Chapter 16

Representation Learning

What is a good representation? Many answers are possible, and this remains a question to be further explored in future research. What we propose as answer in this book is that in general, a good representation is one that makes further learning tasks easy. In an unsupervised learning setting, this could mean that the joint distribution of the different elements of the representation (e.g., elements of the representation vector $\boldsymbol{h}$) is one that is easy to model (e.g., in the extreme, these elements are marginally independent of each other). But that would not be enough: a representation that throws away all information (e.g., $\boldsymbol{h} = 0$ for all inputs $\boldsymbol{x}$) is very easy to model but is also useless. Hence we want to learn a representation that keeps the information (or at least all the relevant information, in the supervised case) and makes it easy to learn functions of interest from this representation.

In Chapter 1, we have introduced the notion of *representation*, the idea that some representations were more helpful (e.g. to classify objects from images or phonemes from speech) than others. As argued there, this suggests *learning representations* in order to "select" the best ones in a systematic way, i.e., by optimizing a function that maps raw data to its representation, instead of - or in addition to - handcrafting them. This motivation for learning input features is discussed in Section 6.7, and is one of the major side-effects of training a feedforward deep network (treated in Chapter 6), typically via supervised learning, i.e., when one has access to (input,target) pairs[1], available for some task of interest. In the case of supervised learning of deep nets, we learn a representation with the objective of selecting one that is best suited to the task of predicting targets given inputs.

Whereas supervised learning has been the workhorse of recent industrial successes of deep learning, the authors of this book believe that it is likely that a key

[1]typically obtained by *labeling* inputs with some target answer that we wish the computer would produce

element of future advances will be *unsupervised learning of representations*.

So how can we exploit the information in data if we don't have labeled examples? Or too few? Pure supervised learning with few labeled examples can easily overfit. On the other hand, humans (and other animals) can sometimes learn a task from just one or very few examples. How is that possible? Clearly they must rely on previously acquired knowledge, either innate or (more likely the case for humans) via previous learning experience. Can we discover good representations purely out of unlabeled examples? (this is treated in the first four sections of this chapter). Can we combine unlabeled examples (which are often easy to obtain) with labeled examples? (this is semi-supervised learning, Section 16.3). And what if instead of one task we have many tasks that could share the same representation or parts of it? (this is multi-task learning, discussed in Section 7.12). What if we have "training tasks" (on which enough labeled examples are available) as well as "test tasks" (not known at the time of learning the representation, and for which only very few labeled examples will be provided)? What if the test task is similar but different from the training task? (this is transfer learning and domain adaptation, discussed in Section 16.2).

16.1 Greedy Layerwise Unsupervised Pre-Training

Unsupervised learning played a key historical role in the revival of deep neural networks, allowing for the first time to train a deep supervised network. We call this procedure *unsupervised pre-training*, or more precisely, *greedy layer-wise unsupervised pre-training*, and it is the topic of this section.

This recipe relies on a one-layer representation learning algorithm such as those introduced in this part of the book, i.e., the auto-encoders (Chapter 15) and the RBM (Section 20.2). Each layer is pre-trained by unsupervised learning, taking the output of the previous layer and producing as output a new representation of the data, whose distribution (or its relation to other variables such as categories to predict) is hopefully simpler.

Greedy layerwise training procedures based on unsupervised criteria have long been used to sidestep the difficulty of jointly training deep architectures with supervision. This approach dates back at least as far as the Neocognitron (Fukushima, 1975). The deep learning renaissance of 2006 began with the discovery that this greedy learning procedure could be used to find a good initialization for a joint learning procedure, and the skilled, professional use of this strategy to outperform academically respected learning algorithms including support vector machines on standard benchmarks Hinton *et al.* (2006); Hinton and Salakhutdinov (2006); Bengio *et al.* (2007a); Ranzato *et al.* (2007a).

It is called *layerwise* because it proceeds one layer at a time, training the

k-th layer while keeping the previous ones fixed. It is called *unsupervised* because each layer is trained with an unsupervised representation learning algorithm. It is called *greedy* because the different layers are not jointly trained with respect to a global training objective, which could make the procedure sub-optimal. In particular, the lower layers (which are first trained) are not adapted after the upper layers are introduced. However it is also called *pre-training*, because it is supposed to be only a first step before a joint training algorithm is applied to *fine-tune* all the layers together with respect to a criterion of interest. In the context of a supervised learning task, it can be viewed as a regularizer (see Chapter 7) and a sophisticated form of parameter initialization.

When we refer to pre-training we will be referring to a specific protocol with two main phases of training: the pretraining phase and the fine-tuning phase. No matter what kind of unsupervised learning algorithm or what model type you employ, in the vast majority of cases, the overall training scheme is nearly the same. While the choice of unsupervised learning algorithm will obviously impact the details, in the abstract, most applications of unsupervised pre-training follows this basic protocol.

As outlined in Algorithm 16.1, in the *pretraining phase*, the layers of the model are trained, in order, in an unsupervised way on their input, beginning with the bottom layer, i.e. the one in direct contact with the input data. Next, the second lowest layer is trained taking the activations of the first layer hidden units as input for unsupervised training. Pretraining proceeds in this fashion, from bottom to top, with each layer training on the "output" or activations of the hidden units of the layer below. After the last layer is pretrained, a supervised layer is put on top, and all the layers are jointly trained with respect to the overall supervised training criterion. In other words, the pre-training was only used to initialize a deep supervised neural network (which could be a convolutional neural network (Ranzato *et al.*, 2007a)). This is illustrated in Figure 16.1.

However, greedy layerwise unsupervised pre-training can also be used as initialization for other unsupervised learning algorithms, such as deep auto-encoders (Hinton and Salakhutdinov, 2006), deep belief networks (Hinton *et al.*, 2006) (Section 20.4), or deep Boltzmann machines (Salakhutdinov and Hinton, 2009a) (Section 20.5).

As discussed in Section 8.7.3, it is also possible to have greedy layerwise *supervised* pre-training, to help *optimize* deep supervised networks. This builds on the premise that training a shallow network is easier than training a deep one, which seems to have been validated in several contexts (Erhan *et al.*, 2010).

Algorithm 16.1 Greedy layer-wise unsupervised pre-training protocol.
Given the following: Unsupervised feature learner $\mathcal{L}$, which takes a training set $\mathcal{D}$ of examples and returns an encoder or feature function $f = \mathcal{L}(\mathcal{D})$. The raw input data is $\boldsymbol{X}$, with one row per example and $f(\boldsymbol{X})$ is the dataset used by the second level unsupervised feature learner. In the case fine-tuning is performed, we use a learner $\mathcal{T}$ which takes an initial function f, input examples $\boldsymbol{X}$ (and in the supervised fine-tuning case, associated targets $\boldsymbol{Y}$), and returns a tuned function. The number of stages is M.

```
$\mathcal{D}^{(0)} = \boldsymbol{X}$
$f \leftarrow$ Identity function
**for** $k = 1 \ldots, M$ **do**
    $f^{(k)} = \mathcal{L}(\mathcal{D})$
    $f \leftarrow f^{(k)} \circ f$
**end for**
**if** *fine-tuning* ***then***
    $f \leftarrow \mathcal{T}(f, \boldsymbol{X}, \boldsymbol{Y})$
***end if***
**Return** $f$
```

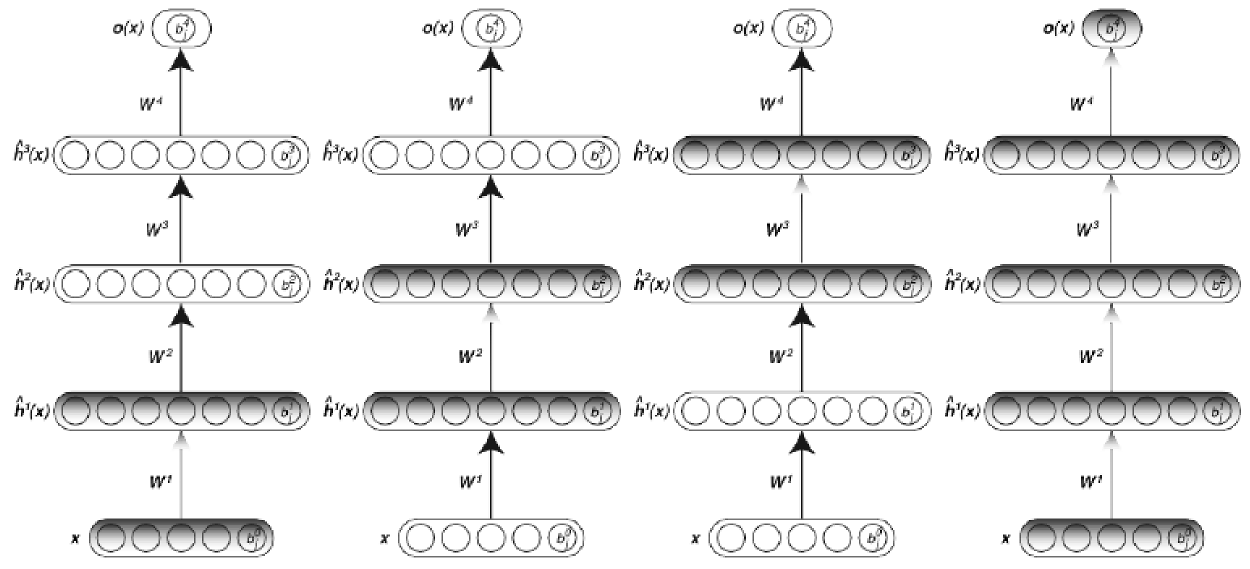


Figure 16.1: Illustration of the greedy layer-wise unsupervised pre-training scheme, in the case of a network with 3 hidden layers. The protocol proceeds in 4 phases (one per hidden layer, plus the final supervised fine-tuning phase), from left to right. For the unsupervised steps, each layer (darker grey) is trained to learn a better representation of the output of the previously trained layer (initially, the raw input). These representations learned by unsupervised learning form the initialization of a deep supervised net, which is then trained (fine-tuned) as usual (last phase, right), with all parameters being free to change (darker grey).

16.1.1 Why Does Unsupervised Pre-Training Work?

What has been observed on several datasets starting in 2006 (Hinton *et al.*, 2006; Bengio *et al.*, 2007a; Ranzato *et al.*, 2007a) is that greedy layer-wise unsupervised pre-training can yield substantial improvements in test error for classification tasks. Later work suggested that the improvements were less marked (or not even visible) when very large labeled datasets are available, although the boundary between the two behaviors remains to be clarified, i.e., it may not just be an issue of number of labeled examples but also how this relates to the complexity of the function to be learned.

A question that thus naturally arises is the following: why and when does unsupervised pre-training work? Although previous studies have mostly focused on the case when the final task is supervised (with supervised fine-tuning), it is also interesting to keep in mind that one gets improvements in terms of both training and test performance in the case of unsupervised fine-tuning, e.g., when training deep auto-encoders (Hinton and Salakhutdinov, 2006).

This "why does it work" question is at the center of the paper by Erhan *et al.* (2010), and their experiments focused on the supervised fine-tuning case. They consider different machine learning hypotheses to explain the results observed, and attempted to confirm those via experiments. We summarize some of this investigation here.

First of all, they studied the *trajectories* of neural networks during supervised fine-tuning, and evaluated how different they were depending on initial conditions, i.e., due to random initialization or due to performing unsupervised pre-training or not. The main result is illustrated and discussed in Figures 16.2 and 16.2. Note that it would not make sense to plot the evolution of parameters of these networks directly, because the same input-to-output function can be represented by different parameter values (e.g., by relabeling the hidden units). Instead, this work plots the trajectories in *function space*, by considering the output of a network (the class probability predictions) for a given set of test examples as a proxy for the function computed. By concatenating all these outputs (over say 1000 examples) and doing dimensionality reduction on these vectors, we obtain the kinds of plots illustrated in the figure.

The main conclusions of these kinds of plots are the following:

1. Each training trajectory goes to a different place, i.e., different trajectories do not converge to the same place. These "places" might be in the vicinity of a local minimum or as we understand it better now (Dauphin *et al.*, 2014) these are more likely to be an "apparent local minimum" in the region of flat derivatives near a saddle point. This suggests that the number of these apparent local minima is huge, and this also is in agreement with

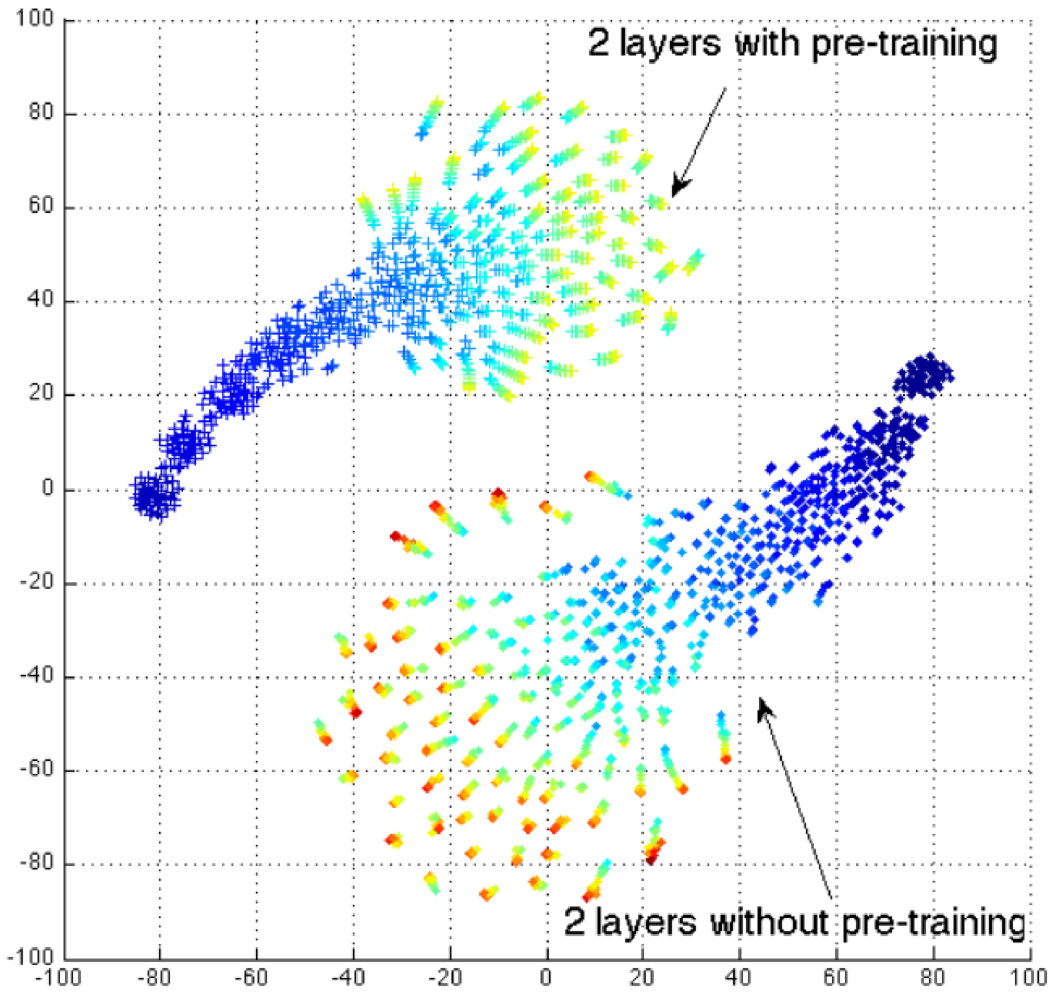


Figure 16.2: Illustrations of the trajectories of different neural networks in *function space* (not parameter space, to avoid the issue of many-to-one mapping from parameter vector to function), with different random initializations and with or without unsupervised pre-training. Each plus or diamond point corresponds to a different neural network, at a particular time during its training trajectory, with the function it computes projected to 2-D by t-SNE (van der Maaten and Hinton, 2008a) (this figure) or by Isomap (Tenenbaum *et al.*, 2000) (Figure 16.3). TODO: should the t be in math mode? Color indicates the number of training epochs. What we see is that no two networks converge to the same function (so a large number of *apparent* local minima seems to exist), and that networks initialized with pre-training learn very different functions, in a region of function space that does not overlap at all with those learned by networks without pre-training. Such curves were introduced by Erhan *et al.* (2010) and are reproduced here with permission.

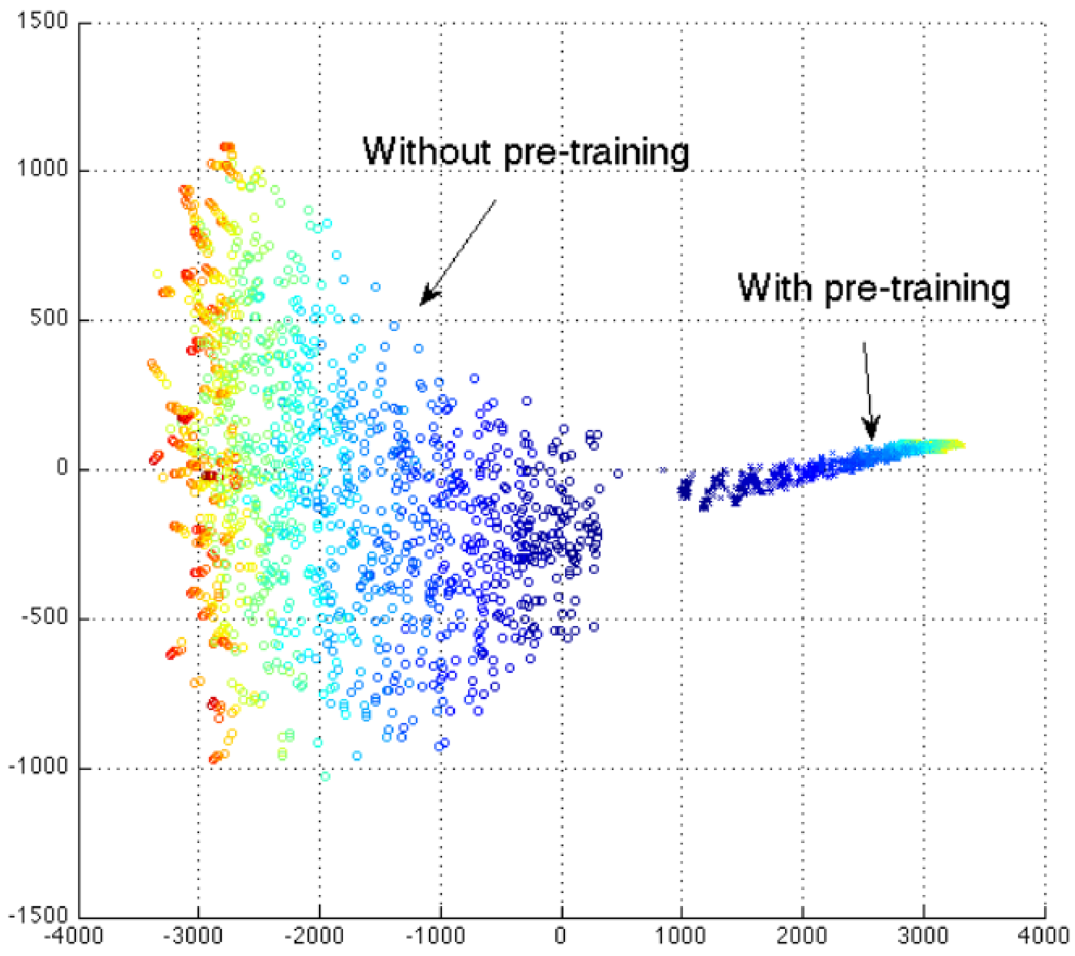
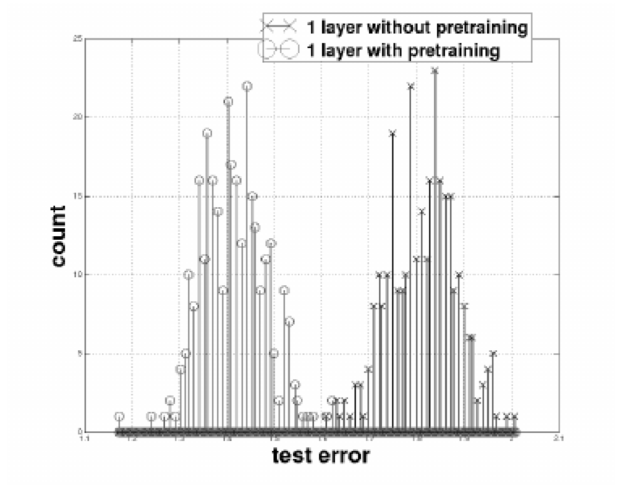


Figure 16.3: See Figure 16.2's caption. This figure only differs in the use of Isomap (Tenenbaum *et al.*, 2000) rather than t-SNE (van der Maaten and Hinton, 2008b) for dimensionality reduction. Note that Isomap tries to preserve global relative distances (and hence volumes), whereas t-SNE only cares about preserving local geometry and neighborhood relationships. We see with the Isomap dimensionality reduction that the volume in function space occupied by the networks with pre-training is much smaller (in fact that volume gets reduced rather than increased, during training), suggesting that the set of solutions enjoy smaller variance, which would be consistent with the observed improvements in generalization error. Such curves were introduced by Erhan *et al.* (2010) and are reproduced here with permission.



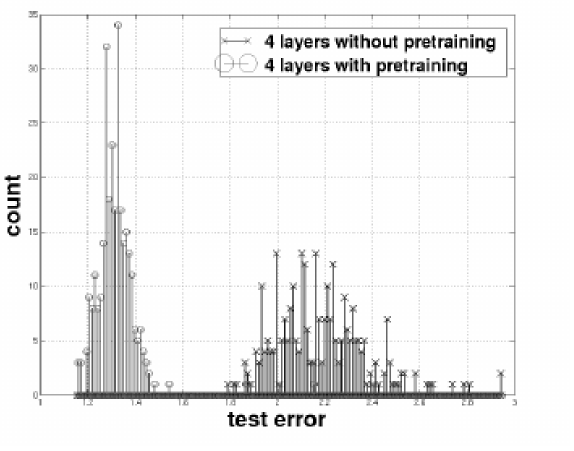


Figure 16.4: Histograms presenting the test errors obtained on `MNIST` using denoising auto-encoder models trained with or without pre-training (400 different initializations each). **Left**: 1 hidden layer. **Right**: 4 hidden layers. We see that the advantage brought by pre-training increases with depth, both in terms of mean error and in terms of the variance of the error (w.r.t. random initialization).
TODO: figure credit saying these came from Erhan 2010....

theory (Dauphin *et al.*, 2014; Choromanska *et al.*, 2014).

2. Depending on whether we initialize with unsupervised pre-training or not, very different functions (in function space) are obtained, covering regions that do not overlap. Hence there is a qualitative effect due to unsupervised pre-training.

3. With unsupervised pre-training, the region of space covered by the solutions associated with different initializations *shrinks* as we consider more training iterations, whereas it *grows* without unsupervised pre-training. This is only apparent in the visualization of Figure 16.3, which attempts to preserve volume. A larger region is bad for generalization (because not all these functions can be the right one together), yielding higher variance. This is consistent with the better generalization observed with unsupervised pre-training.

Another interesting effect is that the advantage of pre-training seems to increase with depth, as illustrated in Figure 16.4, with both the mean and the variance of the error decreasing more for deeper networks.

An important question is whether the advantage brought by pre-training can be seen as a form of regularizer (which could help test error but hurt training error) or simply a way to find a better minimizer of training error (e.g., by initializing near a better minimum of training error). The experiments suggest pre-training actually acts as a regularizer, i.e., hurting training error at least in some cases (with deeper networks). So if it also helps optimization, it is only because it initializes closer to a good solution from the point of view of generalization, not necessarily from the point of view of the training set.

How could unsupervised pre-training act as regularizer? Simply by imposing an extra constraint: the learned representations should not only be consistent with better predicting outputs $\mathbf{y}$ but they should also be consistent with better capturing the variations in the input $\mathbf{x}$, i.e., modeling $P(\mathbf{x})$. This is associated implicitly with a prior, i.e., that $P(\mathbf{y}|\mathbf{x})$ and $P(\mathbf{x})$ share structure, i.e., that learning about $P(\mathbf{x})$ can help to generalize better on $P(\mathbf{y} \mid \mathbf{x})$. Obviously this needs not be the case in general, e.g., if $\mathbf{y}$ is an effect of $\mathbf{x}$. However, if $\mathbf{y}$ is a cause of $\mathbf{x}$, then we would expect this a priori assumption to be correct, as discussed at greater length in Section 16.4 in the context of semi-supervised learning.

A disadvantage of unsupervised pre-training is that it is difficult to choose the capacity hyperparameters (such as when to stop training) for the pre-training phases. An expensive option is to try many different values of these hyperparameters and choose the one which gives the best supervised learning error after fine-tuning. Another potential disadvantage is that unsupervised pre-training may require larger representations than what would be necessarily strictly for the task at hand, since presumably, $\mathbf{y}$ is only one of the factors that explain $\mathbf{x}$.

Today, as many deep learning researchers and practitioners have moved to working with very large labeled datasets, unsupervised pre-training has become less popular in favor of other forms of regularization such as dropout – to be discussed in section 7.11. Nevertheless, unsupervised pre-training remains an important tool in the deep learning toolbox and should particularly be considered when the number of labeled examples is low, such as in the semi-supervised, domain adaptation and transfer learning settings, discussed next.

16.2 Transfer Learning and Domain Adaptation

Transfer learning and domain adaptation refer to the situation where what has been learned in one setting (i.e., distribution P_1) is exploited to improve generalization in another setting (say distribution P_2).

In the case of *transfer learning*, we consider that the task is different but many of the factors that explain the variations in P_1 are relevant to the variations that need to be captured for learning P_2. This is typically understood in a supervised learning context, where the input is the same but the target may be of a different nature, e.g., learn about visual categories that are different in the first and the second setting. If there is a lot more data in the first setting (sampled from P_1), then that may help to learn representations that are useful to quickly generalize when examples of P_2 are drawn. For example, many visual categories *share* low-level notions of edges and visual shapes, the effects of geometric changes, changes in lighting, etc. In general, transfer learning, multi-task learning (Section 7.12), and domain adaptation can be achieved via representation learning when there exist features that would be useful for the different settings or tasks, i.e., there are *shared underlying factors.* This is illustrated in Figure 7.6, with shared lower layers and task-dependent upper layers.

However, sometimes, what is shared among the different tasks is not the semantics of the input but the semantics of the output, or maybe the input needs to be treated differently (e.g., consider user adaptation or speaker adaptation). In that case, it makes more sense to share the upper layers (near the output) of the neural network, and have a task-specific pre-processing, as illustrated in Figure 16.5.

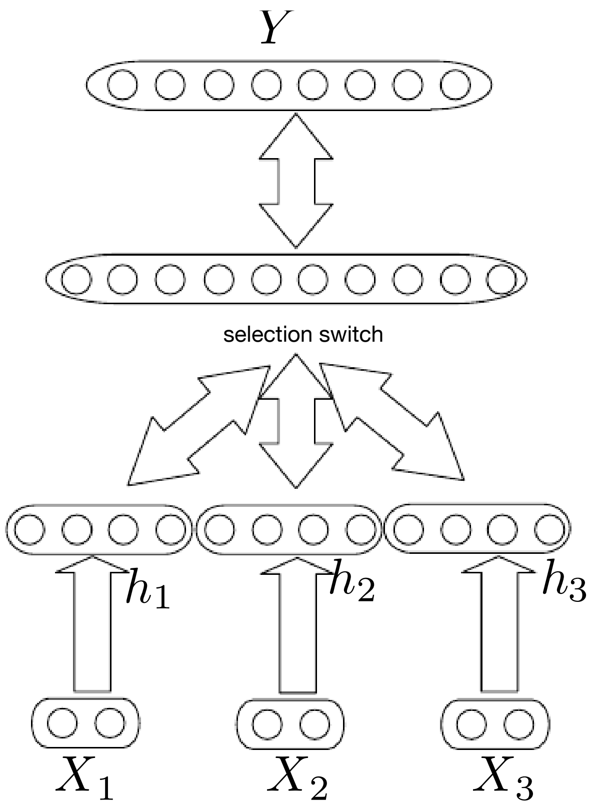


Figure 16.5: Example of architecture for multi-task or transfer learning when the output variable Y has the same semantics for all tasks while the input variable X has a different meaning (and possibly even a different dimension) for each task (or, for example, each user), called X_1, X_2 and X_3 for three tasks in the figure. The lower levels (up to the selection switch) are task-specific, while the upper levels are shared. The lower levels learn to translate their task-specific input into a generic set of features.

In the related case of *domain adaptation*, we consider that the task (and the optimal input-to-output mapping) is the same but the input distribution is slightly different. For example, if we predict sentiment (positive or negative judgement) associated with textual comments posted on the web, the first setting may refer to consumer comments about books, videos and music, while the second setting may refer to televisions or other products. One can imagine that there is an underlying function that tells whether any statement is positive, neutral or negative, but of course the vocabulary, style, accent, may vary from one domain to another, making it more difficult to generalize across domains. Simple unsupervised pre-training (with denoising auto-encoders) has been found to be very successful for sentiment analysis with domain adaptation (Glorot *et al.*, 2011c).

A related problem is that of *concept drift*, which we can view as a form of transfer learning due to gradual changes in the data distribution over time. Both concept drift and transfer learning can be viewed as particular forms of multi-task learning (Section 7.12). Whereas multi-task learning is typically considered in the context of supervised learning, the more general notion of transfer learning is applicable for unsupervised learning and reinforcement learning as well. Figure 7.6 illustrates an architecture in which different tasks share underlying features or factors, taken from a larger pool that explain the variations in the input.

In all of these cases, the objective is to take advantage of data from a first setting to extract information that may be useful when learning or even when directly making predictions in the second setting. One of the potential advantages of representation learning for such generalization challenges, and especially of deep representation learning, is that it may considerably help to generalize by extracting and disentangling a set of explanatory factors from data of the first setting, some of which may be relevant to the second setting. In the case of object recognition from an image, many of the factors of variation that explain visual categories in natural images remain the same when we move from one set of categories to another.

This discussion raises a very interesting and important question which is one of the core questions of this book: *what is a good representation?* Is it possible to learn representations that disentangle the underlying factors of variation? This theme is further explored at the end of this chapter (Section 16.4 and beyond). We claim that learning the most *abstract features* helps to maximize our chances of success in transfer learning, domain adaptation, or concept drift. More abstract features are more general and more likely to be close to the underlying causal factor, i.e., be relevant over many domains, many categories, and many time periods.

A good example of the success of unsupervised deep learning for transfer learning is its success in two competitions that took place in 2011, with results presented at ICML 2011 (and IJCNN 2011) in one case (Mesnil *et al.*, 2011) (the Transfer Learning Challenge, `http://www.causality.inf.ethz.ch/unsupervised-learning.php`) and at NIPS 2011 (Goodfellow *et al.*, 2011) in the other case (the Transfer Learning Challenge that was held as part of the NIPS'2011 workshop on Challenges in learning hierarchical models, `https://sites.google.com/site/nips2011workshop/transfer-lea`

In the first of these competitions, the experimental setup is the following. Each participant is first given a dataset from the first setting (from distribution P_1), basically illustrating examples of some set of categories. The participants must use this to learn a good feature space (mapping the raw input to some representation), such that when we apply this learned transformation to inputs from the transfer setting (distribution P_2), a linear classifier can be trained and generalize well from very few labeled examples. Figure 16.6 illustrates one of the most striking results: as we consider deeper and deeper representations (learned in a purely unsupervised way from data of the first setting P_1), the learning curve on the new categories of the second (transfer) setting P_2 becomes much better, i.e., less labeled examples of the transfer tasks are necessary to achieve the apparently asymptotic generalization performance.

An extreme form of transfer learning is *one-shot learning* or even *zero-shot learning* or *zero-data learning*, where one or even zero example of the new task are given.

One-shot learning (Fei-Fei *et al.*, 2006) is possible because, in the learned representation, the new task corresponds to a very simple region, such as a ball-like region or the region around a corner of the space (in a high dimensional space, there are exponentially many corners). This works to the extent that the factors of variation corresponding to these invariances have been cleanly separated from other factors, in the learned representation space, and we have somehow learned which factors do and do not matter when discriminating objects of certain categories.

Zero-data learning (Larochelle *et al.*, 2008) and zero-shot learning (Richard Socher and Ng, 2013) are only possible because additional information has been exploited during training that provides representations of the "task" or "context", helping the learner figure out what is expected, even though no example of the new task has ever been seen. For example, in a multi-task learning setting, if each task is associated with a set of features, i.e., a distributed representation (that is always provided as an extra input, in addition to the ordinary input associated with the task), then one can generalize to new tasks based on the similarity between the new task and the old tasks, as illustrated in Figure 16.7. One learns a parametrized function from inputs to outputs, parametrized by the task representation. In the case of zero-shot learning (Richard Socher and Ng, 2013), the "task" is a representation of a semantic object (such as a word), and its repre-

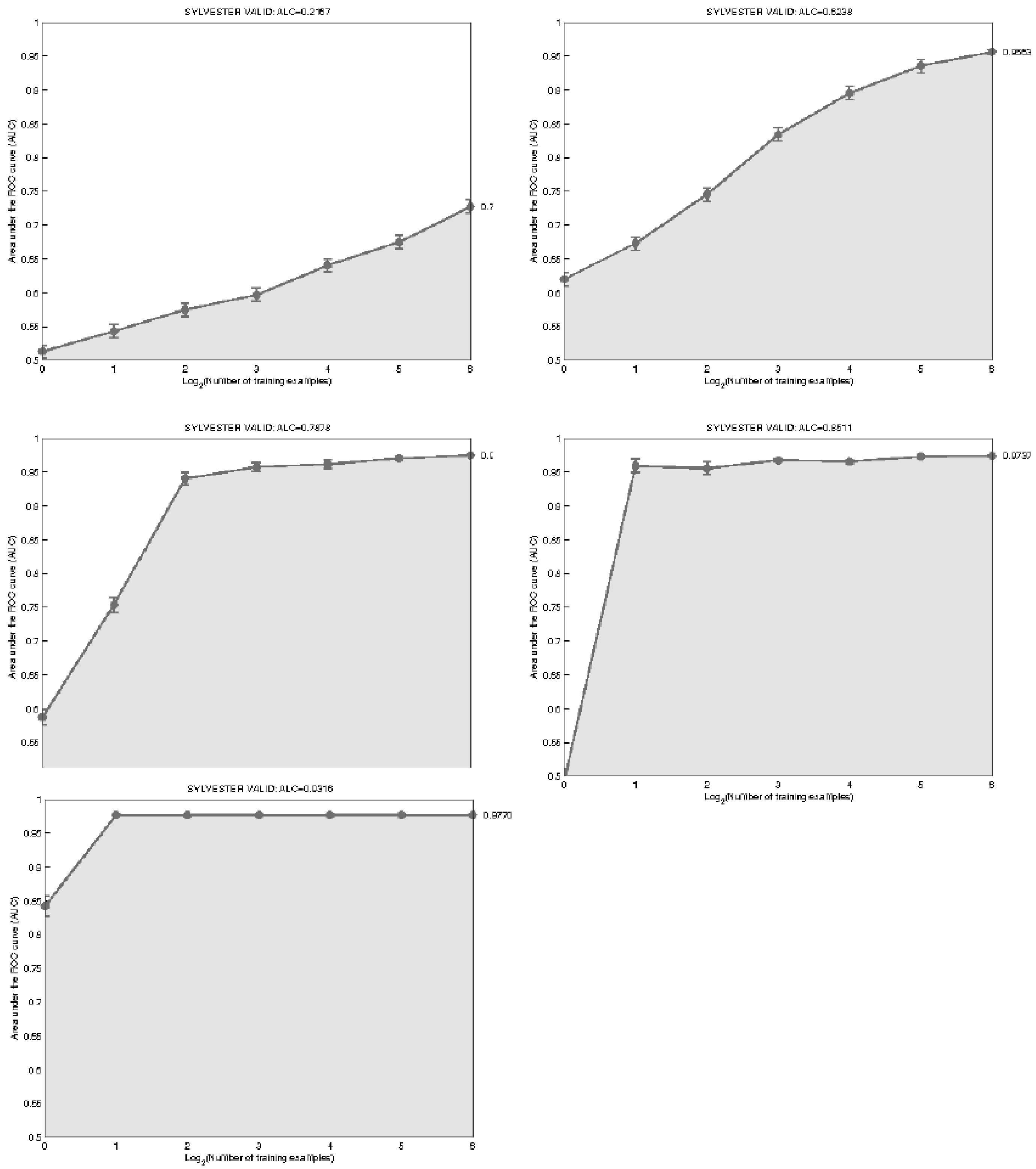


Figure 16.6: Results obtained on the Sylvester validation set (Transfer Learning Challenge). From left to right and top to bottom, respectively 0, 1, 2, 3, and 4 pre-trained layers. Horizontal axis is logarithm of number of labeled training examples on transfer setting (test task). Vertical axis is Area Under the Curve, which reflects classification accuracy. With deeper representations (learned unsupervised), the learning curves considerably improve, requiring fewer labeled examples to achieve the best generalization.

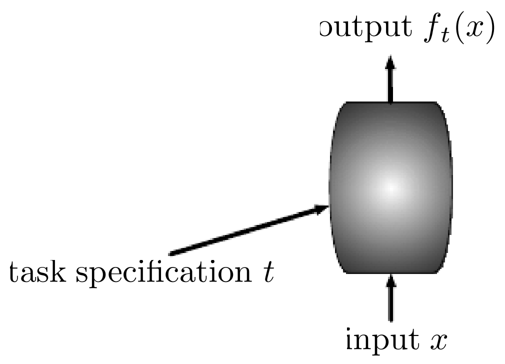


Figure 16.7: Figure illustrating how zero-data or zero-shot learning is possible. The trick is that the new context or task on which no example is given but on which we want a prediction is *represented* (with an input t), e.g., with a set of features, i.e., a distributed representation, and that representation is used by the predictor $f_t(x)$. If t was a one-hot vector for each task, then it would not be possible to generalize to a new task, but with a distributed representation the learner can benefit from the meaning of the individual task features (as they influence the relationship between inputs x and targets y, say), learned on other tasks for which examples are available.

sentation has already been learned from data relating different semantic objects together (such as natural language data, relating words together). On the other hand, for some of the tasks (e.g., words) one has data associating the variables of interest (e.g., words, pixels in images). Thus one can generalize and associate images to words for which no labeled images were previously shown to the learner. A similar phenomenon happens in machine translation (Klementiev *et al.*, 2012; Mikolov *et al.*, 2013; Gouws *et al.*, 2014): we have words in one language, and the relationships between words can be learned from unilingual corpora; on the other hand, we have translated sentences which relate words in one language with words in the other. Even though we may not have labeled examples translating word A in language X to word B in language Y, we can generalize and guess a translation for word A because we have learned a distributed representation for words in language X, a distributed representation for words in language Y, and created a link (possibly two-way) relating the two spaces, via translation examples. Note that this transfer will be most successful if all three ingredients (the two representations and the relations between them) are learned jointly.

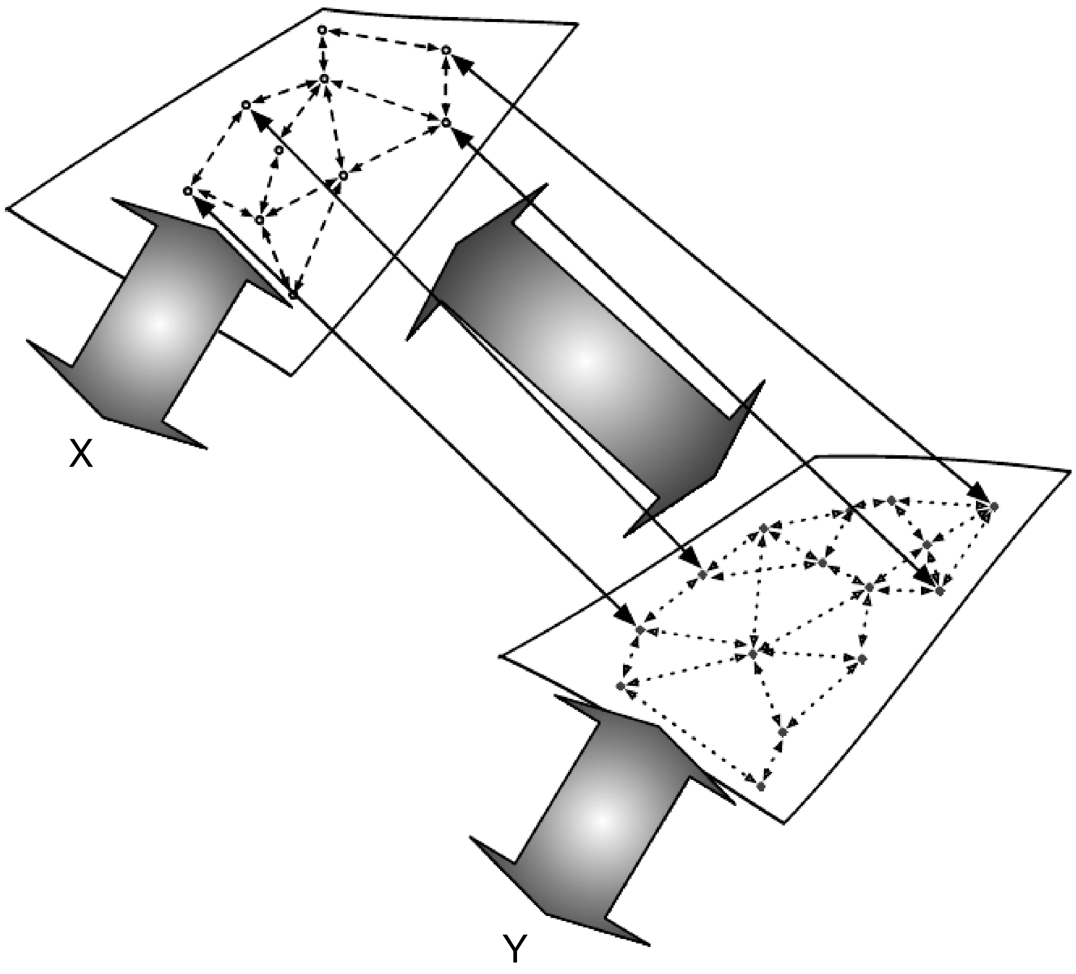


Figure 16.8: Transfer learning between two domains corresponds to zero-shot learning. A first set of data (dashed arrows) can be used to relate examples in one domain (top left, $\boldsymbol{X}$) and fix a relationship between their representations, a second set of data (dotted arrows) can be used to similarly relate examples and their representation in the other domain (bottom right, $\boldsymbol{Y}$), while a third dataset (full large arrows) *anchors* the two representations together, with examples consisting of pairs $(\boldsymbol{x}, \boldsymbol{y})$ taken from the two domains. In this way, one can for example associate an image to a word, even if no images of that word were ever presented, simply because word-representations (top) and image-representations (bottom) have been learned jointly with a two-way relationship between them.

This is illustrated in Figure 16.8, where we see that zero-shot learning is a particular form of transfer learning. The same principle explains how one can perform *multi-modal learning*, capturing a representation in one modality, a representation in the other, and the relationship (in general a joint distribution) between pairs $(\boldsymbol{x}, \boldsymbol{y})$ consisting of one observation $\boldsymbol{x}$ in one modality and another observation $\boldsymbol{y}$ in the other modality (Srivastava and Salakhutdinov, 2012). By learning all three sets of parameters (from $\boldsymbol{x}$ to its representation, from $\boldsymbol{y}$ to its representation, and the relationship between the two representation), concepts in one map are anchored in the other, and vice-versa, allowing one to meaningfully generalize to new pairs.

16.3 Semi-Supervised Learning

As discussed in Section 16.1.1 on the advantages of unsupervised pre-training, unsupervised learning can have a regularization effect in the context of supervised learning. This fits in the more general category of combining unlabeled examples with unknown distribution $P(\mathbf{x})$ with labeled examples $(\mathbf{x}, \mathbf{y})$, with the objective of estimating $P(\mathbf{y} \mid \mathbf{x})$. Exploiting unlabeled examples to improve performance on a labeled set is the driving idea behind semi-supervised learning (Chapelle *et al.*, 2006). For example, one can use unsupervised learning to map X into a representation (also called embedding) such that two examples $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ that belong to the same cluster (or are reachable through a short path going through neighboring examples in the training set) end up having nearby embeddings. One can then use supervised learning (e.g., a linear classifier) in that new space and achieve better generalization in many cases (Belkin and Niyogi, 2002; Chapelle *et al.*, 2003). A long-standing variant of this approach is the application of Principal Components Analysis as a pre-processing step before applying a classifier (on the projected data). In these models, the data is first transformed in a new representation using unsupervised learning, and a supervised classifier is stacked on top, learning to map the data in this new representation into class predictions.

Instead of having separate unsupervised and supervised components in the model, one can consider models in which $P(\mathbf{x})$ (or $P(\mathbf{x}, \mathbf{y})$) and $P(\mathbf{y} \mid \mathbf{x})$ share parameters (or whose parameters are connected in some way), and one can trade-off the supervised criterion $-\log P(\mathbf{y} \mid \mathbf{x})$ with the unsupervised or generative one ($-\log P(\mathbf{x})$ or $-\log P(\mathbf{x}, \mathbf{y})$). It can then be seen that the generative criterion corresponds to a particular form of prior (Lasserre *et al.*, 2006), namely that the structure of $P(\mathbf{x})$ is connected to the structure of $P(\mathbf{y} \mid \mathbf{x})$ in a way that is captured by the shared parametrization. By controlling how much of the generative criterion is included in the total criterion, one can find a better trade-off than with a purely generative or a purely discriminative training criterion (Lasserre *et al.*, 2006; Larochelle and Bengio, 2008b).

In the context of deep architectures, a very interesting application of these ideas involves adding an unsupervised embedding criterion at each layer (or only one intermediate layer) to a traditional supervised criterion (Weston *et al.*, 2008). This has been shown to be a powerful semi-supervised learning strategy, and is an alternative to the unsupervised pre-training approach described earlier in this chapter, which also combine unsupervised learning with supervised learning.

In the context of scarcity of labeled data (and abundance of unlabeled data), deep architectures have shown promise as well. Salakhutdinov and Hinton (2008) describe a method for learning the covariance matrix of a Gaussian Process, in which the usage of unlabeled examples for modeling $P(\mathbf{x})$ improves $P(\mathbf{y} \mid \mathbf{x})$ quite significantly. Note that such a result is to be expected: with few labeled samples, modeling $P(\mathbf{x})$ usually helps, as argued below (Section 16.4). These results show that even in the context of *abundant labeled data*, unsupervised pre-training can have a pronounced positive effect on generalization: a somewhat surprising conclusion.

16.4 Semi-Supervised Learning and Disentangling Underlying Causal Factors

What we put forward as a hypothesis, going a bit further, is that an *ideal representation* is one that *disentangles the underlying causal factors of variation that generated the observed data.* Note that this may be different from "easy to model", but we further assume that for most problems of interest, these two properties coincide: once we "understand" the underlying explanations for what we observe, it generally becomes easy to predict one thing from others.

A very basic question is whether unsupervised learning on input variables $\mathbf{x}$ can yield representations that are useful when later trying to learn to predict some target variable $\mathbf{y}$, given $\mathbf{x}$. More generally, when does semi-supervised learning work? See also Section 16.3 for an earlier discussion.

It turns out that the answer to this question is very different dependent on the underlying relationship between $\mathbf{x}$ and $\mathbf{y}$. Put differently, the question is whether $P(\mathbf{y} \mid \mathbf{x})$, seen as a function of $\mathbf{x}$ has anything to do with $P(\mathbf{x})$. If not, then unsupervised learning of $P(\mathbf{x})$ can be of no help to learn $P(\mathbf{y} \mid \mathbf{x})$. Consider for example the case where $P(\mathbf{x})$ is uniformly distributed and $\mathbb{E}[\mathbf{y} \mid \boldsymbol{x}]$ is some function of interest. Clearly, observing $\boldsymbol{x}$ alone gives us no information about $P(\mathbf{y} \mid \mathbf{x})$. As a better case, consider the situation where $\mathbf{x}$ arises from a mixture, with one mixture component per value of $\mathbf{y}$, as illustrated in Figure 16.9. If the mixture components are well-separated, then modeling $P(\mathbf{x})$ tells us precisely where each component is, and a single labeled example of each example will then be enough to perfectly learn $P(\mathbf{y} \mid \mathbf{x})$. But more generally, what could make $P(\mathbf{y} \mid \mathbf{x})$ and $P(\mathbf{x})$ tied together?

If $\mathbf{y}$ is closely associated with one of the causal factors of $\mathbf{x}$, then, as first argued by Janzing *et al.* (2012), $P(\mathbf{x})$ and $P(\mathbf{y} \mid \mathbf{x})$ will be strongly tied, and unsupervised representation learning that tries to disentangle the underlying factors of variation is likely to be useful as a semi-supervised learning strategy.

Consider the assumption that $\mathbf{y}$ is one of the causal factors of $\mathbf{x}$, and let $\mathbf{h}$

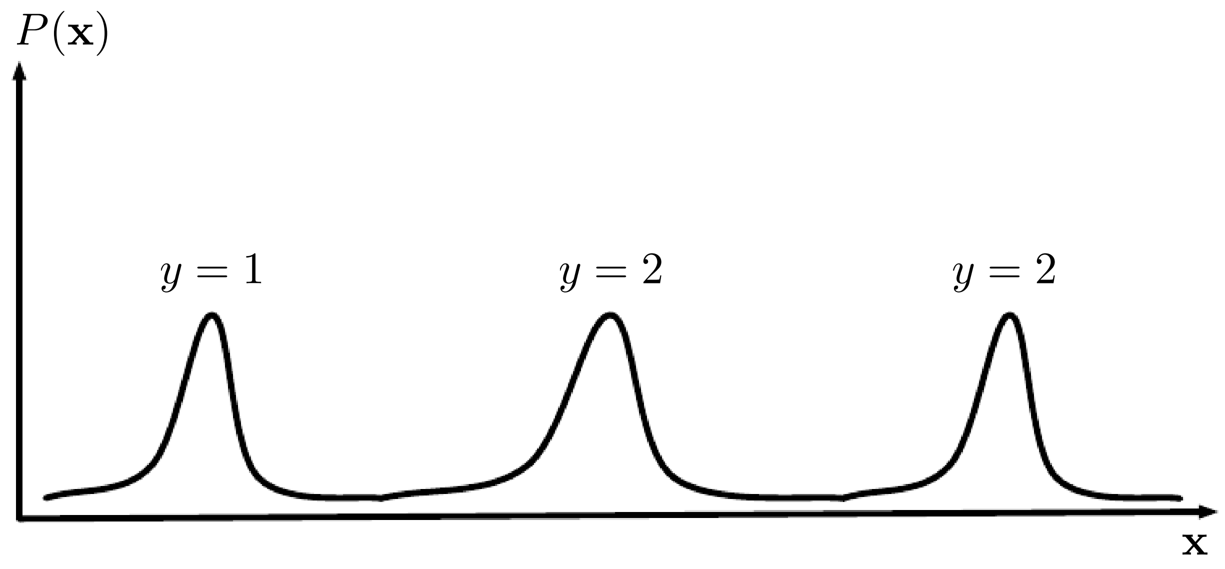


Figure 16.9: Example of a density over $\boldsymbol{x}$ that is a mixture over three components. The component identity is an underlying explanatory factor, y. Because the mixture components (e.g., natural object classes in image data) are statistically salient, just modeling $P(\boldsymbol{x})$ in an unsupervised way with no labeled example already reveals the factor y.

represent all those factors. Then the true generative process can be conceived as structured according to this directed graphical model, with $\mathbf{h}$ as the parent of $\mathbf{x}$:

$$P(\mathbf{h}, \mathbf{x}) = P(\mathbf{x} \mid \mathbf{h}) P(\mathbf{h}).$$

As a consequence, the data has marginal probability

$$P(\boldsymbol{x}) = \int P(\boldsymbol{x} \mid \boldsymbol{h}) p(\boldsymbol{h}) d\boldsymbol{h}$$

or, in the discrete case (like in the mixture example above):

$$P(\boldsymbol{x}) = \sum_{\boldsymbol{h}} P(\boldsymbol{x} \mid \boldsymbol{h}) P(\boldsymbol{h}).$$

From this straightforward observation, we conclude that the best possible model of $\mathbf{x}$ (from a generalization point of view) is the one that uncovers the above "true" structure, with $\boldsymbol{h}$ as a latent variable that explains the observed variations in $\boldsymbol{x}$. The "ideal" representation learning discussed above should thus recover these latent factors. If $\mathbf{y}$ is one of them (or closely related to one of them), then it will be very easy to learn to predict $\mathbf{y}$ from such a representation. We also see that the conditional distribution of $\mathbf{y}$ given $\mathbf{x}$ is tied by Bayes rule to the components in the above equation:

$$P(\mathbf{y} \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \mathbf{y}) P(\mathbf{y})}{P(\mathbf{x})}.$$

Thus the marginal $P(\mathbf{x})$ is intimately tied to the conditional $P(\mathbf{y} \mid \mathbf{x})$ and knowledge of the structure of the former should be helpful to learn the latter, i.e., semi-supervised learning works. Furthermore, not knowing which of the factors in $\mathbf{h}$ will be the one of the interest, say $\mathbf{y} = \mathbf{h}_i$, an unsupervised learner should learn a representation that disentangles all the generative factors $\mathbf{h}_j$ from each other, then making it easy to predict $\mathbf{y}$ from $\mathbf{h}$.

In addition, as pointed out by Janzing *et al.* (2012), if the true generative process has $\mathbf{x}$ as an effect and $\mathbf{y}$ as a cause, then modeling $P(\mathbf{x} \mid \mathbf{y})$ is robust to changes in $P(\mathbf{y})$. If the cause-effect relationship was reversed, it would not be true, since by Bayes rule, $P(\mathbf{x} \mid \mathbf{y})$ would be sensitive to changes in $P(\mathbf{y})$. Very often, when we consider changes in distribution due to different domains, temporal non-stationarity, or changes in the nature of the task, *the causal mechanisms remain invariant* ("the laws of the universe are constant") whereas what changes are the marginal distribution over the underlying causes (or what factors are linked to our particular task). Hence, better generalization and robustness to all kinds of changes can be expected via learning a generative model that attempts to recover the causal factors $\mathbf{h}$ and $P(\mathbf{x} \mid \mathbf{h})$.

16.5 Assumption of Underlying Factors and Distributed Representation

A very basic notion that comes out of the above discussion and of the notion of "disentangled factors" is the very idea that there are underlying factors that generate the observed data. It is a core assumption behind most neural network and deep learning research, more precisely relying on the notion of *distributed representation*.

What we call a distributed representation is one which can express an exponentially large number of concepts by allowing to compose the activation of many features. An example of distributed representation is a vector of n binary features, which can take 2^n configurations, each potentially corresponding to a different region in input space. This can be compared with a *symbolic representation*, where the input is associated with a single symbol or category. If there are n symbols in the dictionary, one can imagine n feature detectors, each corresponding to the detection of the presence of the associated category. In that case only n different configurations of the representation-space are possible, carving n different regions in input space. Such a symbolic representation is also called a one-hot representation, since it can be captured by a binary vector with n bits that are mutually exclusive (only one of them can be active). These ideas are developed further in the next section.

Examples of learning algorithms based on non-distributed representations include:

- Clustering methods, including the k-means algorithm: only one cluster "wins" the competition.
- k-nearest neighbors algorithms: only one template or prototype example is associated with a given input.
- Decision trees: only one leaf (and the nodes on the path from root to leaf) is activated when an input is given.
- Gaussian mixtures and mixtures of experts: the templates (cluster centers) or experts are now associated with a *degree* of activation, which makes the posterior probability of components (or experts) given input look more like a distributed representation. However, as discussed in the next section, these models still suffer from a poor statistical scaling behavior compared to those based on distributed representations (such as products of experts and RBMs).
- Kernel machines with a Gaussian kernel (or other similarly local kernel): although the degree of activtion of each "support vector" or template example is now continuous-valued, the same issue arises as with Gaussian mixtures.
- Language or translation models based on n-grams: the set of contexts (sequences of symbols) is partitioned according to a tree structure of suffixes (e.g. a leaf may correspond to the last two words being w_1 and w_2), and separate parameters are estimated for each leaf of the tree (with some sharing being possible of parameters associated with internal nodes, between the leaves of the sub-tree rooted at the same internal node).

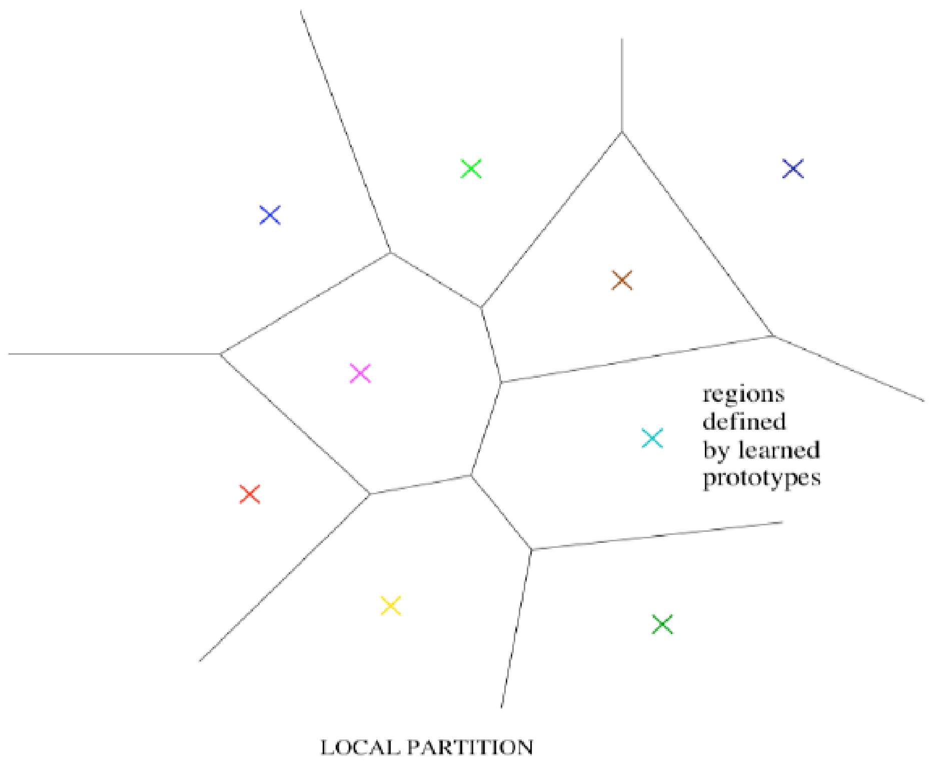


Figure 16.10: Illustration of how a learning algorithm based on a non-distributed representation breaks up the input space into regions, *with a separate set of parameters for each region.* For example, a clustering algorithm or a 1-nearest-neighbor algorithm associates one template (colored X) to each region. This is also true of decision trees, mixture models, and kernel machines with a local (e.g., Gaussian) kernel. In the latter algorithms, the output is not constant by parts but instead interpolates between neighboring regions, but the relationship between the number of parameters (or examples) and the number of regions they can define remains linear. The advantage is that a different answer (e.g., density function, predicted output, etc.) can be *independently* chosen for each region. The disadvantage is that there is no generalization to new regions, except by extending the answer for which there is data, exploiting solely a *smoothness prior*. It makes it difficult to learn a complicated function, with more ups and downs than the available number of examples. Contrast this with a distributed representation, Figure 16.11.

An important related concept that distinguishes a distributed representation from a symbolic one is that *generalization arises due to shared attributes* between different concepts. As pure symbols, “ tt cat” and “`dog`” are as far from each other as any other two symbols. However, if one associates them with a meaningful distributed representation, then many of the things that can be said about cats can generalize to dogs and vice-versa. This is what allows neural language models to generalize so well (Section 12.4). 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. Of course, one would get a distributed representation if one would associated *multiple symbolic attributes* to each symbol.

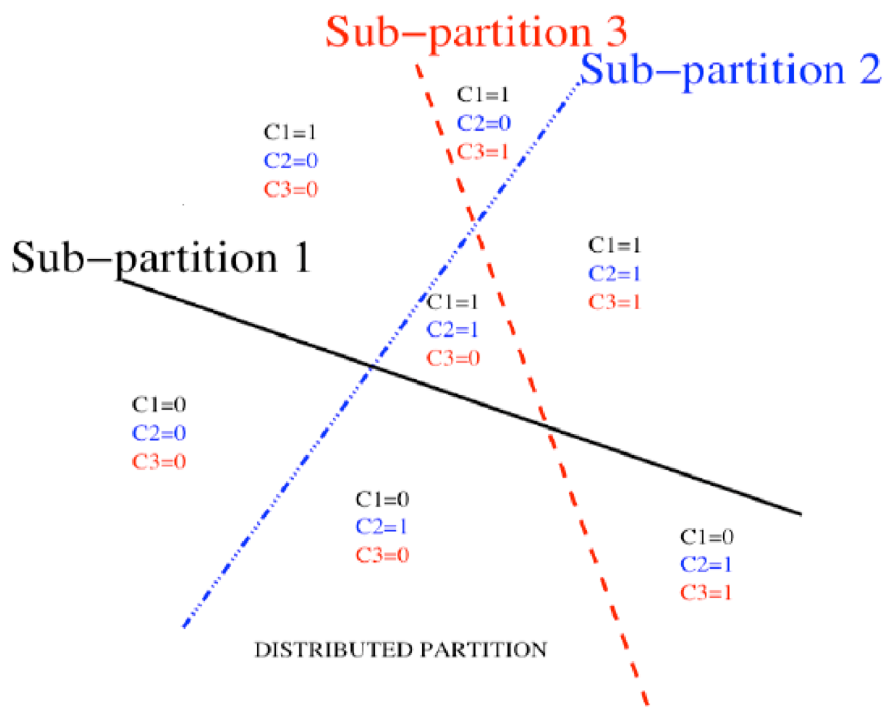


Figure 16.11: Illustration of how a learning algorithm based on a distributed representation breaks up the input space into regions, *with exponentially more regions than parameters*. Instead of a single partition (as in the non-distributed case, Figure 16.10), we have many partitions, one per "feature", and all their possible intersections. In the example of the figure, there are 3 binary features $C1$, $C2$, and $C3$, each corresponding to partitioning the input space in two regions according to a hyperplane, i.e., each is a linear classifier. Each possible intersection of these half-planes forms a region, i.e., each region corresponds to a configuration of the bits specifying whether each feature is 0 or 1, on which side of their hyperplane is the input falling. If the input space is large enough, the number of regions grows exponentially with the number of features, i.e., of parameters. However, the way these regions carve the input space still depends on few parameters: this huge number of regions are not placed independently of each other. We can thus represent a function that *looks complicated* but actually has structure. Basically, the assumption is that one can learn about each feature without having to see the examples for all the configurations of all the other features, i.e., these features corespond to underlying factors explaining the data.

Note that a *sparse representation* is a distributed representation where the number of attributes that are active together is small compared to the total number of attributes. For example, in the case of binary representations, one might have only $k \ll n$ of the n bits that are non-zero. The power of representation grows exponentially with the number of active attributes, e.g., $O(n^k)$ in the above example of binary vectors. At the extreme, a symbolic representation is a very sparse representation where only one attribute at a time can be active.

16.6 Exponential Gain in Representational Efficiency from Distributed Representations

When and why can there be a statistical advantage from using a distributed representation as part of a learning algorithm?

Figures 16.10 and 16.11 explain that advantage in intuitive terms. The argument is that a function that "looks complicated" can be compactly represented using a small number of parameters, if some "structure" is uncovered by the learner. Traditional "non-distributed" learning algorithms generalize only due to the smoothness assumption, which states that if $u \approx v$, then the target function f to be learned has the property that $f(u) \approx f(v)$, in general. There are many ways of formalizing such an assumption, but the end result is that if we have an example (x, y) for which we know that $f(x) \approx y$, then we choose an estimator $\hat{f}$ that approximately satisfies these constraints while changing as little as possible. This assumption is clearly very useful, but it suffers from the curse of dimensionality: in order to learn a target function that takes many different values (e.g. many ups and downs) in a large number of regions[2] , we may need a number of examples that is at least as large as the number of distinguishible regions. One can think of each of these regions as a category or symbol: by having a separate degree of freedom for each symbol (or region), we can learn an arbitrary mapping from symbol to value. However, this does not allow us to generalize to new symbols, new regions.

If we are lucky, there may be some regularity in the target function, besides being smooth. For example, the same pattern of variation may repeat itself many times (e.g., as in a periodic function or a checkerboard). If we only use the smoothness prior, we will need additional examples for each repetition of that pattern. However, as discussed by Montufar *et al.* (2014), a deep architecture could represent and discover such a repetition pattern and generalize to new instances of it. Thus a small number of parameters (and therefore, a small number of examples) could suffice to represent a function that looks complicated (in the sense that it would be expensive to represent with a non-distributed architecture). Figure 16.11 shows a simple example, where we have n binary features

[2]e.g., exponentially many regions: in a d-dimensional space with at least 2 different values to distinguish per dimension, we might want f to differ in 2^d different regions, requiring $O(2^d)$ training examples.

in a d-dimensional space, and where each binary feature corresponds to a linear classifier that splits the input space in two parts. The exponentially large number of intersections of n of the corresponding half-spaces corresponds to as many distinguishable regions that a distributed representation learner could capture. How many regions are generated by an arrangement of n hyperplanes in $\mathbb{R}^d$? This corresponds to the number of regions that a shallow neural network (one hidden layer) can distinguish (Pascanu *et al.*, 2014b), which is

$$\sum_{j=0}^{d} \binom{n}{j} = O(n^d),$$

following a more general result from Zaslavsky (1975), known as Zaslavsky's theorem, one of the central results from the theory of hyperplane arrangements. Therefore, we see a growth that is exponential in the input size and polynomial in the number of hidden units.

Although a distributed representation (e.g. a shallow neural net) can represent a richer function with a smaller number of parameters, there is no free lunch: to construct an *arbitrary* partition (say with 2^d different regions) one will need a correspondingly large number of hidden units, i.e., of parameters and of examples. The use of a distributed representation therefore also corresponds to a prior, which comes on top of the smoothness prior. To return to the hyperplanes examples of Figure 16.11, we see that we are able to get this generalization because we can learn about the location of each hyperplane with only $O(d)$ examples: we do not need to see examples corresponding to all $O(n^d)$ regions.

Let us consider a concrete example. Imagine that the input is the image of a person, and that we have a classifier that detects whether the person is a child or not, another that detects if that person is a male or a female, another that detects whether that person wears glasses or not, etc. Keep in mind that these features are discovered automatically, not fixed a priori. We can learn about the distinction between male and female, or about the presence or absence of glasses, without having to consider all of the configurations of the n features. This form of statistical separability is what allows one to generalize to new configurations of a person's features that have never been seen during training. It corresponds to the prior discussed above regarding the existence of multiple underlying explanatory factors. This prior is very plausible for most of the data distributions on which human intelligence would be useful, but it may not apply to every possible distribution. However, this apparently innocuous assumption buys us a lot, statistically speaking, because it allows the learner to discover structure with a reasonably small number of examples that would otherwise require exponentially more training data.

Another interesting result illustrating the statistical effect of a distributed representations versus a non-distributed one is the mathematical analysis (Montufar and Morton, 2014) of *products of mixtures* (which include the RBM as a special case) versus *mixture of products* (such as the mixture of Gaussians). The analysis shows that a mixture of products can require an exponentially larger number of parameters in order to represent the probability distributions arising out of a product of mixtures.

16.7 Exponential Gain in Representational Efficiency from Depth

In the above example with the input being an image of a person, it would not be reasonable to expect factors such as gender, age, and the presence of glasses to be detected simply from a linear classifier, i.e., a shallow neural network. The kinds of factors that can be chosen almost independently in order to generate data are more likely to be very high-level and related in highly non-linear ways to the input. This demands *deep* distributed representations, where the higher level features (seen as functions of the input) or factors (seen as generative causes) are obtained through the composition of many non-linearities.

It turns out that organizing computation through the composition of many non-linearities and a hierarchy of reused features can give another exponential boost to statistical efficiency. Although 2-layer networks (e.g., with saturating non-linearities, boolean gates, sum/products, or RBF units) can generally be shown to be universal approximators[3], the required number of hidden units may be very large. The main results on the expressive power of deep architectures state that there are families of functions that can be represented efficiently with a deep architecture (say depth k) but would require an exponential number of components (with respect to the input size) with insufficient depth (depth 2 or depth $k-1$).

More precisely, a feedforward neural network with a single hidden layer is a universal approximator (of Borel measurable functions) (Hornik *et al.*, 1989; Cybenko, 1989). Other works have investigated universal approximation of probability distributions by deep belief networks (Le Roux and Bengio, 2010; Montúfar and Ay, 2011), as well as their approximation properties (Montúfar, 2014; Krause *et al.*, 2013).

Regarding the advantage of depth, early theoretical results have focused on circuit operations (neural net unit computations) that are substantially different from those being used in real state-of-the-art deep learning applications,

[3]with enough hidden units they can approximate a large class of functions (e.g. continuous functions) up to some given tolerance level

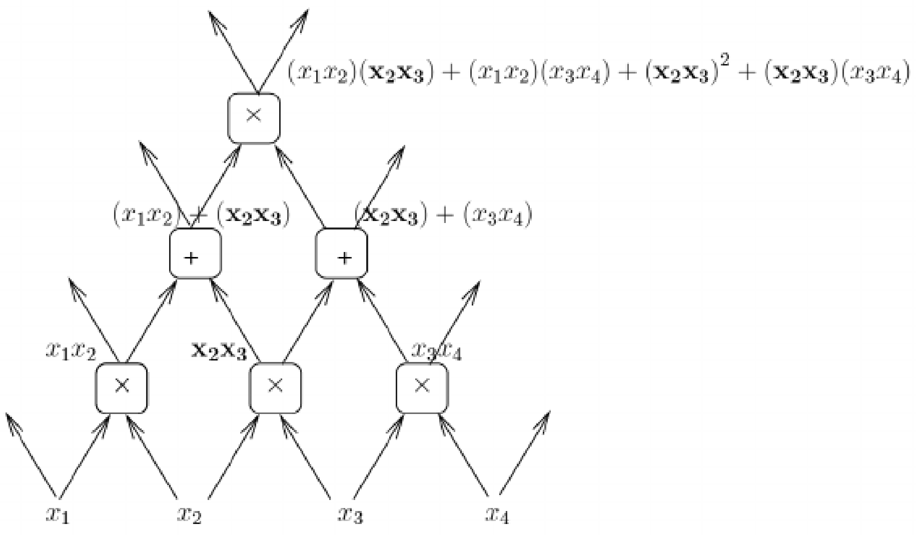

$(x_1x_2)(\mathbf{x_2x_3}) + (x_1x_2)(x_3x_4) + (\mathbf{x_2x_3})^2 + (\mathbf{x_2x_3})(x_3x_4)$
×
$(x_1x_2) + (\mathbf{x_2x_3})$
$(\mathbf{x_2x_3}) + (x_3x_4)$
+
+
x_1x_2
$\mathbf{x_2x_3}$
x_3x_4
×
×
×
x_1
x_2
x_3
x_4

Figure 16.12: A sum-product network (Poon and Domingos, 2011) composes summing units and product units, so that each node computes a polynomial. Consider the product node computing $x_2 x_3$: its value is reused in its two immediate children, and indirectly incorporated in its grand-children. In particular, in the top node shown the product x_2x_3 would arise 4 times if that node's polynomial was expanded as a sum of products. That number could double for each additional layer. In general a deep sum of product can represent polynomials with a number of min-terms that is exponential in depth, and some families of polynomials are represented efficiently with a deep sum-product network but not efficiently representable with a simple sum of products, i.e., a 2-layer network (Delalleau and Bengio, 2011).

such as logic gates (Håstad, 1986) and linear threshold units with non-negative weights (Håstad and Goldmann, 1991). More recently, Delalleau and Bengio (2011) showed that a shallow network requires exponentially many more sum-product hidden units[4] than a deep sum-product network (Poon and Domingos, 2011) in order to compute certain families of polynomials. Figure 16.12 illustrates a sum-product network for representing polynomials, and how a deeper network can be exponentially more efficient because the same computation can be reused exponentially (in depth) many times. Note however that Martens and Medabalimi (2014) showed that sum-product networks may be have limitations in their expressive power, in the sense that there are distributions that can easily be represented by other generative models but that cannot be efficiently represented under the decomposability and completeness conditions associated with the probabilistic interpretation of sum-product networks (Poon and Domingos, 2011).

Closer to the kinds of deep networks actually used in practice (Pascanu *et al.*,

[4]Here, a single sum-product hidden layer summarizes a layer of product units followed by a layer of sum units.

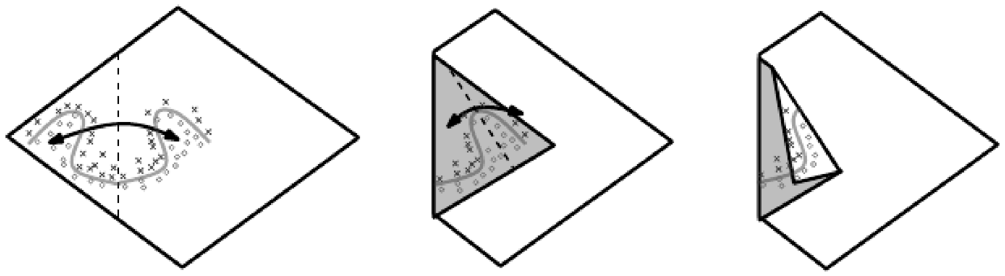


Figure 16.13: An absolute value rectification unit has the same output for every pair of mirror points in its input. The mirror axis of symmetry is given by the hyperplane defined by the weights and bias of the unit. If one considers a function computed on top of that unit (the green decision surface), it will be formed of a mirror image of a simpler pattern, across that axis of symmetry. The middle image shows how it can be obtained by folding the space around that axis of symmetry, and the right image shows how another repeating pattern can be folded on top of it (by another downstream unit) to obtain another symmetry (which is now repeated four times, with two hidden layers). This is an intuitive explanation of the exponential advantage of deeper rectifier networks formally shown in Pascanu *et al.* (2014a); Montufar *et al.* (2014).

2014a; Montufar *et al.*, 2014) showed that piecewise linear networks (e.g. obtained from rectifier non-linearities or maxout units) could represent functions with exponentially more piecewise-linear regions, as a function of depth, compared to shallow neural networks. Figure 16.13 illustrates how a network with absolute value rectification creates mirror images of the function computed on top of some hidden unit, with respect to the input of that hidden unit. Each hidden unit specifies where to fold the input space in order to create mirror responses (on both sides of the absolute value non-linearity). By composing these folding operations, we obtain an exponentially large number of piecewise linear regions which can capture all kinds of regular (e.g. repeating) patterns.

More precisely, the main theorem in Montufar *et al.* (2014) states that the number of linear regions carved out by a deep rectifier network with d inputs, depth L, and n units per hidden layer, is

$$O\left(\binom{n}{d}^{d(L-1)} n^{d}\right),$$

i.e., exponential in the depth L. In the case of maxout networks with k filters per unit, the number of linear regions is

$$O\left(k^{(L-1)+d}\right).$$

16.8 Priors regarding the Underlying Factors

To close this chapter, we come back to the original question: what is a good representation? We proposed that an ideal representation is one that disentangles the underlying causal factors of variation that generated the data, especially those factors that we care about in our applications. It seems clear that if we have direct clues about these factors (like if a factor $\mathbf{y} = \mathbf{h}_i$, a label, is observed at the same time as an input $\mathbf{x}$), then this can help the learner separate these observed factors from the others. This is already what supervised learning does. But in general, we may have a lot more unlabeled data than labeled data: can we use other clues, other hints about the underlying factors, in order to disentangle them more easily?

What we propose here is that indeed we can provide all kinds of broad priors which are as many hints that can help the learner discover, identify and disentangle these factors. The list of such priors is clearly not exhaustive, but it is a starting point, and yet most learning algorithms in the machine learning literature only exploit a small subset of these priors. With absolutely no priors, we know that it is not possible to generalize: this is the essence of the *no-free-lunch theorem for machine learning*. In the space of all functions, which is huge, with any finite training set, there is no general-purpose learning recipe that would dominate all other learning algorithms. Whereas some assumptions are required, when our goal is to build AI or understand human intelligence, it is tempting to focus our attention on the most general and broad priors, that are relevant for most of the tasks that humans are able to successfully learn.

This list was introduced in section 3.1 of Bengio *et al.* (2013c).

- **Smoothness**: we want to learn functions f s.t. $x \approx y$ generally implies $f(x) \approx f(y)$. This is the most basic prior and is present in most machine learning, but is insufficient to get around the curse of dimensionality, as discussed abov and in Bengio *et al.* (2013c). below.

- **Multiple explanatory factors**: the data generating distribution is generated by different underlying factors, and for the most part what one learns about one factor generalizes in many configurations of the other factors. This assumption is behind the idea of **distributed representations**, discussed in Section 16.5 above.

- **Depth**, or **a hierarchical organization of explanatory factors**: the concepts that are useful at describing the world around us can be defined in terms of other concepts, in a hierarchy, with more **abstract** concepts higher in the hierarchy, being defined in terms of less abstract ones. This is the assumption exploited by having **deep representations**.

- **Causal factors**: the input variables $\mathbf{x}$ are consequences, effects, while the explanatory factors are causes, and not vice-versa. As discussed above, this enables the **semi-supervised learning** assumption, i.e., that $P(\mathbf{x})$ is tied to $P(\mathbf{y} \mid \mathbf{x})$, making it possible to improve the learning of $P(\mathbf{y} \mid \mathbf{x})$ via the learning of $P(\mathbf{x})$. More precisely, this entails that representations that are useful for $P(\mathbf{x})$ are useful when learning $P(\mathbf{y} \mid \mathbf{x})$, allowing sharing of statistical strength between the unsupervised and supervised learning tasks.

- **Shared factors across tasks**: in the context where we have many tasks, corresponding to different $\mathbf{y}_i$'s sharing the same input $\mathbf{x}$ or where each task is associated with a subset or a function $f_i(\mathbf{x})$ of a global input $\mathbf{x}$, the assumption is that each $\mathbf{y}_i$ is associated with a different subset from a common pool of relevant factors $\mathbf{h}$. Because these subsets overlap, learning all the $P(\mathbf{y}_i \mid \mathbf{x})$ via a shared intermediate representation $P(\mathbf{h} \mid \mathbf{x})$ allows sharing of statistical strength between the tasks.

- **Manifolds**: probability mass concentrates, and the regions in which it concentrates are locally connected and occupy a tiny volume. In the continuous case, these regions can be approximated by low-dimensional manifolds that a much smaller dimensionality than the original space where the data lives. This is the manifold hypothesis and is covered in Chapter 17, especially with algorithms related to auto-encoders.

- **Natural clustering**: different values of categorical variables such as object classes[5] are associated with separate manifolds. More precisely, the local variations on the manifold tend to preserve the value of a category, and a linear interpolation between examples of different classes in general involves going through a low density region, i.e., $P(\mathbf{x} \mid \mathrm{y} = i)$ for different i tend to be well-separated and not overlap much. For example, this is exploited explicitly in the Manifold Tangent Classifier discussed in Section 17.5. This hypothesis is consistent with the idea that humans have *named* categories and classes because of such statistical structure (discovered by their brain and propagated by their culture), and machine learning tasks often involves predicting such categorical variables.

- **Temporal and spatial coherence**: this is similar to the cluster assumption but concerns sequences or tuples of observations; consecutive or spatially nearby observations tend to be associated with the same value of relevant categorical concepts, or result in a small move on the surface of the high-density manifold. More generally, different factors change at different temporal and spatial scales, and many categorical concepts of inter-

[5]it is often the case that the y of interest is a category

est change slowly. When attempting to capture such categorical variables, this prior can be enforced by making the associated representations slowly changing, i.e., penalizing changes in values over time or space. This prior was introduced in Becker and Hinton (1992).

- **Sparsity**: for any given observation x, only a small fraction of the possible factors are relevant. In terms of representation, this could be represented by features that are often zero (as initially proposed by Olshausen and Field (1996)), or by the fact that most of the extracted features are *insensitive* to small variations of $\mathbf{x}$. This can be achieved with certain forms of priors on latent variables (peaked at 0), or by using a non-linearity whose value is often flat at 0 (i.e., 0 and with a 0 derivative), or simply by penalizing the magnitude of the Jacobian matrix (of derivatives) of the function mapping input to representation. This is discussed in Section 15.8.

- **Simplicity of Factor Dependencies**: in good high-level representations, the factors are related to each other through simple dependencies. The simplest possible is marginal independence, $P(\mathbf{h}) = \prod_i P(\mathbf{h}_i)$, but linear dependencies or those captured by a shallow auto-encoder are also reasonable assumptions. This can be seen in many laws of physics, and is assumed when plugging a linear predictor or a factorized prior on top of a learned representation.