Chapter 15

Representation Learning

In this chapter, we first discuss what it means to learn representations and how the notion of representation can be useful to design deep architectures. We explore how learning algorithms share statistical strength across different tasks, including using information from unsupervised tasks to perform supervised tasks. Shared representations are useful to handle multiple modalities or domains, or to transfer learned knowledge to tasks for which few or no examples are given but a task representation exists. Finally, we step back and argue about the reasons for the success of representation learning, starting with the theoretical advantages of distributed representations (Hinton *et al.*, 1986) and deep representations, ending with the more general idea of underlying assumptions about the data-generating process, in particular about underlying causes of the observed data.

Many information processing tasks can be very easy or very difficult depending on how the information is represented. This is a general principle applicable to daily life, to computer science in general, and to machine learning. For example, it is straightforward for a person to divide 210 by 6 using long division. The task becomes considerably less straightforward if it is instead posed using the Roman numeral representation of the numbers. Most modern people asked to divide CCX by VI would begin by converting the numbers to the Arabic numeral representation, permitting long division procedures that make use of the place value system. More concretely, we can quantify the asymptotic runtime of various operations using appropriate or inappropriate representations. For example, inserting a number into the correct position in a sorted list of numbers is an $O(n)$ operation if the list is represented as a linked list, but only $O(\log n)$ if the list is represented as a red-black tree.

In the context of machine learning, what makes one representation better than another? Generally speaking, a good representation is one that makes a subsequent learning task easier. The choice of representation will usually depend on the choice of the subsequent learning task.

We can think of feedforward networks trained by supervised learning as performing a kind of representation learning. Specifically, the last layer of the network is typically a linear classifier, such as a softmax regression classifier. The rest of the network learns to provide a representation to this classifier. Training with a supervised criterion naturally leads to the representation at every hidden layer (but more so near the top hidden layer) taking on properties that make the classification task easier. For example, classes that were not linearly separable in the input features may become linearly separable in the last hidden layer. In principle, the last layer could be another kind of model, such as a nearest neighbor classifier (Salakhutdinov and Hinton, 2007a). The features in the penultimate layer should learn different properties depending on the type of the last layer.

Supervised training of feedforward networks does not involve explicitly imposing any condition on the learned intermediate features. Other kinds of representation learning algorithms are often explicitly designed to shape the representation in some particular way. For example, suppose we want to learn a representation that makes density estimation easier. Distributions with more independences are easier to model, so we could design an objective function that encourages the elements of the representation vector $\boldsymbol{h}$ to be independent. Just like supervised networks, unsupervised deep learning algorithms have a main training objective but also learn a representation as a side effect. Regardless of how a representation was obtained, it can be used for another task. Alternatively, multiple tasks (some supervised, some unsupervised) can be learned together with some shared internal representation.

Most representation learning problems face a trade-off between preserving as much information about the input as possible and attaining nice properties (such as independence).

Representation learning is particularly interesting because it provides one way to perform unsupervised and semi-supervised learning. We often have very large amounts of unlabeled training data and relatively little labeled training data. Training with supervised learning techniques on the labeled subset often results in severe overfitting. Semi-supervised learning offers the chance to resolve this overfitting problem by also learning from the unlabeled data. Specifically, we can learn good representations for the unlabeled data, and then use these representations to solve the supervised learning task.

Humans and animals are able to learn from very few labeled examples. We do not yet know how this is possible. Many factors could explain improved human performance—for example, the brain may use very large ensembles of classifiers or Bayesian inference techniques. One popular hypothesis is that the brain is able to leverage unsupervised or semi-supervised learning. There are many ways to leverage unlabeled data. In this chapter, we focus on the hypothesis that the unlabeled data can be used to learn a good representation.

15.1 Greedy Layer-Wise Unsupervised Pretraining

Unsupervised learning played a key historical role in the revival of deep neural networks, enabling researchers for the first time to train a deep supervised network without requiring architectural specializations like convolution or recurrence. We call this procedure **unsupervised pretraining**, or more precisely, **greedy layer-wise unsupervised pretraining**. This procedure is a canonical example of how a representation learned for one task (unsupervised learning, trying to capture the shape of the input distribution) can sometimes be useful for another task (supervised learning with the same input domain).

Greedy layer-wise unsupervised pretraining relies on a single-layer representation learning algorithm such as an RBM, a single-layer autoencoder, a sparse coding model, or another model that learns latent representations. Each layer is pretrained using 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. See algorithm 15.1 for a formal description.

Greedy layer-wise training procedures based on unsupervised criteria have long been used to sidestep the difficulty of jointly training the layers of a deep neural net for a supervised task. 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 over all the layers, and that this approach could be used to successfully train even fully connected architectures (Hinton *et al.*, 2006; Hinton and Salakhutdinov, 2006; Hinton, 2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007a). Prior to this discovery, only convolutional deep networks or networks whose depth resulted from recurrence were regarded as feasible to train. Today, we now know that greedy layer-wise pretraining is not required to train fully connected deep architectures, but the unsupervised pretraining approach was the first method to succeed.

Greedy layer-wise pretraining is called **greedy** because it is a **greedy algo-**

rithm, meaning that it optimizes each piece of the solution independently, one piece at a time, rather than jointly optimizing all pieces. It is called **layer-wise** because these independent pieces are the layers of the network. Specifically, greedy layer-wise pretraining proceeds one layer at a time, training the k-th layer while keeping the previous ones fixed. In particular, the lower layers (which are trained first) are not adapted after the upper layers are introduced. It is called **unsupervised** because each layer is trained with an unsupervised representation learning algorithm. However, it is also called **pretraining** because it is supposed to be only a first step before a joint training algorithm is applied to **fine-tune** all the layers together. In the context of a supervised learning task, it can be viewed as a regularizer (in some experiments, pretraining decreases test error without decreasing training error) and a form of parameter initialization.

It is common to use the word "pretraining" to refer not only to the pretraining stage itself but to the entire two-phase protocol that combines the pretraining phase and a supervised learning phase. The supervised learning phase may involve training a simple classifier on top of the features learned in the pretraining phase, or it may involve supervised fine-tuning of the entire network learned in the pretraining phase. No matter what kind of unsupervised learning algorithm or what model type is employed, in most cases, the overall training scheme is nearly the same. While the choice of unsupervised learning algorithm will obviously affect the details, most applications of unsupervised pretraining follow this basic protocol.

Greedy layer-wise unsupervised pretraining can also be used as initialization for other unsupervised learning algorithms, such as deep autoencoders (Hinton and Salakhutdinov, 2006) and probabilistic models with many layers of latent variables. Such models include deep belief networks (Hinton *et al.*, 2006) and deep Boltzmann machines (Salakhutdinov and Hinton, 2009a). These deep generative models are described in chapter 20.

As discussed in section 8.7.4, it is also possible to have greedy layer-wise *supervised* pretraining. 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).

15.1.1 When and Why Does Unsupervised Pretraining Work?

On many tasks, greedy layer-wise unsupervised pretraining can yield substantial improvements in test error for classification tasks. This observation was responsible for the renewed interested in deep neural networks starting in 2006 (Hinton *et al.*,

Algorithm 15.1 *Greedy layer-wise unsupervised pretraining protocol*
Given the following: Unsupervised feature learning algorithm $\mathcal{L}$, which takes a training set of examples and returns an encoder or feature function f. The raw input data is $\boldsymbol{X}$, with one row per example, and $f^{(1)}(\boldsymbol{X})$ is the output of the first stage encoder on $\boldsymbol{X}$. In the case where 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.

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

2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007a). On many other tasks, however, unsupervised pretraining either does not confer a benefit or even causes noticeable harm. Ma *et al.* (2015) studied the effect of pretraining on machine learning models for chemical activity prediction and found that, on average, pretraining was slightly harmful, but for many tasks was significantly helpful. Because unsupervised pretraining is sometimes helpful but often harmful, it is important to understand when and why it works in order to determine whether it is applicable to a particular task.

At the outset, it is important to clarify that most of this discussion is restricted to greedy unsupervised pretraining in particular. There are other, completely different paradigms for performing semi-supervised learning with neural networks, such as virtual adversarial training, described in section 7.13. It is also possible to train an autoencoder or generative model at the same time as the supervised model. Examples of this single-stage approach include the discriminative RBM (Larochelle and Bengio, 2008) and the ladder network (Rasmus *et al.*, 2015), in which the total objective is an explicit sum of the two terms (one using the labels, and one only using the input).

Unsupervised pretraining combines two different ideas. First, it makes use of the idea that the choice of initial parameters for a deep neural network can have a significant regularizing effect on the model (and, to a lesser extent, that it can improve optimization). Second, it makes use of the more general idea that learning about the input distribution can help with learning about the mapping from inputs to outputs.

Both ideas involve many complicated interactions between several parts of the machine learning algorithm that are not entirely understood.

The first idea, that the choice of initial parameters for a deep neural network can have a strong regularizing effect on its performance, is the least understood. At the time that pretraining became popular, it was understood as initializing the model in a location that would cause it to approach one local minimum rather than another. Today, local minima are no longer considered to be a serious problem for neural network optimization. We now know that our standard neural network training procedures usually do not arrive at a critical point of any kind. It remains possible that pretraining initializes the model in a location that would otherwise be inaccessible—for example, a region that is surrounded by areas where the cost function varies so much from one example to another that minibatches give only a very noisy estimate of the gradient, or a region surrounded by areas where the Hessian matrix is so poorly conditioned that gradient descent methods must use very small steps. However, our ability to characterize exactly what aspects of the pretrained parameters are retained during the supervised training stage is limited. This is one reason that modern approaches typically use simultaneous unsupervised learning and supervised learning rather than two sequential stages. One may also avoid struggling with these complicated ideas about how optimization in the supervised learning stage preserves information from the unsupervised learning stage by simply freezing the parameters for the feature extractors and using supervised learning only to add a classifier on top of the learned features.

The other idea, that a learning algorithm can use information learned in the unsupervised phase to perform better in the supervised learning stage, is better understood. The basic idea is that some features that are useful for the unsupervised task may also be useful for the supervised learning task. For example, if we train a generative model of images of cars and motorcycles, it will need to know about wheels, and about how many wheels should be in an image. If we are fortunate, the representation of the wheels will take on a form that is easy for the supervised learner to access. This is not yet understood at a mathematical, theoretical level, so it is not always possible to predict which tasks will benefit from unsupervised learning in this way. Many aspects of this approach are highly dependent on the specific models used. For example, if we wish to add a linear classifier on top of pretrained features, the features must make the underlying classes linearly separable. These properties often occur naturally but do not always do so. This is another reason that simultaneous supervised and unsupervised learning can be preferable—the constraints imposed by the output layer are naturally included from the start.

From the point of view of unsupervised pretraining as learning a representation, we can expect unsupervised pretraining to be more effective when the initial representation is poor. One key example of this is the use of word embeddings. Words represented by one-hot vectors are not very informative because every two distinct one-hot vectors are the same distance from each other (squared L^2 distance of 2). Learned word embeddings naturally encode similarity between words by their distance from each other. Because of this, unsupervised pretraining is especially useful when processing words. It is less useful when processing images, perhaps because images already lie in a rich vector space where distances provide a low-quality similarity metric.

From the point of view of unsupervised pretraining as a regularizer, we can expect unsupervised pretraining to be most helpful when the number of labeled examples is very small. Because the source of information added by unsupervised pretraining is the unlabeled data, we may also expect unsupervised pretraining to perform best when the number of unlabeled examples is very large. The advantage of semi-supervised learning via unsupervised pretraining with many unlabeled examples and few labeled examples was made particularly clear in 2011 with unsupervised pretraining winning two international transfer learning competitions (Mesnil *et al.*, 2011; Goodfellow *et al.*, 2011), in settings where the number of labeled examples in the target task was small (from a handful to dozens of examples per class). These effects were also documented in carefully controlled experiments by Paine *et al.* (2014).

Other factors are likely to be involved. For example, unsupervised pretraining is likely to be most useful when the function to be learned is extremely complicated. Unsupervised learning differs from regularizers like weight decay because it does not bias the learner toward discovering a simple function but rather leads the learner toward discovering feature functions that are useful for the unsupervised learning task. If the true underlying functions are complicated and shaped by regularities of the input distribution, unsupervised learning can be a more appropriate regularizer.

These caveats aside, we now analyze some success cases where unsupervised pretraining is known to cause an improvement and explain what is known about why this improvement occurs. Unsupervised pretraining has usually been used to improve classifiers and is usually most interesting from the point of view of reducing test set error. Unsupervised pretraining can help tasks other than classification, however, and can act to improve optimization rather than being merely a regularizer. For example, it can improve both train and test reconstruction error for deep autoencoders (Hinton and Salakhutdinov, 2006).

Erhan *et al.* (2010) performed many experiments to explain several successes of unsupervised pretraining. Improvements to training error and improvements to test error may both be explained in terms of unsupervised pretraining taking the parameters into a region that would otherwise be inaccessible. Neural network training is nondeterministic and converges to a different function every time it is run. Training may halt at a point where the gradient becomes small, a point where early stopping ends training to prevent overfitting, or at a point where the gradient is large, but it is difficult to find a downhill step because of problems such as stochasticity or poor conditioning of the Hessian. Neural networks that receive unsupervised pretraining consistently halt in the same region of function space, while neural networks without pretraining consistently halt in another region. See figure 15.1 for a visualization of this phenomenon. The region where pretrained networks arrive is smaller, suggesting that pretraining reduces the variance of the estimation process, which can in turn reduce the risk of severe over fitting. In other words, unsupervised pretraining initializes neural network parameters into a region that they do not escape, and the results following this initialization are more consistent and less likely to be very bad than without this initialization.

Erhan *et al.* (2010) also provide some answers to *when* pretraining works best—the mean and variance of the test error were most reduced by pretraining for deeper networks. Keep in mind that these experiments were performed before the invention and popularization of modern techniques for training very deep networks (rectified linear units, dropout and batch normalization) so less is known about the effect of unsupervised pretraining in conjunction with contemporary approaches.

An important question is how unsupervised pretraining can act as a regularizer. One hypothesis is that pretraining encourages the learning algorithm to discover features that relate to the underlying causes that generate the observed data. This is an important idea motivating many other algorithms besides unsupervised pretraining and is described further in section 15.3.

Compared to other forms of unsupervised learning, unsupervised pretraining has the disadvantage of operating with two separate training phases. Many regularization strategies have the advantage of allowing the user to control the strength of the regularization by adjusting the value of a single hyperparameter. Unsupervised pretraining does not offer a clear way to adjust the strength of the regularization arising from the unsupervised stage. Instead, there are very

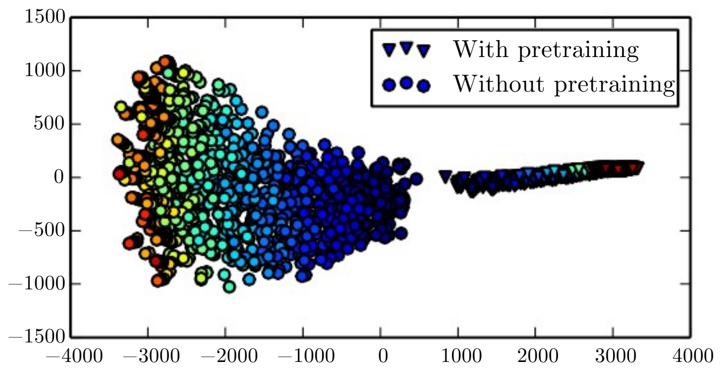

Figure 15.1: Visualization via nonlinear projection of the learning trajectories of different neural networks in *function space* (not parameter space, to avoid the issue of many-to-one mappings from parameter vectors to functions), with different random initializations and with or without unsupervised pretraining. Each point corresponds to a different neural network at a particular time during its training process. This figure is adapted with permission from Erhan *et al.* (2010). A coordinate in function space is an infinite-dimensional vector associating every input $\boldsymbol{x}$ with an output $\boldsymbol{y}$. Erhan *et al.* (2010) made a linear projection to high-dimensional space by concatenating the $\boldsymbol{y}$ for many specific $\boldsymbol{x}$ points. They then made a further nonlinear projection to 2-D by Isomap (Tenenbaum *et al.*, 2000). Color indicates time. All networks are initialized near the center of the plot (corresponding to the region of functions that produce approximately uniform distributions over the class y for most inputs). Over time, learning moves the function outward, to points that make strong predictions. Training consistently terminates in one region when using pretraining and in another, nonoverlapping region when not using pretraining. Isomap tries to preserve global relative distances (and hence volumes) so the small region corresponding to pretrained models may indicate that the pretraining-based estimator has reduced variance.

many hyperparameters, whose effect may be measured after the fact but is often difficult to predict ahead of time. When we perform unsupervised and supervised learning simultaneously, instead of using the pretraining strategy, there is a single hyperparameter, usually a coefficient attached to the unsupervised cost, that determines how strongly the unsupervised objective will regularize the supervised model. One can always predictably obtain less regularization by decreasing this coefficient. In unsupervised pretraining, there is not a way of flexibly adapting the strength of the regularization—either the supervised model is initialized to pretrained parameters, or it is not.

Another disadvantage of having two separate training phases is that each phase has its own hyperparameters. The performance of the second phase usually cannot be predicted during the first phase, so there is a long delay between proposing hyperparameters for the first phase and being able to update them using feedback from the second phase. The most principled approach is to use validation set error in the supervised phase to select the hyperparameters of the pretraining phase, as discussed in Larochelle *et al.* (2009). In practice, some hyperparameters, like the number of pretraining iterations, are more conveniently set during the pretraining phase, using early stopping on the unsupervised objective, which is not ideal but is computationally much cheaper than using the supervised objective.

Today, unsupervised pretraining has been largely abandoned, except in the field of natural language processing, where the natural representation of words as one-hot vectors conveys no similarity information and where very large unlabeled sets are available. In that case, the advantage of pretraining is that one can pretrain once on a huge unlabeled set (for example with a corpus containing billions of words), learn a good representation (typically of words, but also of sentences), and then use this representation or fine-tune it for a supervised task for which the training set contains substantially fewer examples. This approach was pioneered by Collobert and Weston (2008b), Turian *et al.* (2010), and Collobert *et al.* (2011a) and remains in common use today.

Deep learning techniques based on supervised learning, regularized with dropout or batch normalization, are able to achieve human-level performance on many tasks, but only with extremely large labeled datasets. These same techniques outperform unsupervised pretraining on medium-sized datasets such as CIFAR-10 and MNIST, which have roughly 5,000 labeled examples per class. On extremely small datasets, such as the alternative splicing dataset, Bayesian methods outperform methods based on unsupervised pretraining (Srivastava, 2013). For these reasons, the popularity of unsupervised pretraining has declined. Nevertheless, unsupervised pretraining remains an important milestone in the history of deep learning research and continues to influence contemporary approaches. The idea of pretraining has been generalized to **supervised pretraining**, discussed in section 8.7.4, as a very common approach for transfer learning. Supervised pretraining for transfer learning is popular (Oquab *et al.*, 2014; Yosinski *et al.*, 2014) for use with convolutional networks pretrained on the ImageNet dataset. Practitioners publish the parameters of these trained networks for this purpose, just as pretrained word vectors are published for natural language tasks (Collobert *et al.*, 2011a; Mikolov *et al.*, 2013a).

15.2 Transfer Learning and Domain Adaptation

Transfer learning and domain adaptation refer to the situation where what has been learned in one setting (e.g., distribution P_1) is exploited to improve generalization in another setting (say, distribution P_2). This generalizes the idea presented in the previous section, where we transferred representations between an unsupervised learning task and a supervised learning task.

In **transfer learning**, the learner must perform two or more different tasks, but we assume that 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. For example, we may learn about one set of visual categories, such as cats and dogs, in the first setting, then learn about a different set of visual categories, such as ants and wasps, in the second setting. If there is significantly more data in the first setting (sampled from P_1), then that may help to learn representations that are useful to quickly generalize from only very few examples drawn from P_2. Many visual categories *share* low-level notions of edges and visual shapes, the effects of geometric changes, changes in lighting, and so on. In general, transfer learning, multitask learning (section 7.7), and domain adaptation can be achieved via representation learning when there exist features that are useful for the different settings or tasks, corresponding to underlying factors that appear in more than one setting. This is illustrated in figure 7.2, with shared lower layers and task-dependent upper layers.

Sometimes, however, what is shared among the different tasks is not the semantics of the input but the semantics of the output. For example, a speech recognition system needs to produce valid sentences at the output layer, but the earlier layers near the input may need to recognize very different versions of the same phonemes or subphonemic vocalizations depending on which person is speaking. In cases like these, it makes more sense to share the upper layers

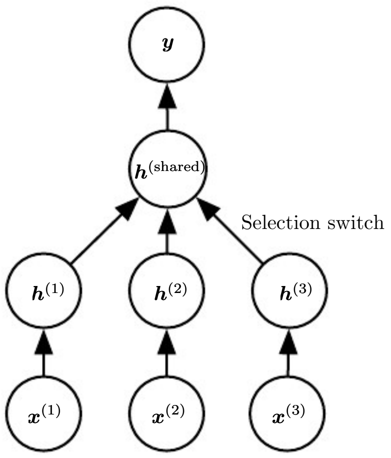

Figure 15.2: Example architecture for multitask or transfer learning when the output variable $\mathbf{y}$ has the same semantics for all tasks while the input variable $\mathbf{x}$ has a different meaning (and possibly even a different dimension) for each task (or, for example, each user), called $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$ and $\mathbf{x}^{(3)}$ for three tasks. 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.

(near the output) of the neural network and have a task-specific preprocessing, as illustrated in figure 15.2.

In the related case of **domain adaptation**, the task (and the optimal input-to-output mapping) remains the same between each setting, but the input distribution is slightly different. For example, consider the task of sentiment analysis, which consists of determining whether a comment expresses positive or negative sentiment. Comments posted on the web come from many categories. A domain adaptation scenario can arise when a sentiment predictor trained on customer reviews of media content, such as books, videos and music, is later used to analyze comments about consumer electronics, such as televisions or smartphones. One can imagine that there is an underlying function that tells whether any statement is positive, neutral, or negative, but of course the vocabulary and style may vary from one domain to another, making it more difficult to generalize across domains. Simple unsupervised pretraining (with denoising autoencoders) has been found to be very successful for sentiment analysis with domain adaptation (Glorot *et al.*, 2011b).

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 multitask learning. While the phrase "multitask learning" typically refers to supervised learning tasks, the more general notion of transfer learning is applicable to unsupervised learning and reinforcement learning as well.

In all these cases, the objective is to take advantage of data from the first setting to extract information that may be useful when learning or even when directly making predictions in the second setting. The core idea of representation learning is that the same representation may be useful in both settings. Using the same representation in both settings allows the representation to benefit from the training data that is available for both tasks.

As mentioned before, unsupervised deep learning for transfer learning has found success in some machine learning competitions (Mesnil *et al.*, 2011; Goodfellow *et al.*, 2011). 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), 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 few labeled examples. One of the most striking results found in this competition is that as an architecture makes use of deeper and deeper representations (learned in a purely unsupervised way from data collected in the first setting, P_1), the learning curve on the new categories of the second (transfer) setting P_2 becomes much better. For deep representations, fewer labeled examples of the transfer tasks are necessary to achieve the apparently asymptotic generalization performance.

Two extreme forms of transfer learning are **one-shot learning** and **zero-shot learning**, sometimes also called **zero-data learning**. Only one labeled example of the transfer task is given for one-shot learning, while no labeled examples are given at all for the zero-shot learning task.

One-shot learning (Fei-Fei *et al.*, 2006) is possible because the representation learns to cleanly separate the underlying classes during the first stage. During the transfer learning stage, only one labeled example is needed to infer the label of many possible test examples that all cluster around the same point in representation space. This works to the extent that the factors of variation corresponding to these invariances have been cleanly separated from the other factors, in the learned representation space, and that we have somehow learned which factors do and do not matter when discriminating objects of certain categories.

As an example of a zero-shot learning setting, consider the problem of having a learner read a large collection of text and then solve object recognition problems. It may be possible to recognize a specific object class even without having seen an image of that object if the text describes the object well enough. For example, having read that a cat has four legs and pointy ears, the learner might be able to guess that an image is a cat without having seen a cat before.

Zero-data learning (Larochelle *et al.*, 2008) and zero-shot learning (Palatucci *et al.*, 2009; Socher *et al.*, 2013b) are only possible because additional information has been exploited during training. We can think of the zero-data learning scenario as including three random variables: the traditional inputs $\boldsymbol{x}$, the traditional outputs or targets $\boldsymbol{y}$, and an additional random variable describing the task, T. The model is trained to estimate the conditional distribution $p(\boldsymbol{y} \mid \boldsymbol{x}, T)$, where T is a description of the task we wish the model to perform. In our example of recognizing cats after having read about cats, the output is a binary variable y with $y = 1$ indicating "yes" and $y = 0$ indicating "no." The task variable T then represents questions to be answered, such as "Is there a cat in this image?" If we have a training set containing unsupervised examples of objects that live in the same space as T, we may be able to infer the meaning of unseen instances of T. In our example of recognizing cats without having seen an image of the cat, it is important that we have had unlabeled text data containing sentences such as "cats have four legs" or "cats have pointy ears."

Zero-shot learning requires T to be represented in a way that allows some sort of generalization. For example, T cannot be just a one-hot code indicating an object category. Socher *et al.* (2013b) provide instead a distributed representation of object categories by using a learned word embedding for the word associated with each category.

A similar phenomenon happens in machine translation (Klementiev *et al.*, 2012; Mikolov *et al.*, 2013b; 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 that 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 and a distributed representation for words in language Y, then created a link (possibly two-way) relating the two spaces, via training examples consisting of matched pairs of sentences in both languages. This transfer will be most successful if all three ingredients (the two representations and the relations between them) are learned jointly.

Zero-shot learning is a particular form of transfer learning. The same principle

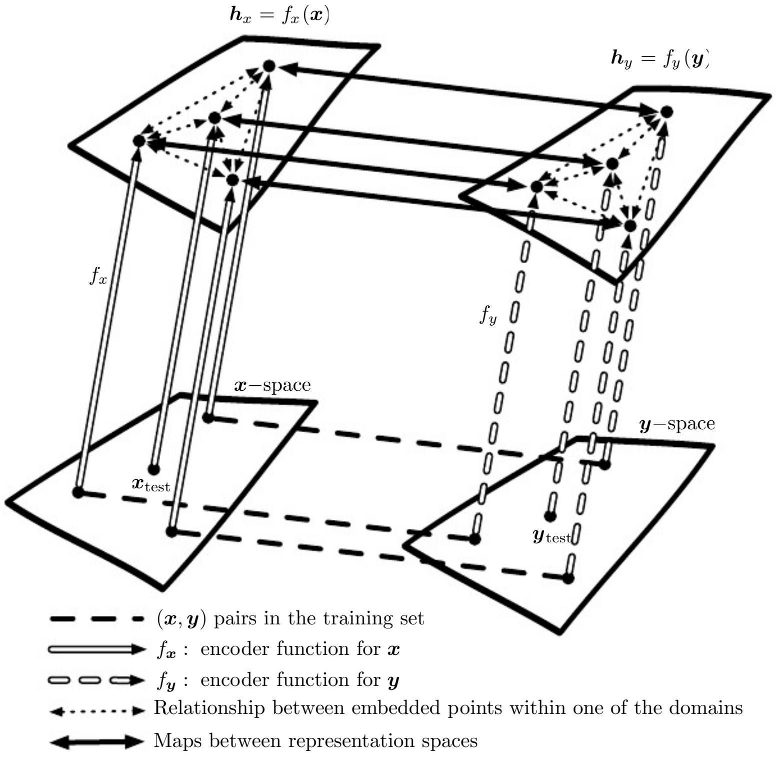
$h_x = f_x(x)$
$h_y = f_y(y)$
f_x
f_y
x−space
y−space
x_{test}
y_{test}
(x, y) pairs in the training set
f_x : encoder function for x
f_y : encoder function for y
Relationship between embedded points within one of the domains
Maps between representation spaces

Figure 15.3: Transfer learning between two domains $\boldsymbol{x}$ and $\boldsymbol{y}$ enables zero-shot learning. Labeled or unlabeled examples of $\boldsymbol{x}$ allow one to learn a representation function f_x and similarly with examples of $\boldsymbol{y}$ to learn f_y. Each application of the f_x and f_y functions appears as an upward arrow, with the style of the arrows indicating which function is applied. Distance in $\boldsymbol{h}_x$ space provides a similarity metric between any pair of points in $\boldsymbol{x}$ space that may be more meaningful than distance in $\boldsymbol{x}$ space. Likewise, distance in $\boldsymbol{h}_y$ space provides a similarity metric between any pair of points in $\boldsymbol{y}$ space. Both of these similarity functions are indicated with dotted bidirectional arrows. Labeled examples (dashed horizontal lines) are pairs $(\boldsymbol{x}, \boldsymbol{y})$ that allow one to learn a one-way or two-way map (solid bidirectional arrow) between the representations $f_x(\boldsymbol{x})$ and the representations $f_y(\boldsymbol{y})$ and to anchor these representations to each other. Zero-data learning is then enabled as follows. One can associate an image $\boldsymbol{x}_{\text{test}}$ to a word $\boldsymbol{y}_{\text{test}}$, even if no image of that word was ever presented, simply because word representations $f_y(\boldsymbol{y}_{\text{test}})$ and image representations $f_x(\boldsymbol{x}_{\text{test}})$ can be related to each other via the maps between representation spaces. It works because, although that image and that word were never paired, their respective feature vectors $f_x(\boldsymbol{x}_{\text{test}})$ and $f_y(\boldsymbol{y}_{\text{test}})$ have been related to each other. Figure inspired from suggestion by Hrant Khachatrian.

explains how one can perform **multimodal 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 representations), concepts in one representation are anchored in the other, and vice versa, allowing one to meaningfully generalize to new pairs. The procedure is illustrated in figure 15.3.

15.3 Semi-Supervised Disentangling of Causal Factors

An important question about representation learning is: what makes one representation better than another? One hypothesis is that an ideal representation is one in which the features within the representation correspond to the underlying causes of the observed data, with separate features or directions in feature space corresponding to different causes, so that the representation disentangles the causes from one another. This hypothesis motivates approaches in which we first seek a good representation for $p(\boldsymbol{x})$. Such a representation may also be a good representation for computing $p(\boldsymbol{y} \mid \boldsymbol{x})$ if $\boldsymbol{y}$ is among the most salient causes of $\boldsymbol{x}$. This idea has guided a large amount of deep learning research since at least the 1990s (Becker and Hinton, 1992; Hinton and Sejnowski, 1999) in more detail. For other arguments about when semi-supervised learning can outperform pure supervised learning, we refer the reader to section 1.2 of Chapelle *et al.* (2006).

In other approaches to representation learning, we have often been concerned with a representation that is easy to model—for example, one whose entries are sparse or independent from each other. A representation that cleanly separates the underlying causal factors may not necessarily be one that is easy to model. However, a further part of the hypothesis motivating semi-supervised learning via unsupervised representation learning is that for many AI tasks, these two properties coincide: once we are able to obtain the underlying explanations for what we observe, it generally becomes easy to isolate individual attributes from the others. Specifically, if a representation $\boldsymbol{h}$ represents many of the underlying causes of the observed $\boldsymbol{x}$, and the outputs $\boldsymbol{y}$ are among the most salient causes, then it is easy to predict $\boldsymbol{y}$ from $\boldsymbol{h}$.

First, let us see how semi-supervised learning can fail because unsupervised learning of $p(\mathbf{x})$ is of no help to learning $p(\mathbf{y} \mid \mathbf{x})$. Consider, for example, the case where $p(\mathbf{x})$ is uniformly distributed and we want to learn $f(\boldsymbol{x}) = \mathbb{E}[\mathbf{y} \mid \boldsymbol{x}]$. Clearly,

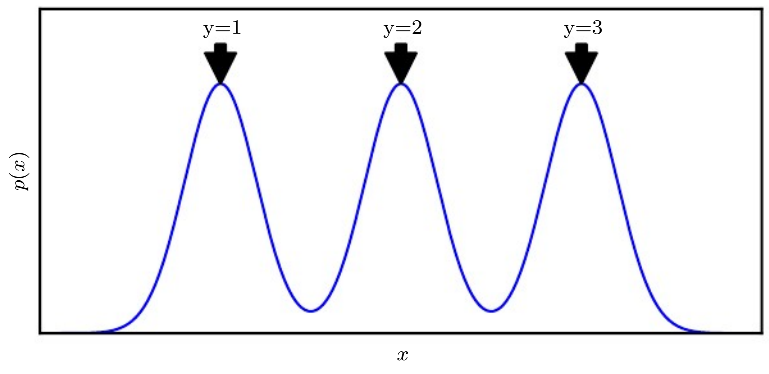

Figure 15.4: Mixture model. Example of a density over x that is a mixture over three components. The component identity is an underlying explanatory factor, y. Because the mixture components (e.g., natural object classes in image data) are statistically salient, just modeling $p(x)$ in an unsupervised way with no labeled example already reveals the factor y.

observing a training set of $\boldsymbol{x}$ values alone gives us no information about $p(\mathbf{y} \mid \mathbf{x})$.

Next, let us see a simple example of how semi-supervised learning can succeed. Consider the situation where $\mathbf{x}$ arises from a mixture, with one mixture component per value of $\mathbf{y}$, as illustrated in figure 15.4. If the mixture components are well separated, then modeling $p(\mathbf{x})$ reveals precisely where each component is, and a single labeled example of each class will then be enough to perfectly learn $p(\mathbf{y} \mid \mathbf{x})$. But more generally, what could tie $p(\mathbf{y} \mid \mathbf{x})$ and $p(\mathbf{x})$ together?

If $\mathbf{y}$ is closely associated with one of the causal factors of $\mathbf{x}$, then $p(\mathbf{x})$ 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 $\mathbf{h}$ represent all those factors. The true generative process can be conceived as structured according to this directed graphical model, with $\mathbf{h}$ as the parent of $\mathbf{x}$:

$$p(\mathbf{h}, \mathbf{x}) = p(\mathbf{x} \mid \mathbf{h}) p(\mathbf{h}). \tag{15.1}$$

As a consequence, the data has marginal probability

$$p(\boldsymbol{x}) = \mathbb{E}_{\mathbf{h}} p(\boldsymbol{x} \mid \boldsymbol{h}). \tag{15.2}$$

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 these (or closely related to one of them), then it will be 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:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{y})p(\mathbf{y})}{p(\mathbf{x})}. \tag{15.3}$$

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. Therefore, in situations respecting these assumptions, semi-supervised learning should improve performance.

An important research problem regards the fact that most observations are formed by an extremely large number of underlying causes. Suppose $\mathbf{y} = \mathrm{h}_i$, but the unsupervised learner does not know which h_i. The brute force solution is for an unsupervised learner to learn a representation that captures *all* the reasonably salient generative factors h_j and disentangles them from each other, thus making it easy to predict $\mathbf{y}$ from $\mathbf{h}$, regardless of which h_i is associated with $\mathbf{y}$.

In practice, the brute force solution is not feasible because it is not possible to capture all or most of the factors of variation that influence an observation. For example, in a visual scene, should the representation always encode all the smallest objects in the background? It is a well-documented psychological phenomenon that human beings fail to perceive changes in their environment that are not immediately relevant to the task they are performing—see, for example Simons and Levin (1998). An important research frontier in semi-supervised learning is determining *what* to encode in each situation. Currently, two of the main strategies for dealing with a large number of underlying causes are to use a supervised learning signal at the same time as the unsupervised learning signal so that the model will choose to capture the most relevant factors of variation, or to use much larger representations if using purely unsupervised learning.

An emerging strategy for unsupervised learning is to modify the definition of which underlying causes are most salient. Historically, autoencoders and generative models have been trained to optimize a fixed criterion, often similar to mean squared error. These fixed criteria determine which causes are considered salient. For example, mean squared error applied to the pixels of an image implicitly specifies that an underlying cause is only salient if it significantly changes the brightness of a large number of pixels. This can be problematic if the task we wish to solve involves interacting with small objects. See figure 15.5 for an example

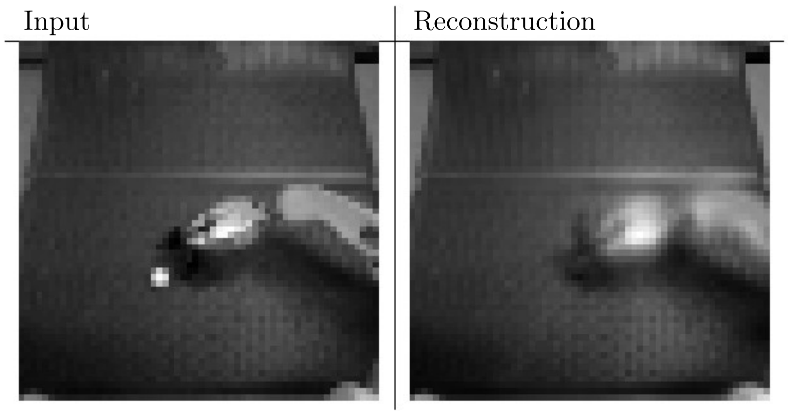

Figure 15.5: An autoencoder trained with mean squared error for a robotics task has failed to reconstruct a ping pong ball. The existence of the ping pong ball and all its spatial coordinates are important underlying causal factors that generate the image and are relevant to the robotics task. Unfortunately, the autoencoder has limited capacity, and the training with mean squared error did not identify the ping pong ball as being salient enough to encode. Images graciously provided by Chelsea Finn.

of a robotics task in which an autoencoder has failed to learn to encode a small ping pong ball. This same robot is capable of successfully interacting with larger objects, such as baseballs, which are more salient according to mean squared error.

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 a recently developed approach called **generative adversarial networks** (Goodfellow *et al.*, 2014c). In this approach, a generative model is trained to fool a feedforward classifier. The feedforward classifier attempts to recognize all samples from the generative model as being fake and all samples from the training set as being real. In this framework, any structured pattern that the feedforward network can recognize is highly salient. The generative adversarial network is described in more detail in section 20.10.4. For the purposes of the present discussion, it is sufficient to understand that the networks *learn* how to determine what is salient. Lotter *et al.* (2015) showed that models trained to generate images of human heads will often neglect to generate the ears when trained with mean squared error, but will successfully generate the ears when trained with the adversarial framework. Because the ears are not extremely bright or dark compared to the surrounding skin, they are not especially salient according to mean squared error loss, but their highly recognizable shape

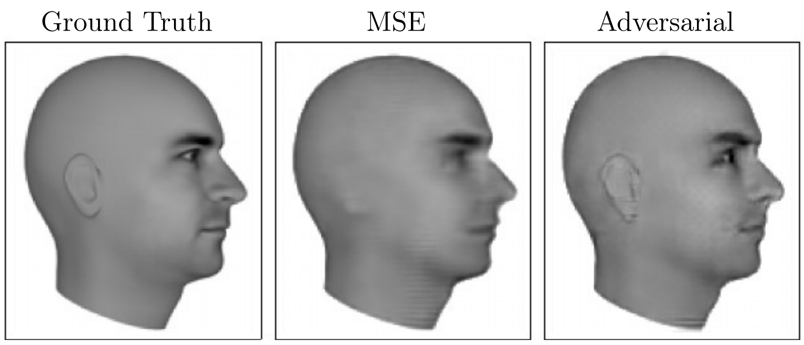

Figure 15.6: Predictive generative networks provide an example of the importance of learning which features are salient. In this example, the predictive generative network has been trained to predict the appearance of a 3-D model of a human head at a specific viewing angle. *(Left)*Ground truth. This is the correct image, which the network should emit. *(Center)*Image produced by a predictive generative network trained with mean squared error alone. Because the ears do not cause an extreme difference in brightness compared to the neighboring skin, they were not sufficiently salient for the model to learn to represent them. *(Right)*Image produced by a model trained with a combination of mean squared error and adversarial loss. Using this learned cost function, the ears are salient because they follow a predictable pattern. Learning which underlying causes are important and relevant enough to model is an important active area of research. Figures graciously provided by Lotter *et al.* (2015).

and consistent position means that a feedforward network can easily learn to detect them, making them highly salient under the generative adversarial framework. See figure 15.6 for example images. Generative adversarial networks are only one step toward determining which factors should be represented. We expect that future research will discover better ways of determining which factors to represent and develop mechanisms for representing different factors depending on the task.

A benefit of learning the underlying causal factors, as pointed out by Schölkopf *et al.* (2012), is that 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 were reversed, this 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 nonstationarity, or changes in the nature of the task, *the causal mechanisms remain invariant* ("the laws of the universe are constant"), while the marginal distribution over the underlying causes can change. 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 $p(\mathbf{x} \mid \mathbf{h})$.

15.4 Distributed Representation

Distributed representations of concepts—representations composed of many elements that can be set separately from each other—are one of the most important tools for representation learning. Distributed representations are powerful because they can use n features with k values to describe k^n different concepts. As we have seen throughout this book, neural networks with multiple hidden units and probabilistic models with multiple latent variables both make use of the strategy of distributed representation. We now introduce an additional observation. Many deep learning algorithms are motivated by the assumption that the hidden units can learn to represent the underlying causal factors that explain the data, as discussed in section 15.3. Distributed representations are natural for this approach, because each direction in representation space can correspond to the value of a different underlying configuration variable.

An example of a 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, as illustrated in figure 15.7. 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, as illustrated in figure 15.8. 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). A symbolic representation is a specific example of the broader class of nondistributed representations, which are representations that may contain many entries but without significant meaningful separate control over each entry.

The following examples of learning algorithms are based on nondistributed representations:

- Clustering methods, including the k-means algorithm: each input point is assigned to exactly one cluster.

- k-nearest neighbors algorithms: one or a few templates or prototype examples are associated with a given input. In the case of $k > 1$, multiple values describe each input, but they cannot be controlled separately from each other, so this does not qualify as a true distributed representation.

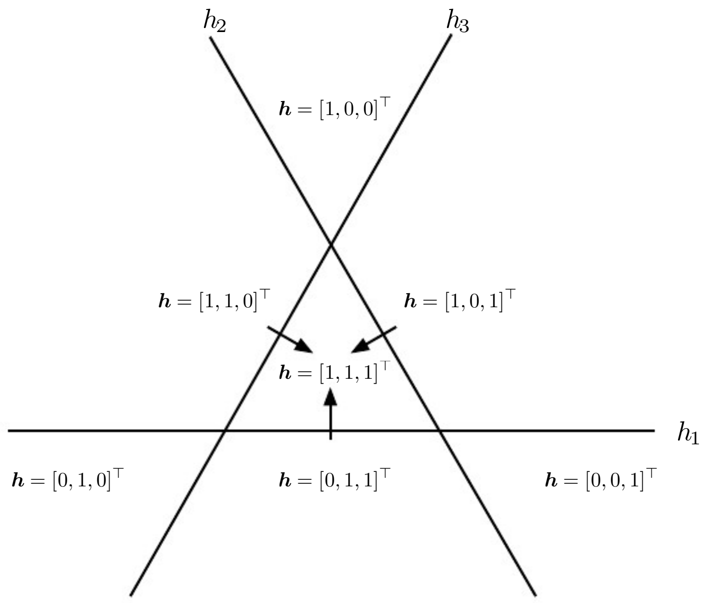
h_2
h_3
$\boldsymbol{h} = [1, 0, 0]^\top$
$\boldsymbol{h} = [1, 1, 0]^\top$
$\boldsymbol{h} = [1, 0, 1]^\top$
$\boldsymbol{h} = [1, 1, 1]^\top$
h_1
$\boldsymbol{h} = [0, 1, 0]^\top$
$\boldsymbol{h} = [0, 1, 1]^\top$
$\boldsymbol{h} = [0, 0, 1]^\top$

Figure 15.7: Illustration of how a learning algorithm based on a distributed representation breaks up the input space into regions. In this example, there are three binary features h_1, h_2, and h_3. Each feature is defined by thresholding the output of a learned linear transformation. Each feature divides $\mathbb{R}^2$ into two half-planes. Let h_i^+ be the set of input points for which $h_i = 1$, and h_i^- be the set of input points for which $h_i = 0$. In this illustration, each line represents the decision boundary for one h_i, with the corresponding arrow pointing to the h_i^+ side of the boundary. The representation as a whole takes on a unique value at each possible intersection of these half-planes. For example, the representation value $[1, 1, 1]^\top$ corresponds to the region $h_1^+ \cap h_2^+ \cap h_3^+$. Compare this to the non-distributed representations in figure 15.8. In the general case of d input dimensions, a distributed representation divides $\mathbb{R}^d$ by intersecting half-spaces rather than half-planes. The distributed representation with n features assigns unique codes to $O(n^d)$ different regions, while the nearest neighbor algorithm with n examples assigns unique codes to only n regions. The distributed representation is thus able to distinguish exponentially many more regions than the nondistributed one. Keep in mind that not all $\boldsymbol{h}$ values are feasible (there is no $\boldsymbol{h} = \boldsymbol{0}$ in this example), and that a linear classifier on top of the distributed representation is not able to assign different class identities to every neighboring region; even a deep linear-threshold network has a VC dimension of only $O(w \log w)$, where w is the number of weights (Sontag, 1998). The combination of a powerful representation layer and a weak classifier layer can be a strong regularizer; a classifier trying to learn the concept of "person" versus "not a person" does not need to assign a different class to an input represented as "woman with glasses" than it assigns to an input represented as "man without glasses." This capacity constraint encourages each classifier to focus on few h_i and encourages $\boldsymbol{h}$ to learn to represent the classes in a linearly separable way.

- 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. As with the k-nearest neighbors algorithm, each input is represented with multiple values, but those values cannot readily be controlled separately from each other.
- Kernel machines with a Gaussian kernel (or other similarly local kernel): although the degree of activation 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. A leaf may correspond to the last two words being w_1 and w_2, for example. Separate parameters are estimated for each leaf of the tree (with some sharing being possible).

For some of these nondistributed algorithms, the output is not constant by parts but instead interpolates between neighboring regions. The relationship between the number of parameters (or examples) and the number of regions they can define remains linear.

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, "`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. For example, our distributed representation may contain entries such as "`has_fur`" or "`number_of_legs`" that have the same value for the embedding of both "`cat`" and "`dog`." Neural language models that operate on distributed representations of words generalize much better than other models that operate directly on one-hot representations of words, as discussed in 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.

When and why can there be a statistical advantage from using a distributed representation as part of a learning algorithm? Distributed representations can have a statistical advantage when an apparently complicated structure can be compactly represented using a small number of parameters. Some traditional nondistributed learning algorithms generalize only due to the smoothness assumption, which

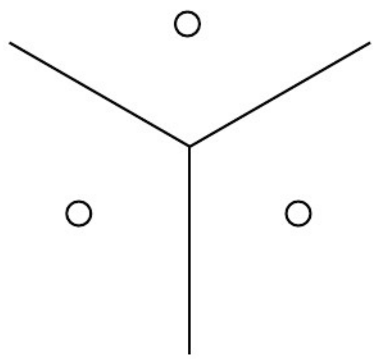

Figure 15.8: Illustration of how the nearest neighbor algorithm breaks up the input space into different regions. The nearest neighbor algorithm provides an example of a learning algorithm based on a nondistributed representation. Different non-distributed algorithms may have different geometry, but they typically break the input space into regions, *with a separate set of parameters for each region.* The advantage of a nondistributed approach is that, given enough parameters, it can fit the training set without solving a difficult optimization algorithm, because it is straightforward to choose a different output *independently* for each region. The disadvantage is that such nondistributed models generalize only locally via the smoothness prior, making it difficult to learn a complicated function with more peaks and troughs than the available number of examples. Contrast this with a distributed representation, figure 15.7.

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 when we move to a nearby input $x + \epsilon$. This assumption is clearly very useful, but it suffers from the curse of dimensionality: to learn a target function that increases and decreases many times in many different regions,[1] we may need a number of examples that is at least as large as the number of distinguishable 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 decoder mapping from symbol to value. However, this does not allow us to generalize to new symbols for new regions.

If we are lucky, there may be some regularity in the target function, besides being smooth. For example, a convolutional network with max pooling can recognize an object regardless of its location in the image, even though spatial translation of the object may not correspond to smooth transformations in the input space.

Let us examine a special case of a distributed representation learning algorithm, which extracts binary features by thresholding linear functions of the input. Each binary feature in this representation divides $\mathbb{R}^d$ into a pair of half-spaces, as illustrated in figure 15.7. The exponentially large number of intersections of n of the corresponding half-spaces determines how many regions this distributed representation learner can distinguish. How many regions are generated by an arrangement of n hyperplanes in $\mathbb{R}^d$? By applying a general result concerning the intersection of hyperplanes (Zaslavsky, 1975), one can show (Pascanu *et al.*, 2014b) that the number of regions this binary feature representation can distinguish is

$$\sum_{j=0}^{d} \binom{n}{j} = O(n^d). \tag{15.4}$$

Therefore, we see a growth that is exponential in the input size and polynomial in the number of hidden units.

This provides a geometric argument to explain the generalization power of distributed representation: with $O(nd)$ parameters (for n linear threshold features in $\mathbb{R}^d$), we can distinctly represent $O(n^d)$ regions in input space. If instead we made no assumption at all about the data, and used a representation with one unique symbol for each region, and separate parameters for each symbol to recognize its

[1] Potentially, we may want to learn a function whose behavior is distinct in 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.

corresponding portion of $\mathbb{R}^d$, then specifying $O(n^d)$ regions would require $O(n^d)$ examples. More generally, the argument in favor of the distributed representation could be extended to the case where instead of using linear threshold units we use nonlinear, possibly continuous, feature extractors for each of the attributes in the distributed representation. The argument in this case is that if a parametric transformation with k parameters can learn about r regions in input space, with $k \ll r$, and if obtaining such a representation was useful to the task of interest, then we could potentially generalize much better in this way than in a nondistributed setting, where we would need $O(r)$ examples to obtain the same features and associated partitioning of the input space into r regions. Using fewer parameters to represent the model means that we have fewer parameters to fit, and thus require far fewer training examples to generalize well.

A further part of the argument for why models based on distributed representations generalize well is that their capacity remains limited despite being able to distinctly encode so many different regions. For example, the VC dimension of a neural network of linear threshold units is only $O(w \log w)$, where w is the number of weights (Sontag, 1998). This limitation arises because, while we can assign very many unique codes to representation space, we cannot use absolutely all the code space, nor can we learn arbitrary functions mapping from the representation space $\boldsymbol{h}$ to the output $\boldsymbol{y}$ using a linear classifier. The use of a distributed representation combined with a linear classifier thus expresses a prior belief that the classes to be recognized are linearly separable as a function of the underlying causal factors captured by $\boldsymbol{h}$. We will typically want to learn categories such as the set of all images of all green objects or the set of all images of cars, but not categories that require nonlinear XOR logic. For example, we typically do not want to partition the data into the set of all red cars and green trucks as one class and the set of all green cars and red trucks as another class.

The ideas discussed so far have been abstract, but they may be experimentally validated. Zhou *et al.* (2015) found that hidden units in a deep convolutional network trained on the ImageNet and Places benchmark datasets learn features that are often interpretable, corresponding to a label that humans would naturally assign. In practice it is certainly not always the case that hidden units learn something that has a simple linguistic name, but it is interesting to see this emerge near the top levels of the best computer vision deep networks. What such features have in common is that one could imagine *learning about each of them without having to see all the configurations of all the others*. Radford *et al.* (2015) 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. Figure 15.9 demonstrates that one direction in representation space corresponds

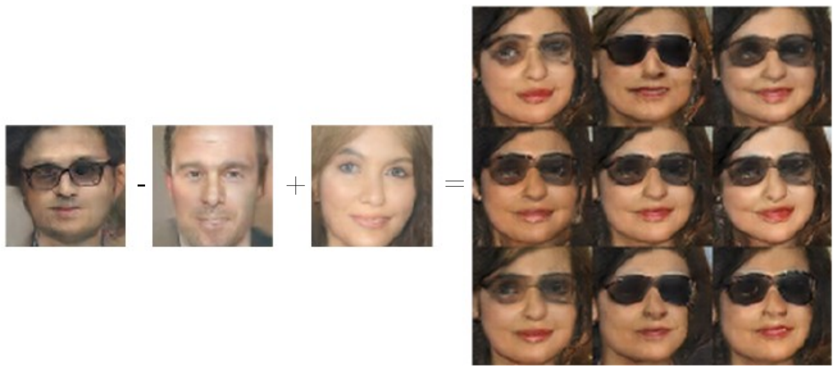

Figure 15.9: A generative model has learned a distributed representation that disentangles the concept of gender from the concept of wearing glasses. If we begin with the representation of the concept of a man with glasses, then subtract the vector representing the concept of a man without glasses, and finally add the vector representing the concept of a woman without glasses, we obtain the vector representing the concept of a woman with glasses. The generative model correctly decodes all these representation vectors to images that may be recognized as belonging to the correct class. Images reproduced with permission from Radford *et al.* (2015).

to whether the person is male or female, while another corresponds to whether the person is wearing glasses. These features were discovered automatically, not fixed a priori. There is no need to have labels for the hidden unit classifiers: gradient descent on an objective function of interest naturally learns semantically interesting features, as long as the task requires such features. We can learn about the distinction between male and female, or about the presence or absence of glasses, without having to characterize all the configurations of the $n - 1$ other features by examples covering all these combinations of values. 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.

15.5 Exponential Gains from Depth

We have seen in section 6.4.1 that multilayer perceptrons are universal approximators, and that some functions can be represented by exponentially smaller deep networks compared to shallow networks. This decrease in model size leads to improved statistical efficiency. In this section, we describe how similar results apply more generally to other kinds of models with distributed hidden representations.

In section 15.4, we saw an example of a generative model that learned about the explanatory factors underlying images of faces, including the person's gender and whether they are wearing glasses. The generative model that accomplished this task was based on a deep neural network. It would not be reasonable to expect a shallow network, such as a linear network, to learn the complicated relationship between these abstract explanatory factors and the pixels in the image. In this and other AI tasks, the factors that can be chosen almost independently from each other yet still correspond to meaningful inputs are more likely to be very high level and related in highly nonlinear ways to the input. We argue that 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 nonlinearities.

It has been proved in many different settings that organizing computation through the composition of many nonlinearities and a hierarchy of reused features can give an exponential boost to statistical efficiency, on top of the exponential boost given by using a distributed representation. Many kinds of networks (e.g., with saturating nonlinearities, Boolean gates, sum/products, or RBF units) with a single hidden layer can be shown to be universal approximators. A model family that is a universal approximator can approximate a large class of functions (including all continuous functions) up to any nonzero tolerance level, given enough hidden units. However, the required number of hidden units may be very large. Theoretical results 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 that would require an exponential number of hidden units (with respect to the input size) with insufficient depth (depth 2 or depth $k-1$).

In section 6.4.1, we saw that deterministic feedforward networks are universal approximators of functions. Many structured probabilistic models with a single hidden layer of latent variables, including restricted Boltzmann machines and deep belief networks, are universal approximators of probability distributions (Le Roux and Bengio, 2008, 2010; Montúfar and Ay, 2011; Montúfar, 2014; Krause *et al.*, 2013).

In section 6.4.1, we saw that a sufficiently deep feedforward network can have an exponential advantage over a network that is too shallow. Such results can also be obtained for other models such as probabilistic models. One such probabilistic model is the **sum-product network**, or SPN (Poon and Domingos, 2011). These models use polynomial circuits to compute the probability distribution over a set of random variables. Delalleau and Bengio (2011) showed that there exist probability distributions for which a minimum depth of SPN is required to avoid needing an exponentially large model. Later, Martens and Medabalimi (2014) showed that there are significant differences between every two finite depths of SPN, and that some of the constraints used to make SPNs tractable may limit their representational power.

Another interesting development is a set of theoretical results for the expressive power of families of deep circuits related to convolutional nets, highlighting an exponential advantage for the deep circuit even when the shallow circuit is allowed to only approximate the function computed by the deep circuit (Cohen *et al.*, 2015). By comparison, previous theoretical work made claims regarding only the case where the shallow circuit must exactly replicate particular functions.

15.6 Providing Clues to Discover Underlying Causes

To close this chapter, we come back to one of our original questions: what makes one representation better than another? One answer, first introduced in section 15.3, is that an ideal representation is one that disentangles the underlying causal factors of variation that generated the data, especially those factors that are relevant to our applications. Most strategies for representation learning are based on introducing clues that help the learning find these underlying factors of variations. The clues can help the learner separate these observed factors from the others. Supervised learning provides a very strong clue: a label $\boldsymbol{y}$, presented with each $\boldsymbol{x}$, that usually specifies the value of at least one of the factors of variation directly. More generally, to make use of abundant unlabeled data, representation learning makes use of other, less direct hints about the underlying factors. These hints take the form of implicit prior beliefs that we, the designers of the learning algorithm, impose in order to guide the learner. Results such as the no free lunch theorem show that regularization strategies are necessary to obtain good generalization. While it is impossible to find a universally superior regularization strategy, one goal of deep learning is to find a set of fairly generic regularization strategies that are applicable to a wide variety of AI tasks, similar to the tasks that people and animals are able to solve.

We provide here a list of these generic regularization strategies. The list is clearly not exhaustive but gives some concrete examples of how learning algorithms can be encouraged to discover features that correspond to underlying factors. This list was introduced in section 3.1 of Bengio *et al.* (2013d) and has been partially expanded here.

- *Smoothness*: This is the assumption that $f(\boldsymbol{x} + \epsilon\boldsymbol{d}) \approx f(\boldsymbol{x})$ for unit $\boldsymbol{d}$ and small ϵ. This assumption allows the learner to generalize from training examples to nearby points in input space. Many machine learning algorithms leverage this idea, but it is insufficient to overcome the curse of dimensionality.

- *Linearity*: Many learning algorithms assume that relationships between some variables are linear. This allows the algorithm to make predictions even very far from the observed data, but can sometimes lead to overly extreme predictions. Most simple machine learning algorithms that do not make the smoothness assumption instead make the linearity assumption. These are in fact different assumptions—linear functions with large weights applied to high-dimensional spaces may not be very smooth. See Goodfellow *et al.* (2014b) for a further discussion of the limitations of the linearity assumption.

- *Multiple explanatory factors*: Many representation learning algorithms are motivated by the assumption that the data is generated by multiple underlying explanatory factors, and that most tasks can be solved easily given the state of each of these factors. Section 15.3 describes how this view motivates semi-supervised learning via representation learning. Learning the structure of $p(\boldsymbol{x})$ requires learning some of the same features that are useful for modeling $p(\boldsymbol{y} \mid \boldsymbol{x})$ because both refer to the same underlying explanatory factors. Section 15.4 describes how this view motivates the use of distributed representations, with separate directions in representation space corresponding to separate factors of variation.

- *Causal factors*: The model is constructed in such a way that it treats the factors of variation described by the learned representation $\boldsymbol{h}$ as the causes of the observed data $\boldsymbol{x}$, and not vice versa. As discussed in section 15.3, this is advantageous for semi-supervised learning and makes the learned model more robust when the distribution over the underlying causes changes or when we use the model for a new task.

- *Depth*, or *a hierarchical organization of explanatory factors*: High-level, abstract concepts can be defined in terms of simple concepts, forming a hierarchy. From another point of view, the use of a deep architecture expresses our belief that the task should be accomplished via a multistep program, with each step referring back to the output of the processing accomplished via previous steps.

- *Shared factors across tasks*: When we have many tasks corresponding to different y_i variables sharing the same input $\mathbf{x}$, or when 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 y_i is associated with a different subset from a common pool of relevant factors $\mathbf{h}$. Because these subsets overlap, learning all the $P(\mathrm{y}_i \mid \mathbf{x})$ 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 with a much smaller dimensionality than the original space where the data live. Many machine learning algorithms behave sensibly only on this manifold (Goodfellow *et al.*, 2014b). Some machine learning algorithms, especially autoencoders, attempt to explicitly learn the structure of the manifold.

- *Natural clustering*: Many machine learning algorithms assume that each connected manifold in the input space may be assigned to a single class. The data may lie on many disconnected manifolds, but the class remains constant within each one of these. This assumption motivates a variety of learning algorithms, including tangent propagation, double backprop, the manifold tangent classifier and adversarial training.

- *Temporal and spatial coherence*: Slow feature analysis and related algorithms make the assumption that the most important explanatory factors change slowly over time, or at least that it is easier to predict the true underlying explanatory factors than to predict raw observations such as pixel values. See section 13.3 for further description of this approach.

- *Sparsity*: Most features should presumably not be relevant to describing most inputs—there is no need to use a feature that detects elephant trunks when representing an image of a cat. It is therefore reasonable to impose a prior that any feature that can be interpreted as "present" or "absent" should be absent most of the time.

- *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 autoencoder 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.

The concept of representation learning ties together all 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.