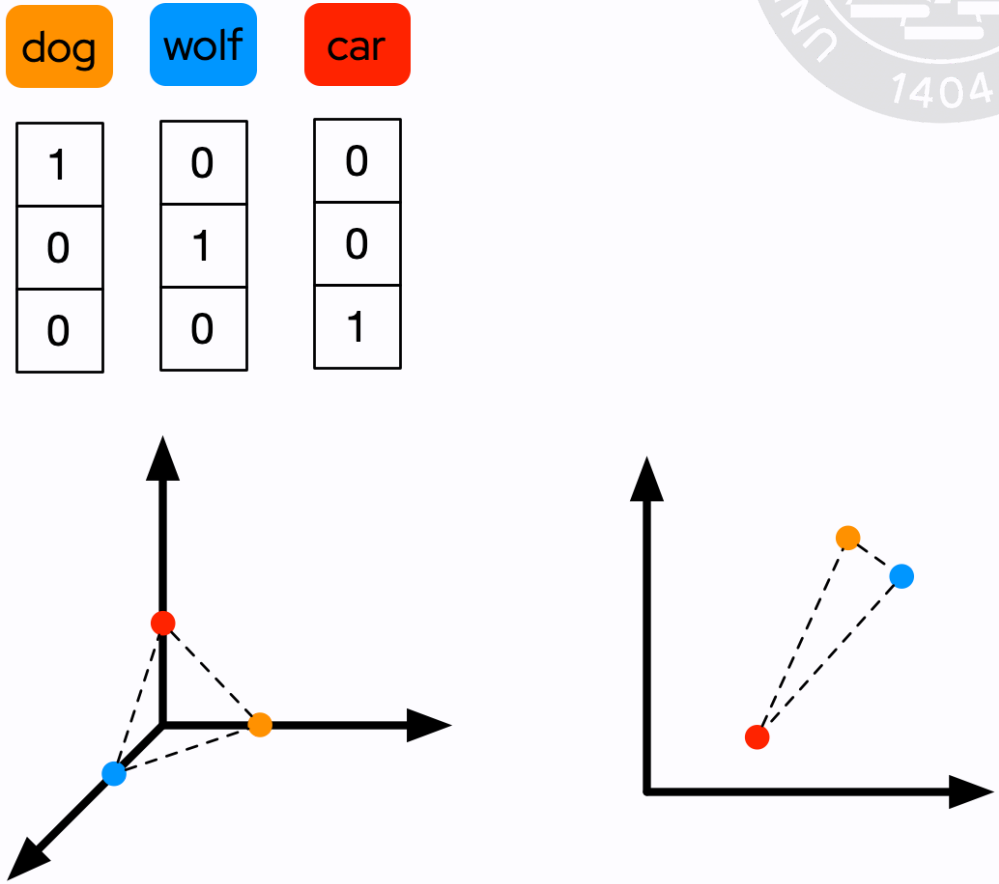
The details of the network may have a play here (e.g., using linear output units may force the representations of the examples to be linearly separable).

When does it help?

Unsupervised pretraining often helps **when the original representation is poor**.

Example: *learning word embeddings*

Words represented by one-hot vectors are particularly poor: any two different vectors are at the same distance from each other.



When does it help?

Viewing unsupervised pretraining **as a regularizer**, it is most useful **when the number of examples is small**.

For the same reasons, it is also most useful **when the number of *unlabeled* examples is large**.

When does it help?

Unsupervised pretraining is also likely to be useful **when the function to be learned is extremely complicated**.

In contrast with other regularizers, unsupervised pretraining **does not force the learnt function to be simple**, rather it helps in discovering feature functions that are helpful in the unsupervised task.

If the true underlying functions are *complicated* **and** *shaped by regularities* of the input distribution, unsupervised pretraining can be a more appropriate regularizer.

Disadvantages

As a regularization technique, unsupervised pretraining has the problem that it is **difficult to calibrate**.

In most regularization techniques there is a single parameter allowing to set the strength of the regularization.

With unsupervised training, either the network is initialized using pretraining or it is not.

Also, the hyper-parameters of the pretraining phase needs to be adjusted and this can be extremely slow.

Today usage

Today, unsupervised pretraining has been **largely abandoned**.

Modern alternatives

Deep learning techniques based on supervised learning, regularized with dropout or batch normalization usually outperform unsupervised pretraining.

In many fields (NLP in particular), a form of unsupervised pretraining called ***self-supervised learning*** *is used to learn good representations.*

Representation Learning Applications

Transfer Learning and Domain Adaptation

Transfer Learning and Domain adaptation refer to the situation where what has been learned in one setting (distribution P_1) is exploited to improve generalization in another setting (distribution P_2).

This **generalizes the idea of greedy pretraining** where the transferred representations were between unsupervised and a supervised tasks.

Transfer Learning

In **Transfer Learning** the learner must perform two or more different tasks, but we assume that many of the factors that explain variations in P_1 are relevant to the variations that need to be captured for learning P_2.

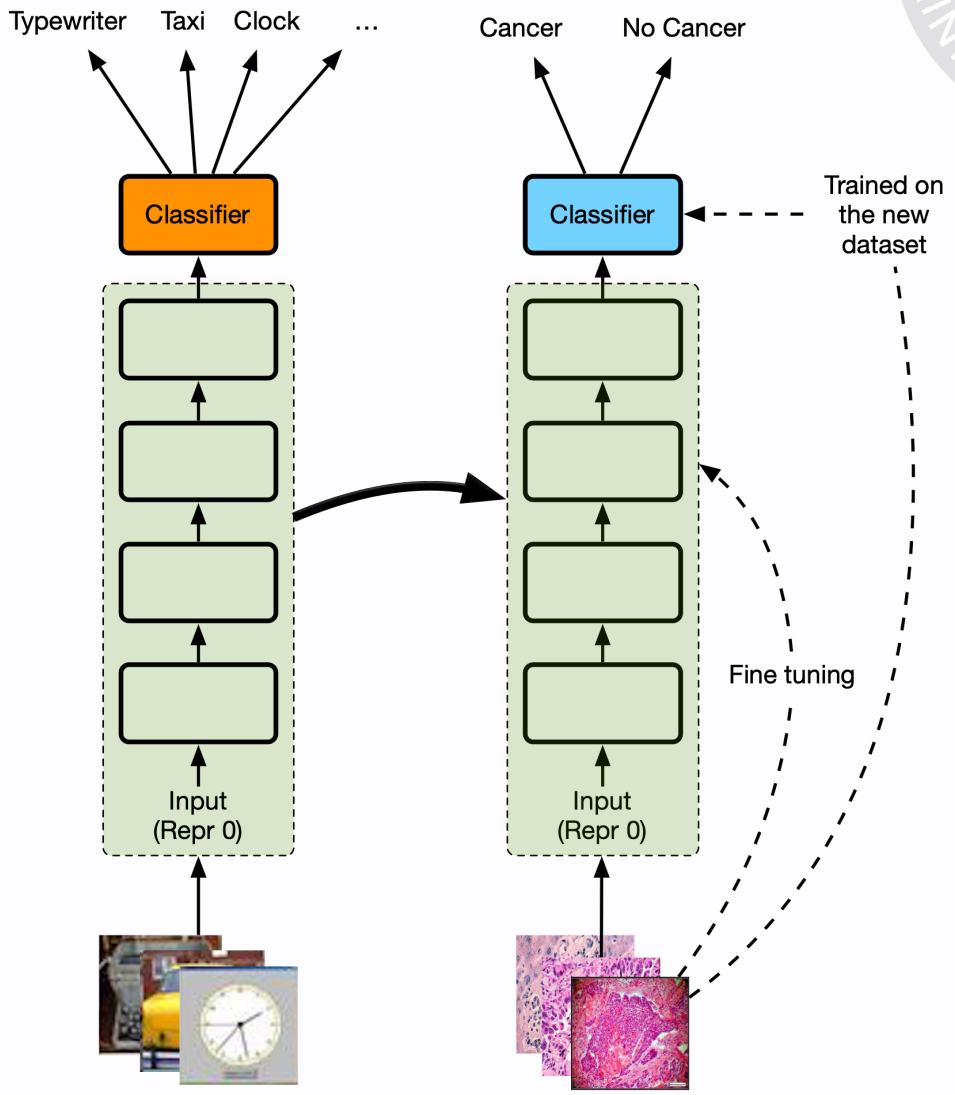
Example: *learning to recognize objects in images of (P_1) cats and dogs and then using the same network to recognize objects in images of (P_2) ants and wasps.*

One of the most striking results found is that **depth** seems to be crucial in this process:

> ***Note***
>
> ***As an architecture makes use of deeper and deeper representations****, the learning curve on the new categories of the second (transfer) setting* P_2 *becomes much better.*

Transfer Learning Example

Task: Differentiate between images that depict cancerous cells and those that depict benign cells.



PyTorch implementation

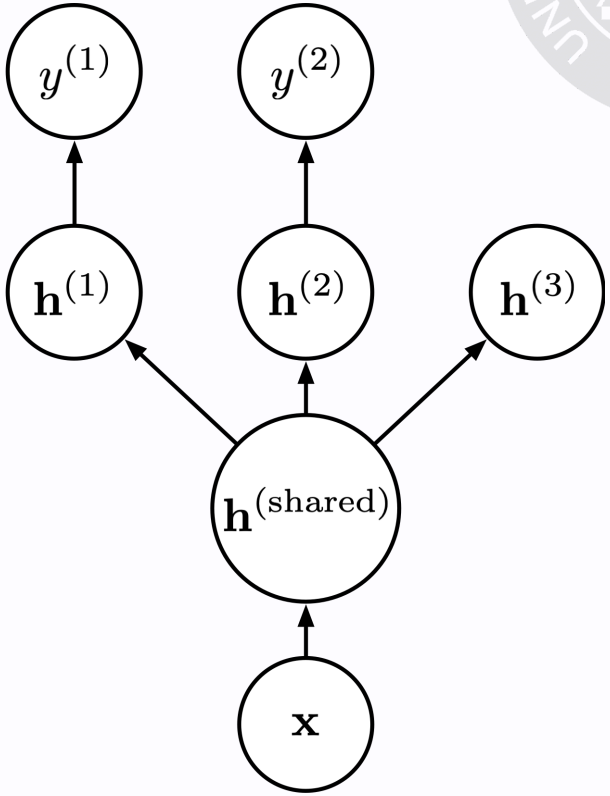
Transfer learning is relatively straightforward to implement in PyTorch. The library offers easy APIs to **retrieve the weights** in a given model, **to set the weights** in a second model, and to freeze them so that they will not be updated in the future.

```
model2_dict = model2.state_dict()

# creating a new dictionary containing only the weights
# we want to copy from model1
transf_weights = {
    k:v for k,v in model1.state_dict().items()
    if k in model1_repr_layers_keys
}

# updating model2's state dict with the pretrained weights
model2_dict.update(transf_weights)

# loading the updated state dict into model2
model2.load_state_dict(model2_dict)

# freezing the weights of the layers we copied from model1
model2.layer1.requires_grad_(False)
model2.layer2.requires_grad_(False)
...

# Or, instead of freezing them, we can set a different
# learning rate for the layers we copied
optimizer = torch.optim.Adam([
    {'params': model2.layer1.parameters(), 'lr': 1e-4},
    {'params': model2.layer2.parameters(), 'lr': 1e-4},
    ...
])
```

Transfer Learning

In many cases, transfer learning and domain adaptation assume there is a common (shared) **representation that explain the variations in the input data**. These variation are then adapted to the various tasks by different output layers.

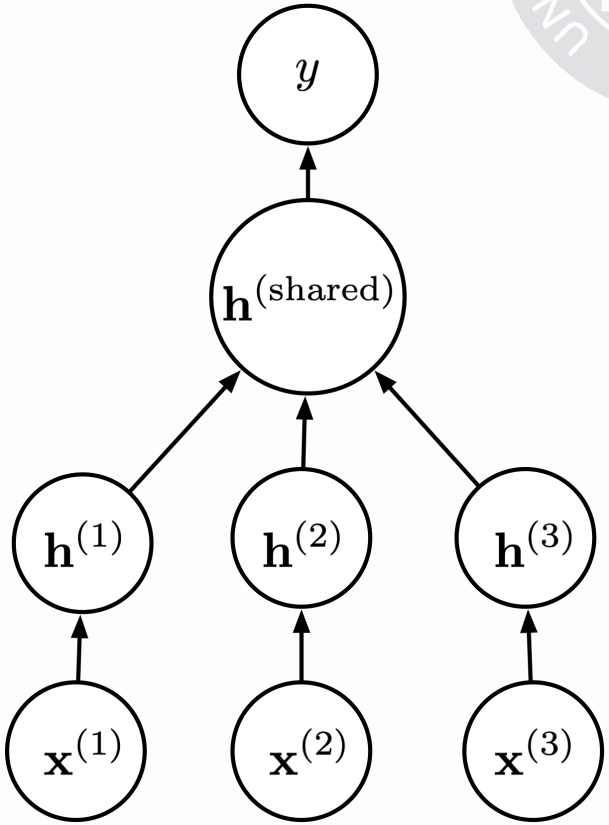
Example: Image recognition.



Transfer Learning

There are however situations where the opposite is true. There are situations, where what is shared between different tasks not the semantics of the input but, rather, the semantics of the output and the inputs need to be adapted to be compatible with that.

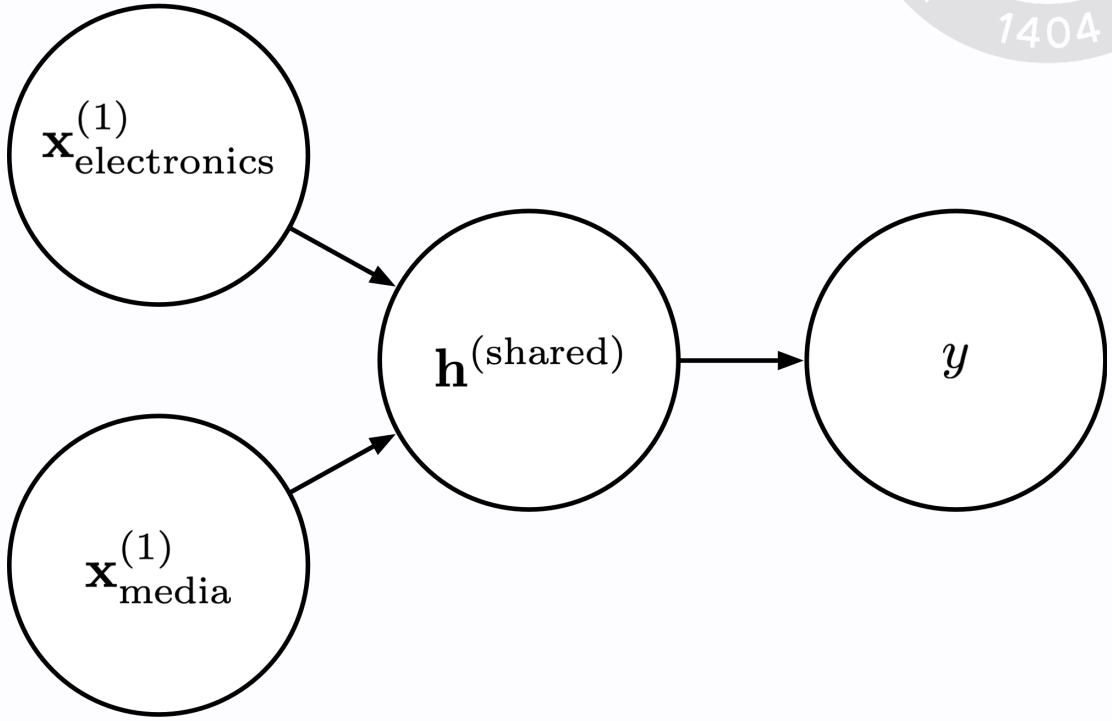
Example: Speech recognition.



Domain Adaptation

In domain adaptation, the task remains the same between each setting, but the input distribution is slightly different.

Example: Sentiment analysis



Domain Adaptation Example

Task: create a representation that captures the underlying patterns in the data while *ignoring domain-specific nuances*.

Example of source and target images:

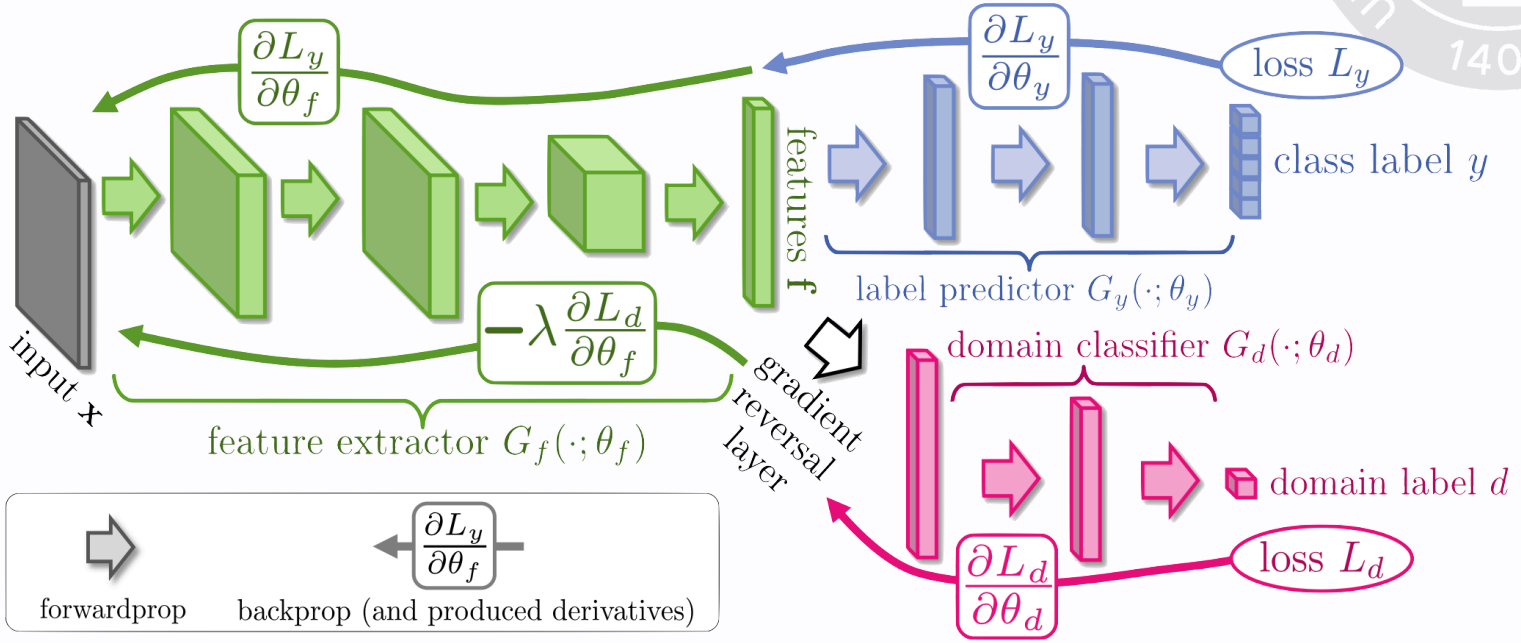


Image from Ganin and Lempitsky, 2015[1]

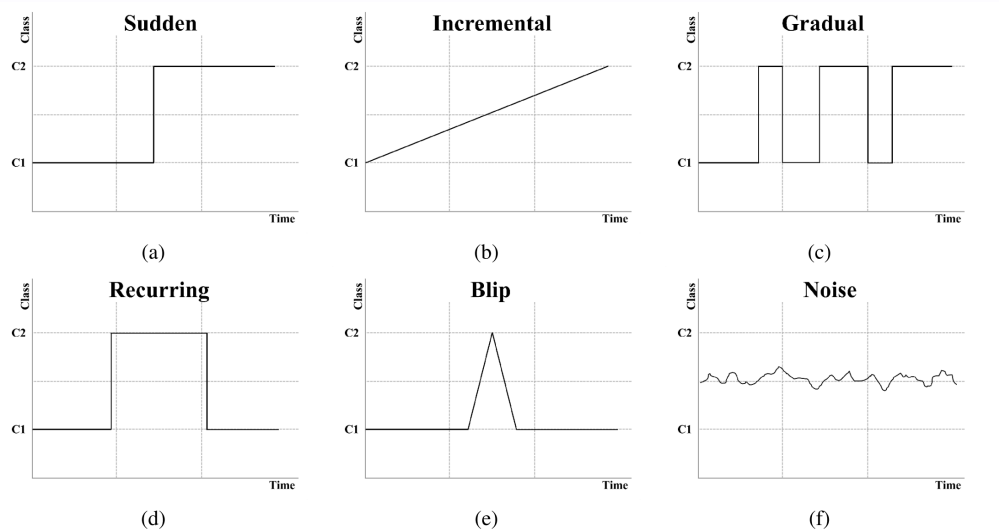
1: **Unsupervised Domain Adaptation by Backpropagation.** *Yaroslav Ganin and Victor Lempitsky,* ICML, 2015.

Concept Drift

Concept Drift occurs when the concept underlying the distribution of the data shifts in time. A model learnt at a given point in time need, then, to be updated to take into account the drift.

As in previous cases, the idea is to exploit data from a given setting (the distribution before the drift) to get an advantage in another setting (the distribution after the drift).

Concept drift is a large and very studied problem in ML. See *An Overview on Concept Drift Learning*[1] for an introduction to the problem.



1: *Adriana Sayuri Iwashita and João Paulo Papa,* **An Overview on Concept Drift Learning.** 2018.

One-Shot Learning

An extreme case of transfer learning is *one-shot learning* where only a single labeled example is given for the new setting (usually a set of new labels).

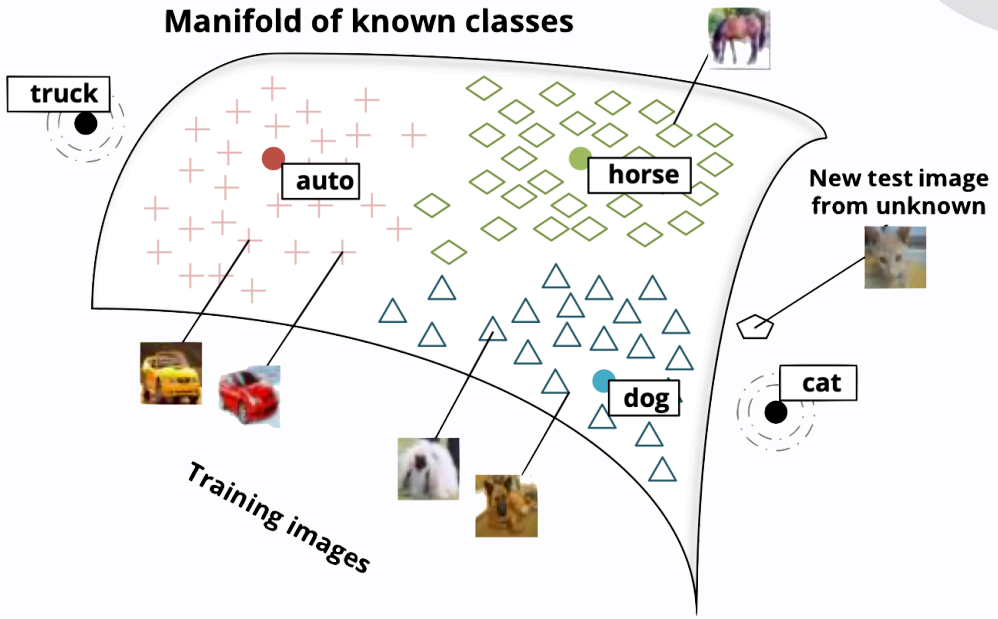


Image from Socher et al. (2013)[1]

1: **Zero-shot learning through cross-modal transfer.** *Socher, Richard, et al.*, 2013.

Zero-Shot Learning

In *zero-shot learning* (a.k.a., *zero-data learning*) one needs to adapt to a new setting without seeing *any* labelled example.

One can think to zero-data learning as including three random variables: the traditional input $\mathbf{x}$, the traditional outputs $\mathbf{y}$, and an additional random variable describing the task, T. The model is trained to model $p(\mathbf{y}|\mathbf{x}, T)$.

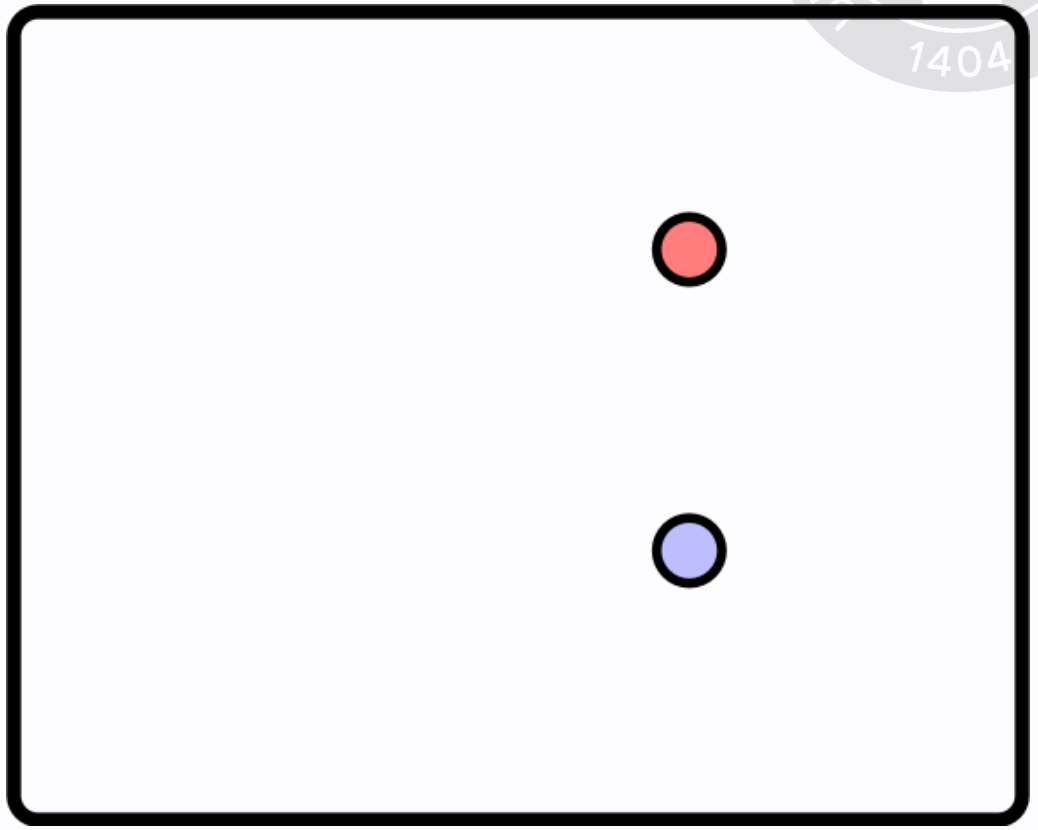
Example: Image recognition where T is a description of the object being recognized (e.g., $y = 1$ if there is a cat in the scene and T describes cats as having four legs and pointed ears).

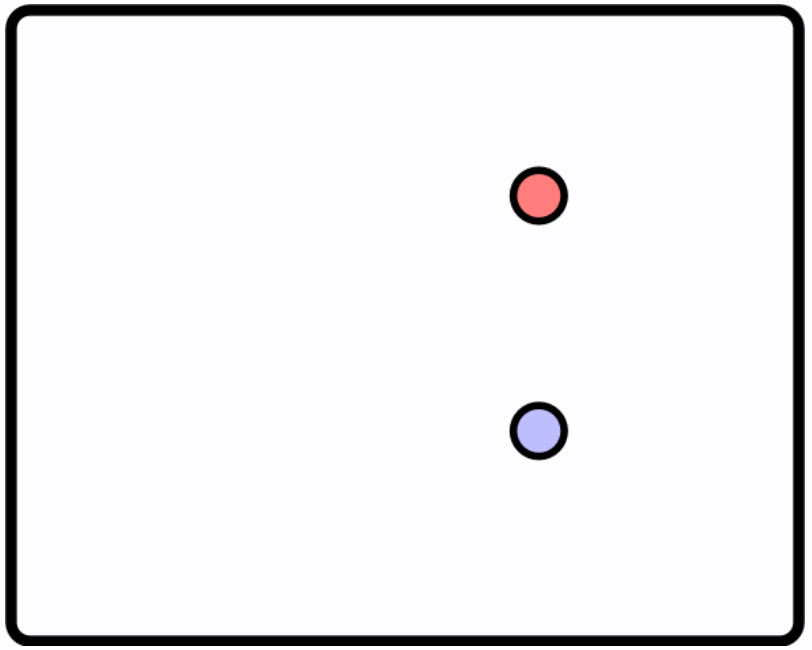
Example 2: ChatGPT answering to the user based on a description T of the desired output.

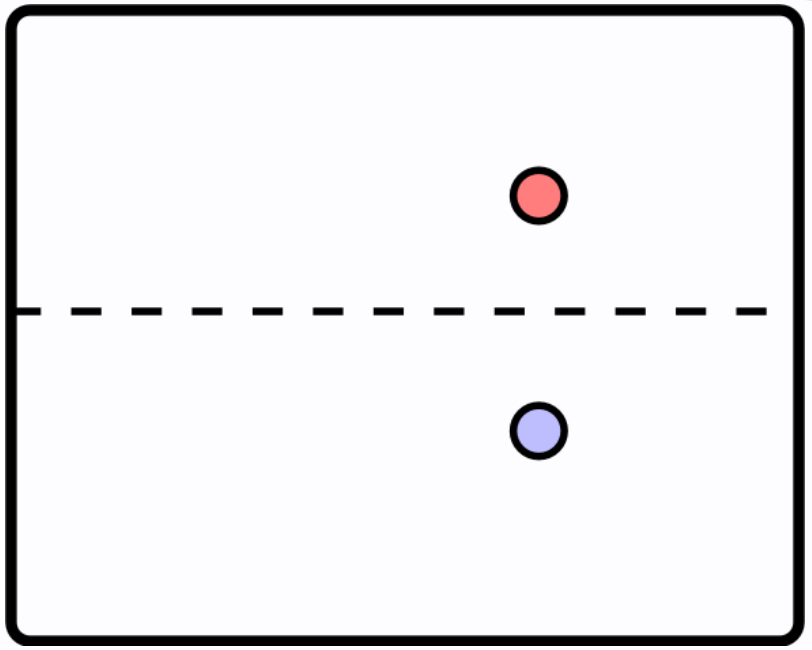
Semi Supervised Learning and Causality

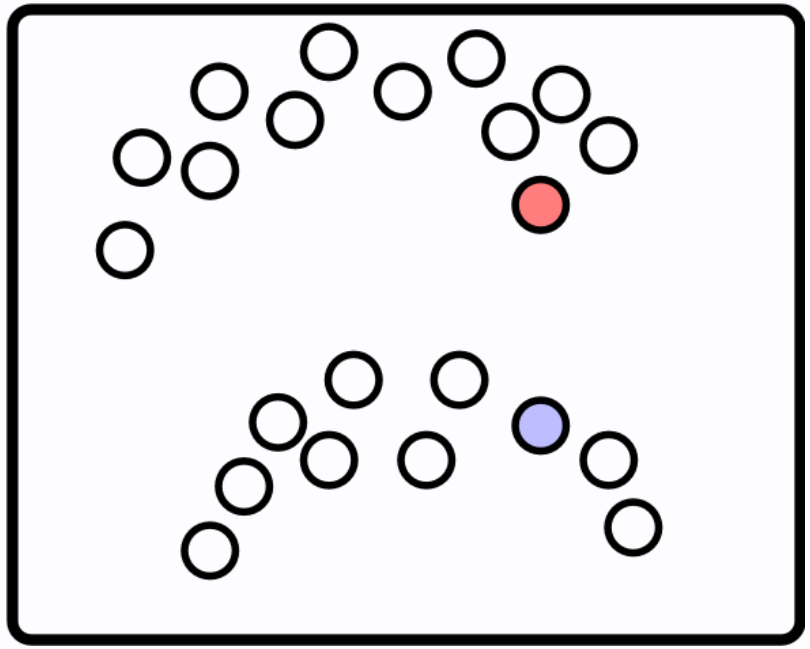
Semi-supervised learning (SSL)

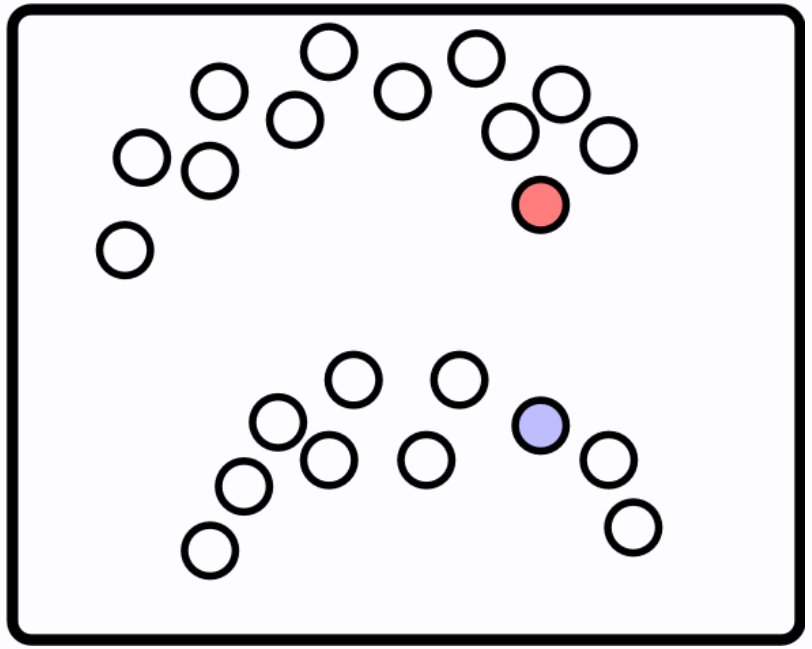
Semi-supervised learning is a setting where we are given a possibly large amount of examples, but only few of them are labelled. The task is to **exploit the unlabeled examples to better perform on the supervised task**.

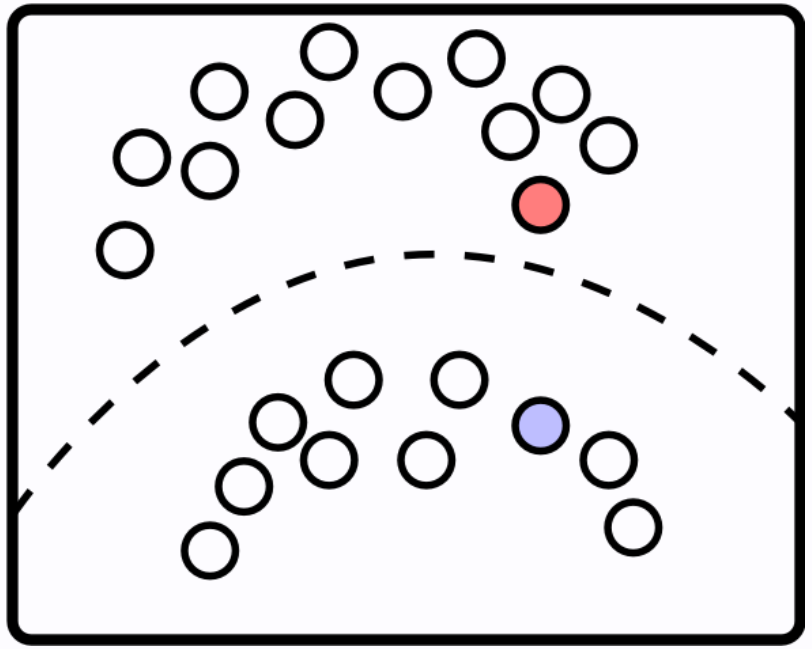












What makes one representation better than another?

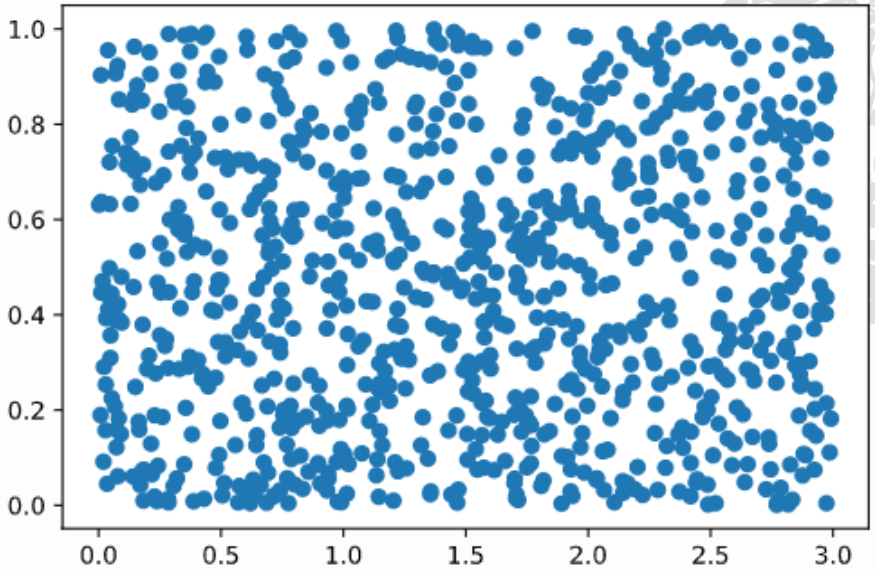
Hypothesis: *An **ideal representation** is one in which the **features** within the representation correspond to the **underlying causes** of the observed data.*

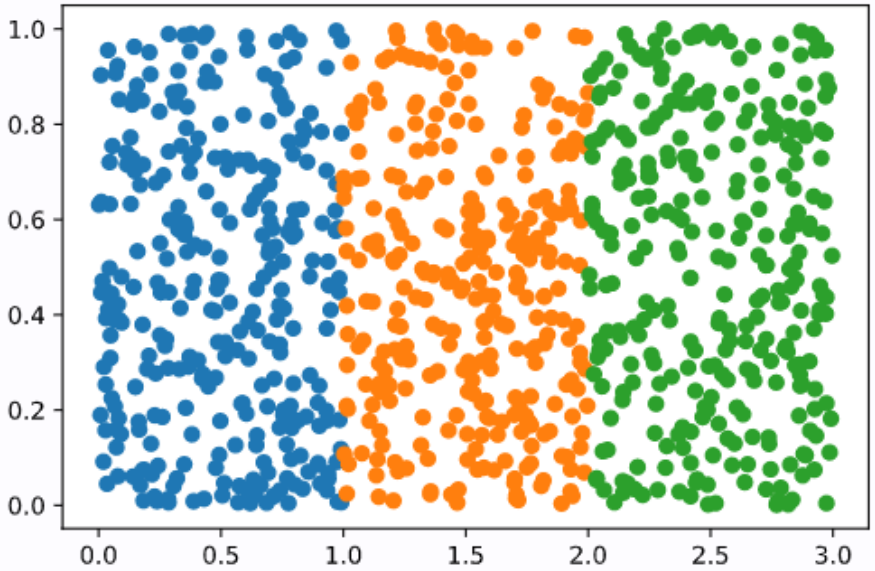
SSL and Causality

The "causality" hypothesis underlies a large deal of research motivated by the idea that disentangling the causal factors in $p(\mathbf{x})$ could be a good step for learning $p(\mathbf{y}|\mathbf{x})$ and **motivates the SSL approach**.

SSL and Causality: a failure

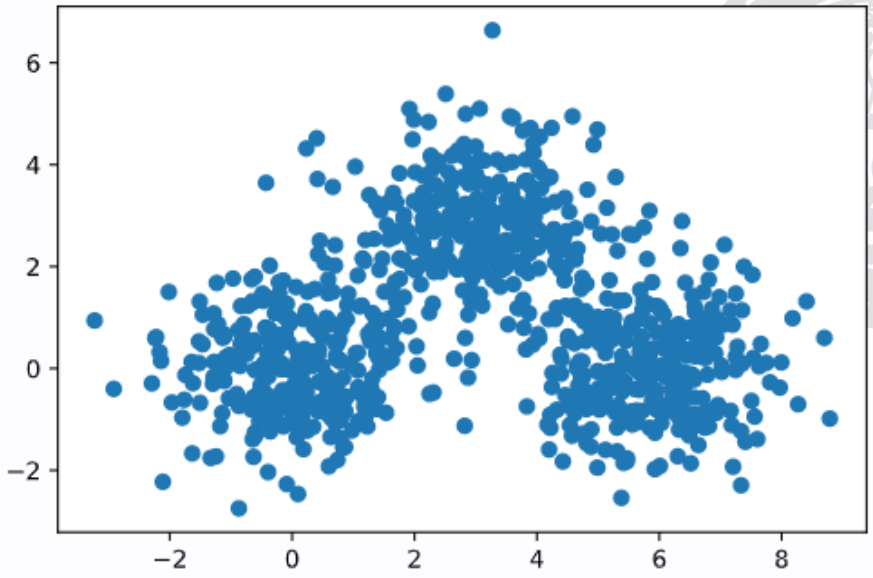
Sometimes semi-supervised learning fails because unsupervised learning of $p(\mathbf{x})$ is of no help to learn $p(\mathbf{y}|\mathbf{x})$.

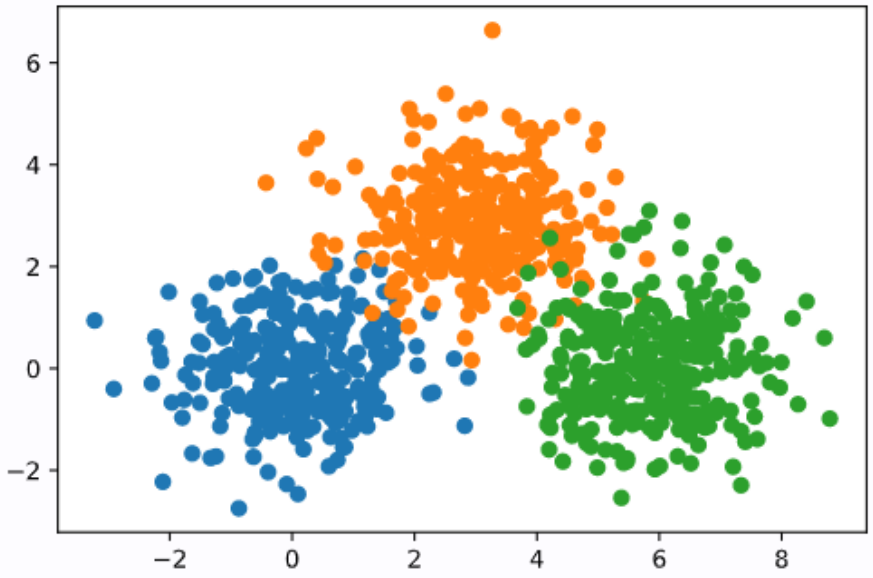




SSL and Causality: a success

Other times, $\mathbf{y}$ is among the salient causes of $p(\mathbf{x})$. In these cases learning $p(\mathbf{x})$ can be very helpful.





SSL and Causality

If $\mathbf{h}$ represents all factors causing $\mathbf{x}$ and we assume that $\mathbf{y}$ is related to one of them, then, the generative process can be conceived as:

$$p(\mathbf{h}, \mathbf{x}) = p(\mathbf{x}|\mathbf{h})p(\mathbf{h})$$

and the data has marginal probability:

$$p(\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{h}, \mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{x}|\mathbf{h})p(\mathbf{h}) = \mathbb{E}_{\mathbf{h}}[p(\mathbf{x}|\mathbf{h})]$$

the best possible model of $\mathbf{x}$ is the one that uncovers the above "true" structure, with $\mathbf{h}$ as a latent variable that explains the observed variations in $\mathbf{x}$.

SSL and Causality

Very often **most observations are formed by an extremely large number of underlying causes**, the brute force approach of encoding *all* possible factors of variations does not work: for instance in a visual scene, should the representation always encode all of the smallest objects in the background?

It is, then, necessary to decide what to encode into $\mathbf{h}$, i.e., to find a strategy to guide the network to keep only the **relevant** part of $\mathbf{h}$.

Two main strategies:

- use a supervised signal to guide the unsupervised process;
- use a much larger $\mathbf{h}$ when using only unsupervised learning.

What to Encode?

Another emerging strategy is to **change the definition of what is salient**.

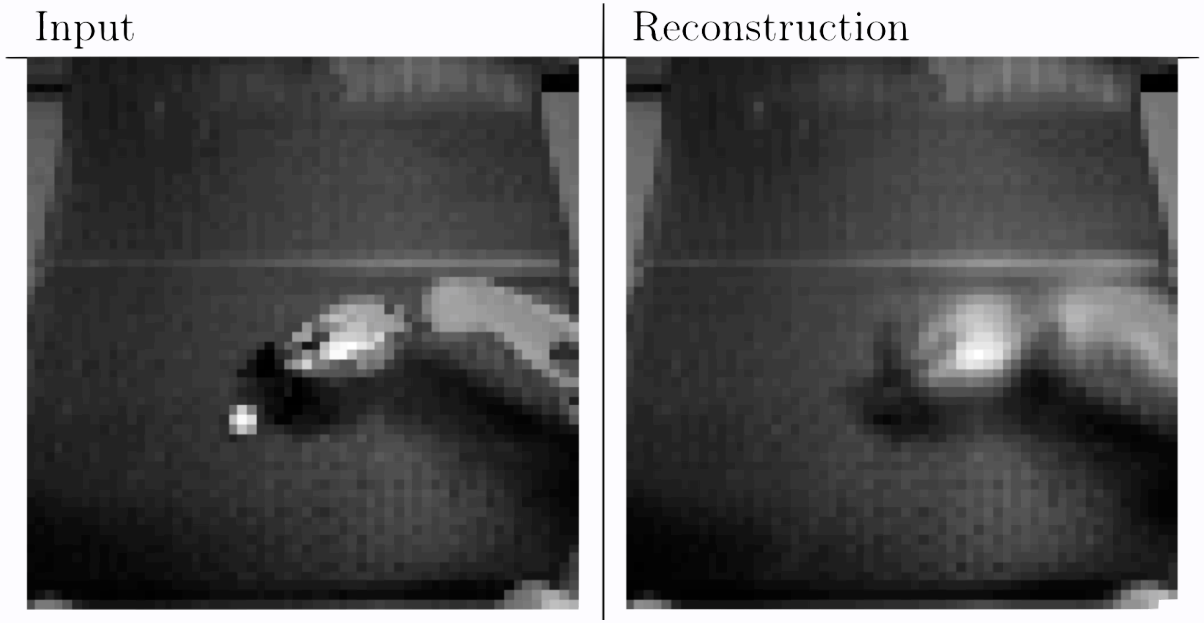
Historically, one would optimize against a fixed criterion often similar to **mean squared error** (MSE). This may problematic.

For instance, the MSE applied to pixels of images implicitly specifies that a cause is only relevant if affects the brightness of a large number of pixels.

What to Encode?

An autoencoder trained with mean squared error for a robotic task has failed to reconstruct the ping pong ball.

Image from the text book p. 544.



What to Encode?

Other definitions of salience are possible.

For example, if a group of pixels follows a highly recognizable pattern, even if that pattern does not involve extreme brightness or darkness, then that pattern could be considered extremely salient.

One way to implement such a definition of salience is to use **generative adversarial networks** (GANs). In GANs the salience is defined implicitly by a game played by an encoder trying to fool a discriminator.

Causal Factors are Robust

Very often, when we consider changes in distribution due to different domains, temporal non-stationarity, or when the nature of the task changes, the **causal mechanisms remain invariant** (the laws of the universe are constant) while the marginal distribution over the underlying causes can change.

Distributed Representation

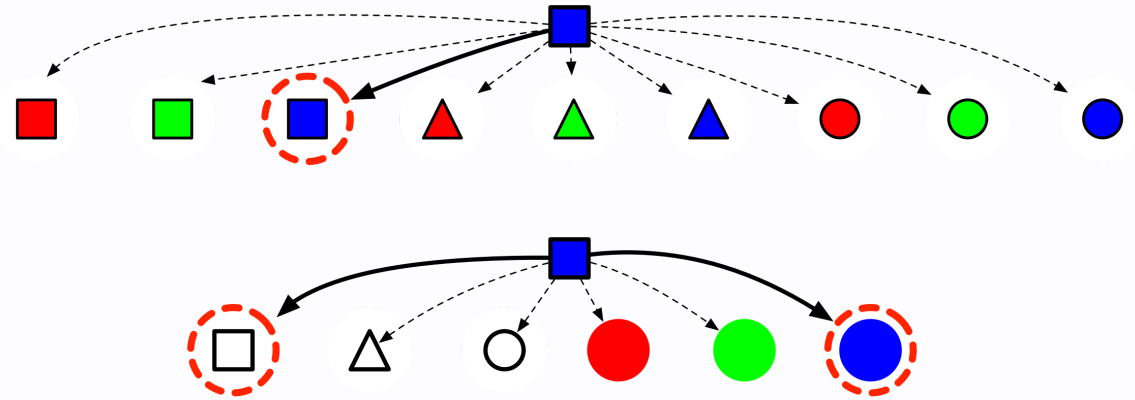
An interesting facet of neural networks is that they often learn distributed representations of the data.

> *A **distributed representation** is one where each object is described by many features and each feature is involved in the representation of many objects.*

Example

Consider the case where a system needs to learn the color and shapes of some objects:

- colors ∈ { red, green, blue }
- objects ∈ { square, circle, triangle }

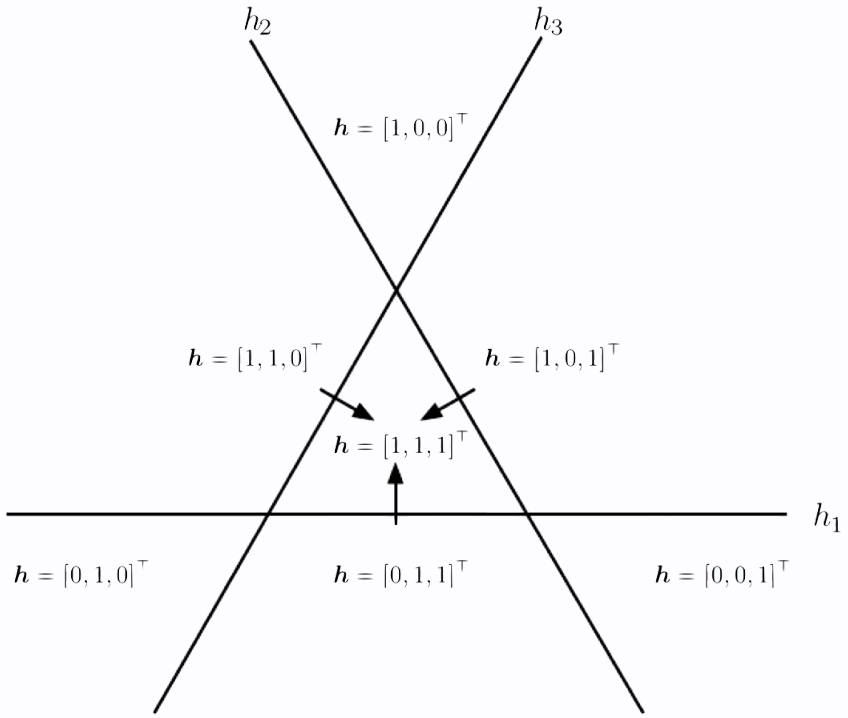


Distributed Representations: Generalization

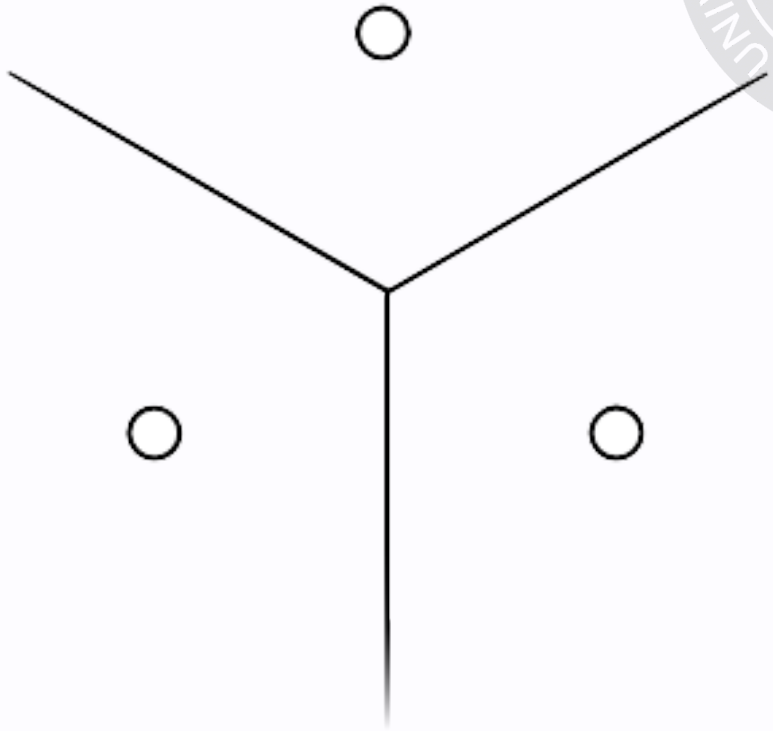
Distributed representations induce a rich similarity space, in which semantically close concepts (or inputs) are close in distance, a property that is absent from purely symbolic representations.

Example: "cat" and "dog" as symbols are as far apart as any other pair of symbols. If they are represented with a distributed representation including "having fur" and "having four legs" they can be immediately seen as related concepts.

Distributed Representations: Generalization



Distributed representation.



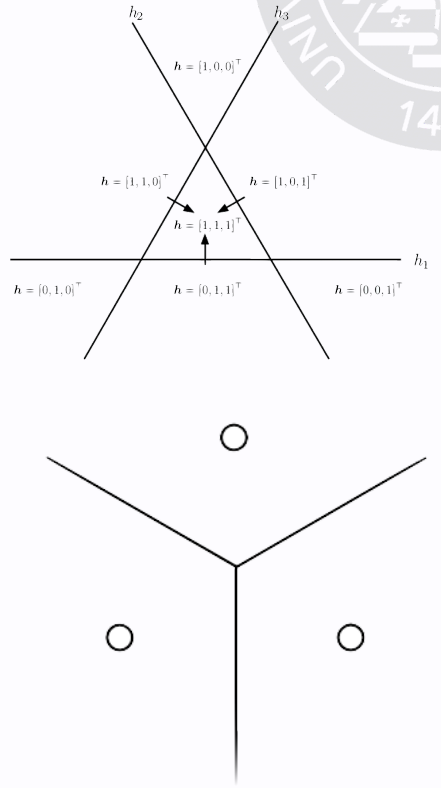
Non-distributed representation by the nearest neighbor algorithm.

Distributed representations **help in representing an exponential number of regions** using a linear number of parameters.

This **does not automatically translate in an exponential advantage for the classification algorithm**: e.g., a linear classifier on top of the distributedrepresentation is not able to assign different class identities to every neighboring region.

The VC dimension of a neural network of linear threshold units is only $O(w \log w)$, where w is the number of weights in the network.[1]

This implies that **despite the representation being able to distinguish between many *zones*, *not all zones* can be represented**.



1: **Vapnik-Chervonenkis dimension of neural nets.** *Bartlett, P. et al.,* The handbook of brain theory and neural networks, 2003.

VC Dimension

The **Vapnik-Chervonenkis** (VC) dimension is a measure of the *capacity* of a model (hypothesis) space. It is used in PAC (Probably Approximately Correct) theory to derive bounds on the generalization ability of a learning model.

Shattering a set

A set is said to be ***shattered*** *by a family of functions $\mathcal{H}$ if for any labeling of the points in the set there is a function in $\mathcal{H}$ that correctly labels all the points.*

VC Dimension

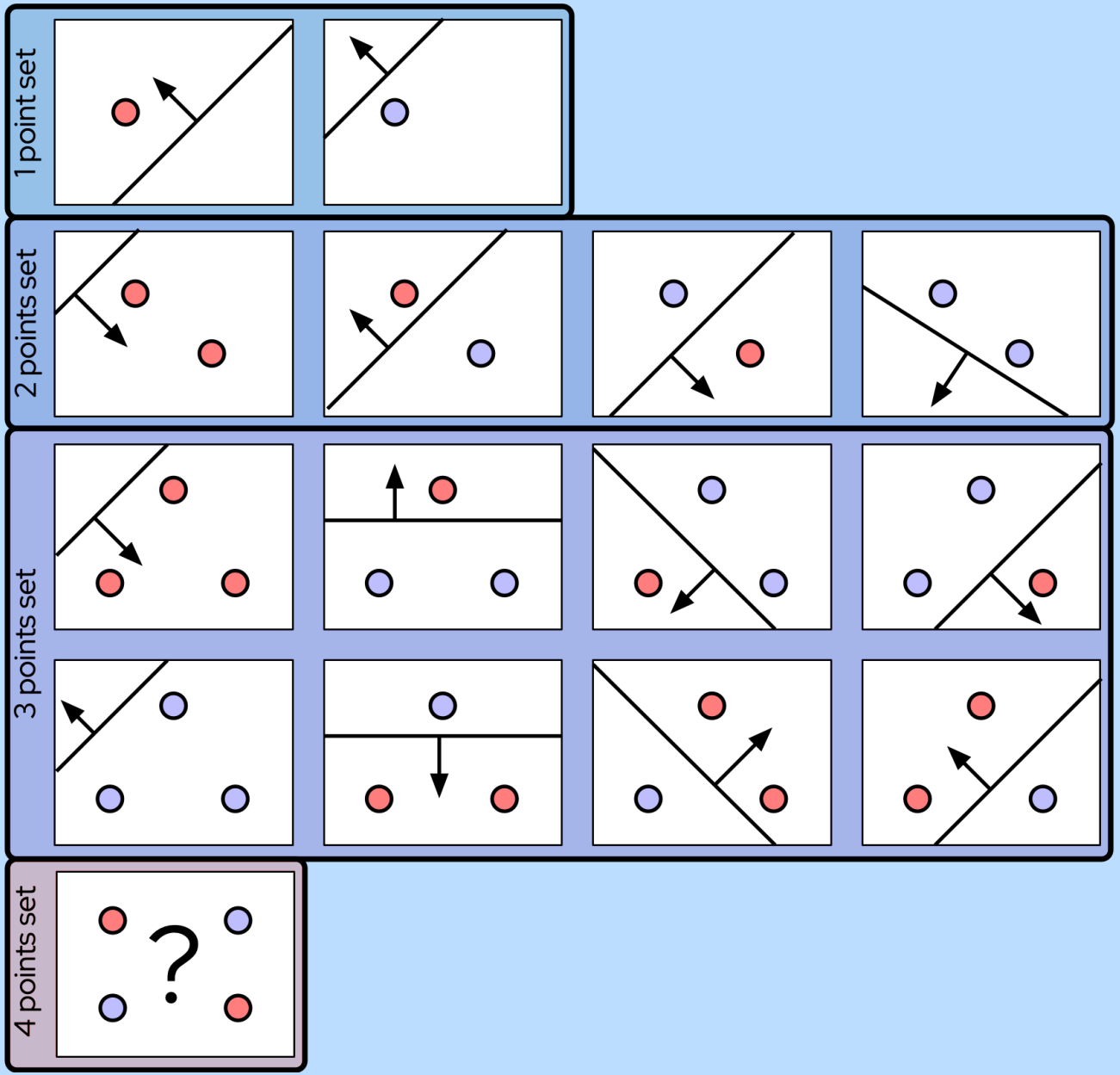
The VC dimension of an hypothesis space $\mathcal{H}$ defined over an instance space X is the size of the largest finite subset of X that can be ***shattered*** *by $\mathcal{H}$.*

Example

Consider the set of all linear classifiers in $\mathbb{R}^2$:

$F = \{f_{\mathbf{w}}(\mathbf{x}) | f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} \geq 0, \mathbf{w} \in \mathbb{R}^2\}$.

The VC dimension of F is 3.



Question

Which is the VC dimension of the set of constant functions (i.e., linear classifiers parallel to the x axis)?

Importance of VC Dimension

VC dimension is used to deduce bounds on the generalization capabilities of learning machines. So keeping the VC dimension of the hypothesis space small is usually a sensible goal in designing new learning systems.

Distributed Representations can be Interpretable

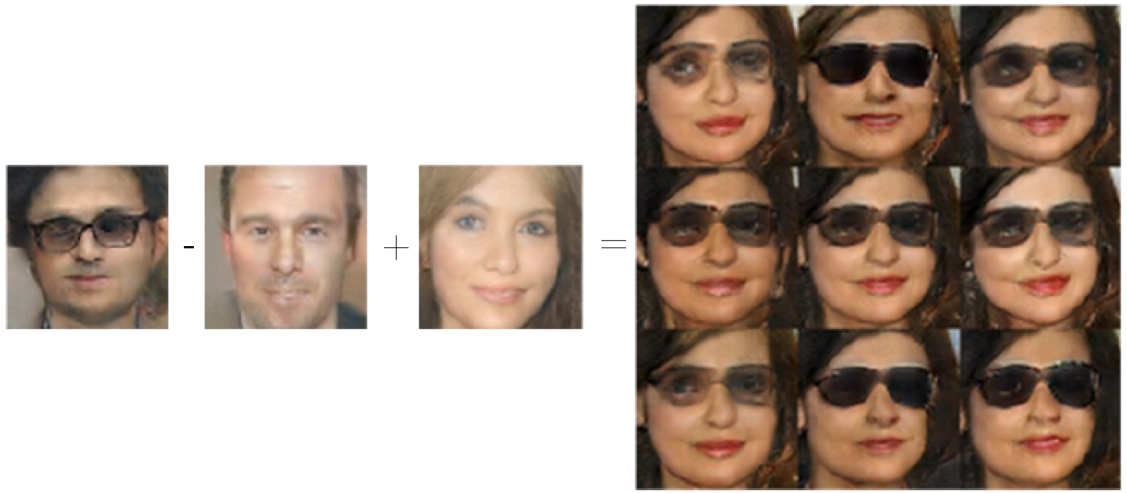
In Zhou et al (2014)[1] authors find that hidden units in a deep convolutional network trained on the ImageNet and Places benchmark datasets learn features that are very often interpretable, corresponding to a label that humans would naturally assign.

> ***Note***
>
> *While this can happen,* ***it is not the norm*** *and interpreting the representation learnt by a deep neural network is still one of the most important problems in the field.*

1: Object detectors emerge in deep scene cnns. *Zhou B, Khosla A, Lapedriza A, Oliva A, Torralba A,* Arxiv, 2014.

Distributed Representations

In Radford et al. (2015)[1], authors demonstrated that a generative model can learn a representation of images of faces, with separate directions in representation space capturing different underlying factors of variation.



1: **Unsupervised representation learning with deep convolutional generative adversarial networks.** *Radford, A., Metz, L., & Chintala, S. (2015),* Arxiv, 2015.

Exponential Gains from Depth

Exponential Gains from Depth

Theoretical results[1] concerning the expressive power of deep architectures state that there are families of functions that can be represented efficiently by an architecture of depth k, but would require an exponential number of hidden units (with respect to the input size) with insufficient depth.

1: **On the Expressive Power of Deep Learning: A Tensor Analysis.** *Nadav Cohen, Or Sharir, Amnon Shuashua,* JMLR: Workshop and Conference Proceedings vol 49:1–31, 2016.