Overview of machine learning

Kevin P. Murphy

Last updated November 26, 2007

1 Introduction

In this Chapter, we provide a brief overview of the most commonly studied problems and solution methods within the field of machine learning. This Chapter might be hard to understand on a first reading; we recommend you return to it after reading some of the subsequent chapters. However, it serves to provide a "big picture" and some motivation, and to introduce some useful notation.

2 Supervised learning

The most widely studied problem in machine learning is **supervised learning**. We are given a **labeled training set** of input-output pairs, $D = (\mathbf{x}_i, y_i)_{i=1}^n$, and have to learn a way to predict the **output** or **target** $\tilde{y}$ for a novel **test input** $\tilde{\mathbf{x}}$ (i.e, for $\tilde{x} \notin D$). (We use the tilde notation to denote test cases that we have not seen before.) Some examples include: predicting if someone has cancer $\tilde{y} \in \{0, 1\}$ given some measured variables $\tilde{\mathbf{x}}$; predicting the stock price tomorrow $\tilde{y} \in \mathbb{R}$ given the stock prices today $\tilde{\mathbf{x}}$; etc.

A common approach is to just predict one's "best guess", such as $\hat{y}(\tilde{\mathbf{x}})$. However, we prefer to compute a probability distribution over the output, $p(\tilde{y}|\tilde{\mathbf{x}})$, since it is very useful to have a measure of confidence associated with one's prediction, especially in medical and financial domains. In addition, probabilistic methods are essential for unsupervised learning, as we discuss in Section 3.

If y is discrete or **categorical**, say $y \in \{1, 2, \ldots, C\}$, this problem is called **classification** or **pattern recognition**. If there are $C = 2$ classes or **labels**, the problem is called **binary** classification (see Figure 1 for an example), otherwise it is called **multi-class** classification. We usually assume the classes are mutually exclusive, so y can only be in one possible state. If we want to allow multiple labels, we can represent y by a bit-vector of length C, so $y_j = 1$ if y belongs to class j.

If y is continuous, say $y \in \mathbb{R}$, this problem is called **regression**. If y is multidimensional, say $y \in \mathbb{R}^q$, we call it **multivariate regression**. If y is discrete, but ordered (e.g., $y \in$ {low,medium,high}), the problem is called **ordinal regression**.

A priori, our prediction might be quite poor, but we are provided with a labeled training set of input-output pairs, $D = (\mathbf{x}_i, y_i)_{i=1}^n$, which provides a set of examples of the "right response" for a set of possible inputs. If each input

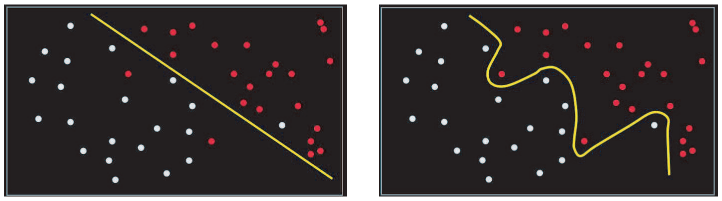
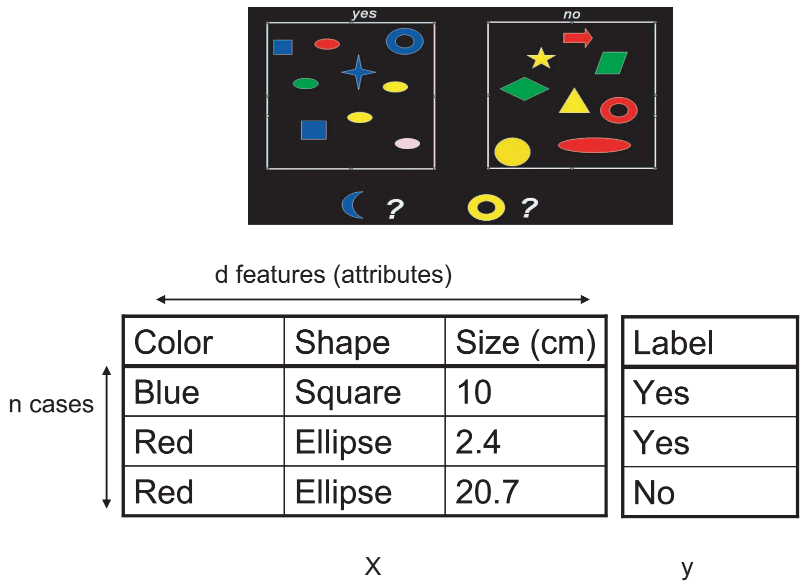


Figure 1: Illustration of binary classification where the input space is $\mathbb{R}^2$. White points are negative, red points are positive. The axes could represent cholestrol and insulin levels, and the class labels could represent "healthy" or "unhealthy". **Left:** linear classifier: we separate the classes with a line. Points on the **decision boundary** are equally likely to be in class 1 or class 2; as we move away from the boundary, $p(y = 1|\mathbf{x})$ increases or decreases, depending on which side of the boundary we are on. The parameters controlling the shape of the line are called $\mathbf{w}$. The line makes 3 misclassification errors. **Right:** we separate the classes with a "wiggly" boundary. This makes no errors, but may have **overfit**. Source: Leslie Kaelbling.



Color	Shape	Size (cm)	Label
Blue	Square	10	Yes
Red	Ellipse	2.4	Yes
Red	Ellipse	20.7	No

Figure 2: Design matrix. We assume the user has converted the input data (here, coloured shapes) into an adequate set of features, and has labeled every example. The learning algorithm then tries to learn a mapping from the features to the class label. Figure courtesy of Leslie Kaelbling.

$\mathbf{x}_i$ is a d-dimensional vector, called a **feature vector**, we can store the training data in an $n \times d$ **design matrix**. See Figure 2 for an example. We can then use the **posterior predictive density**, $p(\tilde{y}|\tilde{\mathbf{x}}, D)$, to generate possible answers.[1]

We can assess how well we are doing by computing the **posterior predictive likelihood**, i.e., how much probability mass does our model M assign to future data? We can approximate this by looking at the empirical performance of the model on a **test set**. In particular, we can compute the average **log-likelihood** of the test data:

$$\ell(M|\tilde{D}) = \frac{1}{m}\sum_{i=1}^{m} \log p(\tilde{y}_i|\tilde{\mathbf{x}}, D, M) \tag{1}$$

where m is the number of test cases, M is the model we are testing, and D is the training data. A good model will be able to predict the future, and hence will have high predictive (log) likelihood.

A simpler performance measure, in the case of classifiers, is just to compute the **misclassification rate** or **error rate**. However, we may want more than just being right; we may want to know when we are right, so that if we are uncertain about our answer, we can do something else, like ask for help, rather than make a potentially serious mistake (see Chapter **??**). In addition, another advantage of using log-likelihood is that it is well-defined even in the case of unsupervised learning, where there is no well defined error signal, as we will see in Section 3.

2.1 Parametric vs non-parametric models

In this book, we will mostly focus on **parametric models**, which essentially means we can "absorb" the training data into a fixed-length parameter vector $\boldsymbol{\theta}$ (of size independent of n), and then "throw away" the data. The process of infering $p(\boldsymbol{\theta}|D)$ is called **learning**; often we will approximate it by computing a point estimate, or "best guess", $\hat{\boldsymbol{\theta}}(D)$. (For example, when predicting the outcome of a series of coin tosses, we may estimate that the probability of heads is given by $\hat{\theta} = N_1/n$, where N_1 is the number of heads in D, and n is the total number of trials in D.) Given our parameter estimate, we can predict the future as follows:

$$p(\tilde{y}|\tilde{\mathbf{x}}, D) = \int p(\tilde{y}|\tilde{\mathbf{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta}|D)d\boldsymbol{\theta} \tag{2}$$

[1]The posterior predictive is just the probability distribution over possible outputs $\tilde{y}$ given the input $\tilde{\mathbf{x}}$, and given the training data D. It is called "posterior" because it is the distribution after we have seen the training data D. By contrast, $p(\tilde{y}|\tilde{\mathbf{x}})$ would be call the prior predictive.

We will often use a **plug-in** approximation to this:

$$p(\tilde{y}|\tilde{\mathbf{x}}, D) \approx p(\tilde{y}|\tilde{\mathbf{x}}, \hat{\boldsymbol{\theta}}(D)) \tag{3}$$

See Section **??** for more details.

The main alternative to a parametric model is a **non-parametric** model, in which the number of parameters is allowed to grow with the size of the data. A simple example is a **memory-based approach**, such as a **nearest neighbor** classifier (see Section **??**), in which we remember all our training examples, $(\mathbf{x}_i, y_i)$, and predict $\tilde{y}$ by comparing $\tilde{\mathbf{x}}$ to all the stored $\mathbf{x}_i$, and using the label y_i of the $\mathbf{x}_i$ that is closest to $\tilde{\mathbf{x}}$. Such models often work very well, but they need a lot of time and memory at test time (and possibly training time, too). In addition, such models are often hard to interpret, which may or may not be important in any given problem.

2.2 Generative vs discriminative models

There are two main ways to compute $p(y|\mathbf{x})$, depending on whether we write

$$p(y, \mathbf{x}) = p(y|\mathbf{x})p(\mathbf{x}) \tag{4}$$

or

$$p(y, \mathbf{x}) = p(y)p(\mathbf{x}|y) \tag{5}$$

(We drop the conditioning on $\boldsymbol{\theta}$ and D for notational simplicity.) The first approach is to directly estimate the **conditional density** $p(y|\mathbf{x})$. In the context of classification, this is called a **discriminative model**, since it discriminates between different classes y *given* the input. The distribution of the input, $p(\mathbf{x})$, is irrelevant, since we assume that the input is always known or **exogeneous**. (This is not true if we have **missing values** in the input, however.) We shall usually use $\mathbf{w}$ to denote the parameters of the distribution $p(y|\mathbf{x})$. These are often called regression **weights**; in statistics they are denoted by $\boldsymbol{\beta}$ and are called regression **coefficients**.

The alternative is to learn a **joint density model** of the inputs and outputs, $p(y, \mathbf{x})$. Then we can use **Bayes rule**[2] to infer the posterior on y:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} \tag{6}$$

The joint is usually written as

$$p(y, \mathbf{x}) = p(y)p(\mathbf{x}|y) \tag{7}$$

where $p(\mathbf{x}|y)$ is the **class-conditional density** and $p(y)$ is the prior over class labels. (The terminology is specific to the classification setting, since in the regression setting, it is more common to use discriminative models.) We will often use $\boldsymbol{\pi}$ to denote the parameters of the class prior $p(y)$, and $\boldsymbol{\theta}$ to denote the parameters of the class-conditional densities $p(\mathbf{z}|y)$. The approach of learning $p(y, \mathbf{x})$ is called **generative modeling**, since $p(\mathbf{x}|y)$ specifies a way to *generate* the feature vectors $\mathbf{x}$ for each possible class y. We will see an example of this in Section **??**.

We discuss the advantages and disadvantages of discriminative vs generative classifiers in Section **??**. Most classification methods that you may have heard of (logistic regression, neural networks, decision trees, SVMs) are discriminative (see Chapter **??** for a discussion of these models). However, generative classifiers are often easier to learn, so we will study these first. In particular, in Chapter **??**, we will study the naive Bayes classifier, which is widely used for email **spam filtering**.

2.3 Graphical models

The distinction between generative and discriminative models becomes much clearer if we use the notation of (directed) **graphical models**. We will explain these in Section **??**, and in more detail in Chapter **??**. However, the basic idea is very simple: we create a graph in which nodes represent random variables (which may be scalars or vectors), such as $\mathbf{x}$, y, $\boldsymbol{\theta}$, etc. We then draw arrows between the nodes which have a direct (probabilistic) dependence between them. We shade the nodes that are **observed** (known), and leave unshaded the nodes that are not observed (unknown).

[2]Recall that Bayes rule is simply $p(Y = i|X = j) = \frac{p(X=j|Y=i)p(Y=i)}{p(X=j)}$, where $p(X = j) = \sum_i p(X = j|Y = i)p(Y = i)$ is just a normalization constant to ensure $\sum_i p(Y = i|X = j) = 1$. See Section **??** for more details.

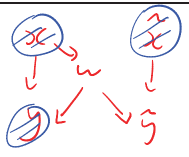
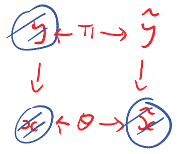
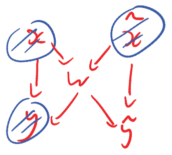
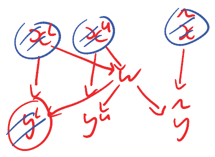
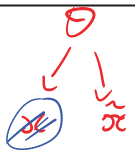
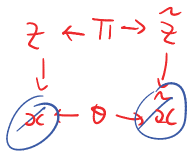
Name	Training data	Goal	Model
Conditional density estimation (discriminative model)	$D = (\mathbf{x}_i, y_i)$	$p(\tilde{y}\|\tilde{\mathbf{x}}, D)$	
Joint density estimation (generative model)	$D = (\mathbf{x}_i, y_i)$	$p(\tilde{y}\|\tilde{\mathbf{x}}, D)$	
Transductive learning	$D = (\mathbf{x}_i, y_i, \tilde{\mathbf{x}}_i)$	$p(\tilde{y}_i\|D)$	
Semi-supervised learning	$D = (\mathbf{x}_i^l, y_i^l, \mathbf{x}_i^u)$	$p(\tilde{y}\|\tilde{\mathbf{x}}, D)$	
Density estimation	$D = (\mathbf{x}_i)$	$p(\tilde{\mathbf{x}}\|D)$	
Latent variable model	$D = (\mathbf{x}_i)$	$p(\tilde{\mathbf{z}}\|\tilde{\mathbf{x}}, D)$	

Table 1: Most of machine learning, represented in terms of directed graphical models. Nodes on the left represent training data, nodes on the right represent test data, and nodes in the middle (with Greek letters, plus $\mathbf{w}$) are parameters. Shaded nodes are observed, unshaded nodes are unknown. The first four methods are supervised, the rest are unsupervised.

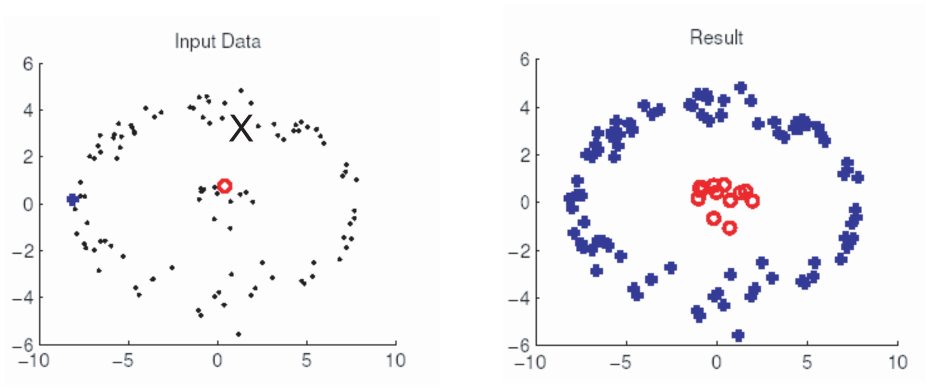


Figure 3: Example of semi-supervised learning. **Left**: two points are labeled, red and blue. Note that the point marked X is closer to the red point in terms of Euclidean distance, but closer to the blue point in terms of distance along the data **manifold**. **Right**: we propagate the labels to the nearest neighbors, thus effectively labeling all the points. Source: Nando de Freitas.

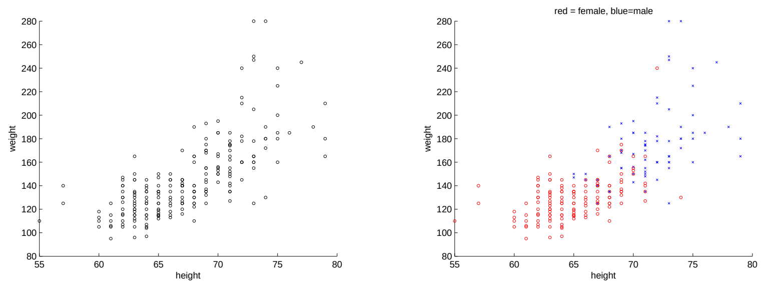


Figure 4: Left: A scatterplot of height and weight of various people. Right: points have been colour coded: men are blue crosses, women are red circles. Figure produced by `heightWeightDataPlot`.

We give some examples in Figure 2.3. For discriminative models, we have arcs from $\mathbf{x}$ and $\mathbf{w}$ to y to represent the term $p(y|\mathbf{x}, \mathbf{w})$. Similarly we have arcs from $\tilde{\mathbf{x}}$ and $\mathbf{w}$ to $\tilde{y}$. The goal of learning is to estimate $\mathbf{w}$ from $D = (\mathbf{x}_i, y_i)$, and then use this to predict $\tilde{y}$ given $\tilde{\mathbf{x}}$ and $\mathbf{w}$.

For generative models, we have arcs from $\boldsymbol{\pi}$ to y, to represent $p(y|\pi)$, and arcs from from y and $\boldsymbol{\theta}$ to $\mathbf{x}$, to represent the term $p(\mathbf{x}|y, \boldsymbol{\theta})$. At test time, $\tilde{\mathbf{x}}$ is observed, and we use Bayes rule to “invert the arrow” and infer $\tilde{y}$ from $\tilde{\mathbf{x}}$.

2.4 Variants of supervised learning

Some variations on the standard supervised learning paradigm have been proposed. In **transductive learning**, one is given access to the test questions at training time. That is, $D = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n \cup \{\tilde{\mathbf{x}}_i\}_{i=1}^m$, where n is the number of training cases and m is the number of test cases. One can use this when learning the weights, $p(\mathbf{w}|D)$. The goal is just to predict the outputs $\tilde{y}_i$ for the fixed test set $\tilde{\mathbf{x}}_i$, rather than for arbitrary test inputs. Note that in a standard discriminative model, we are allowed to include an arc from $\mathbf{x}$ to $\mathbf{w}$, reflecting the fact that the prior on $\mathbf{w}$ can depend on $\mathbf{x}$, $p(\mathbf{w}|\mathbf{x})$, since everything can be conditioned on $\mathbf{x}$. (We will see an example of this when we study g-priors in Section **??**.) However, in transductive learning, we can *also* include an arc from $\tilde{\mathbf{x}}$ to $\mathbf{w}$.

In **semi-supervised learning**, only some of the training cases have labels; call these $(\mathbf{x}_i^\ell, y_i^\ell)$. The other training cases have features but no labels; call these $\mathbf{x}_i^u$. The goal is to use all this data to predict future outputs, $p(\tilde{y}|\tilde{x}, D)$. By exploiting the fact that the inputs may be **similar** in some way, we can **propagate** the labels from the labeled examples to the unlabeled ones, and thereby increase the effective size of the training set. See Figure 3. (A similar technique can be used for transductive learning.)

3 Unsupervised learning

In **unsupervised learning**, the data only consists of a set of features, $D = \{\mathbf{x}_i\}$, with no target variable y_i to predict. The goal is to fit a model to all the data, in order to discover something "interesting". For example, given the data in Figure 4(left), we might hope to discover that there are two **clusters** underlying the data (in this case corresponding to male and females), and furthemore, we might hope to infer the assignment of points to clusters, as in Figure 4(right). Of course, since we are never told the "true" clustering (which might not even exist), we cannot measure performance in this way. Instead, we will assess performance of an unsupervised model M by its predictive likelihood:

$$\ell(M|\tilde{D}) = \frac{1}{m}\sum_{i=1}^{m} p(\tilde{\mathbf{x}}_i|D, M) \tag{8}$$

This is similar to Equation 1, except it tries to predict all the variables $\mathbf{x}$, rather than just the target variable y.

The hope is that a model that can model the data accurately has discovered something fundamental about the data, and hence it is sensible to try and "look under the hood" of the model, for example by examining its parameters, in order to learn something about the purported mechanism responsible for generating the data. Note that such model fitting plus model examination is the mainstay of statistics, whereas machine learning has traditionally been more concerned purely with supervised prediction tasks, often using hard-to-interpret models. This is also reflected in the fact that statisticians worry a lot about the confidence they can attach to their parameter estimates, whereas this topic is rarely discussed within machine learning.

Note that it is possible to try to intepret the parameters of a conditional density model, $p(y|\mathbf{x}, \boldsymbol{\theta})$, as well. Indeed, this is very common in statistics (linear regression being a very widely used example of a conditional density model). The only difference from supervised machine learning is that the goal is to learn something about the data, rather than merely predicting accurately. In other words, it is often not enough to build an accurate predictor; one often wants to know *why* it is an accurate predictor. Such a task is best described as simply "model fitting", and does not fall neatly into the categories of "supervised" or "unsupervised" learning.

Density models (both unconditional and conditional) often contain **hidden variables** or **latent variables**, $\mathbf{z}_i$, whose values are never observed. These are not part of the data, but are part of the model. They are just like parameters, except there are $O(n)$ of them, i.e., there is one $\mathbf{z}_i$ for each $\mathbf{x}_i$ (and for each y_i, if we are using a conditional density model). By contrast, we assume the number of parameters is fixed. Furthermore, we often think of "fixing" the parameters to the best value learned from the training data, $\hat{\theta}(D)$, whereas the hidden variables will be free to change at test time. However, in Section **??**, we will see that, from the Bayesian viewpoint, this distinction between hidden variables and parameters is rather artificial.

The meaning of the hidden variables $\mathbf{z}_i$ depends on the model/ application. In the clustering example above, $z_i \in \{1, \ldots, K\}$ represents the cluster to which $\mathbf{x}_i$ is assigned, and K is the (unknown) number of clusters. Another example is **dimensionality reduction**, in which $\mathbf{z}_i$ is a low-dimensional representation of $\mathbf{x}_i$. For more flexible models, it can be harder to interpret the hidden states. Figure 5 shows an example of a **Boltzmann machine**, which is an (undirected) graphical model defined on binary variables. The observed variables $\mathbf{x}$ correspond to the data (here clamped to the digit '8'). A typical hidden state, sampled from the posterior $p(\mathbf{z}|\mathbf{x}, \hat{\boldsymbol{\theta}})$, is shown. This is an example of a **distributed representation**: the "meaning" of the state is encoded across the whole set of binary variables, which can be thought of as acting like stochastic **neurons**.

A different subset of the variables are on in each sample (rather like neural firing). Each such bit pattern $\mathbf{z}$, together with the model's parameters $\boldsymbol{\theta}$, induces a different prediction about the data, $\mathbf{x}$. Since many bit patterns can make similar predictions (just as many different visual scenes can produce the same image), it is not clear what the "right" bit pattern should be. Hence the system stochastically moves between different **interpretations** of the data.

A similar thing is presumed to happen in **human vision**: given an image $\mathbf{x}$, there are many possible interpretations (scenes) $\mathbf{z}$ that could have produced it (think of optical illusions); the brain's job is to solve the **inverse problem** of inferring $\mathbf{z}$ from $\mathbf{x}$. Since there are multiple possible answers (due to ambiguity), the brain uses prior knowledge in the form of $p(\mathbf{z})$, together with Bayes rule, to infer the posterior over scenes, $p(\mathbf{z}|\mathbf{x})$. This is sometimes called **state estimation**. See Figure 6.

In some problems, the meaning of the latent variables $\mathbf{z}_i$ is well-defined, because the user imposes a meaning on them. For example, in the Boltzmann machine in Figure 5, the nodes in the small 2×5 block on the left have been

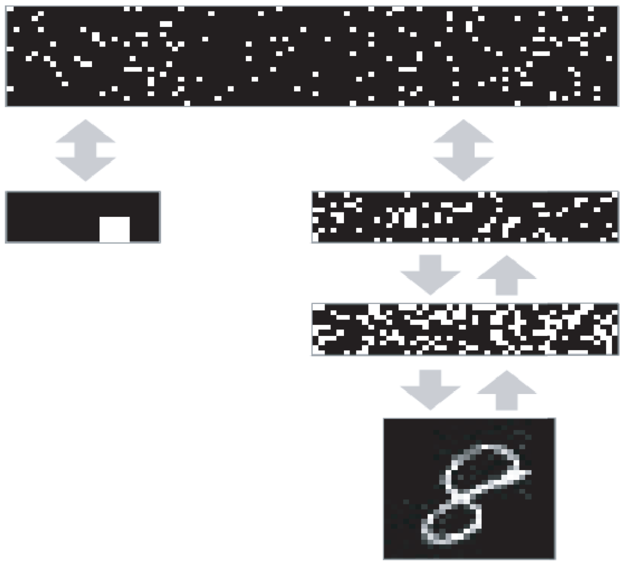


Figure 5: Visualization of the hidden state of a probabilistic model trained to generate handwritten digits. The observed data $\mathbf{x}$ is the digit '8' in the bottom right corner. The hidden variables $\mathbf{x}$ are binary random variables, structured as a series of "stacked" bipartite graphs. (This model is called a **Boltzmann machine**, and will be studied in Section **??**). We are showing a sample from the posterior $p(\mathbf{z}|vx, \hat{\theta})$. The small 2×4 rectangular block in the top left represents the class label. Currently the bit corresponding to digit 8 is activated, indicating a correct classification. Source: `http://www.cs.toronto.edu/~hinton/adi/index.htm`.

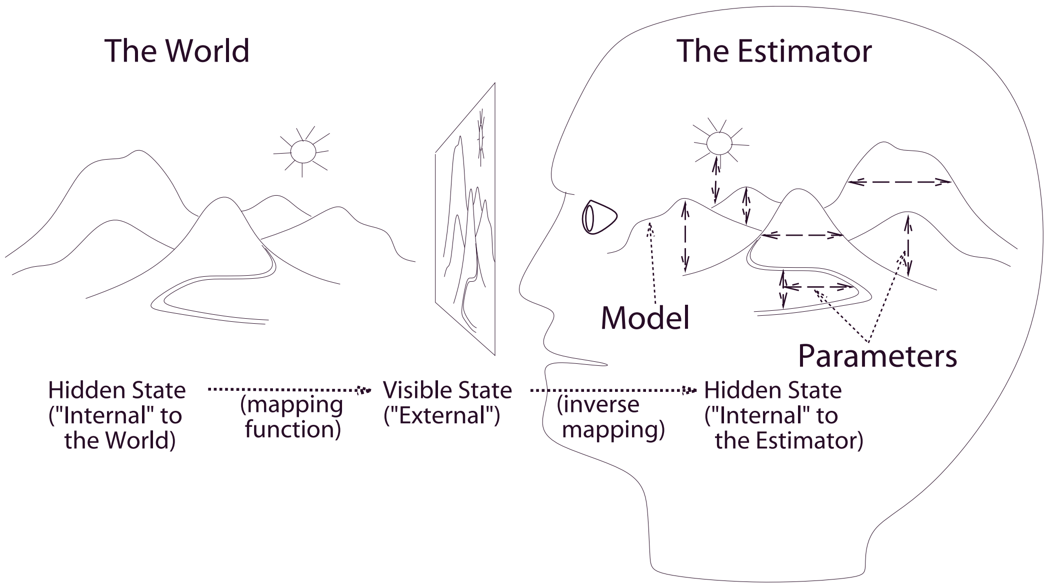

The World
The Estimator
Model
Parameters
Hidden State ("Internal" to the World)
(mapping function)
Visible State ("External")
(inverse mapping)
Hidden State ("Internal" to the Estimator)

Figure 6: Computer/human vision as inverse graphics. The world $\mathbf{z}$ generates observations $\mathbf{x}$, the system must invert this to infer $\mathbf{z}$ from $\mathbf{x}$. Since this is an ill-posed problem, the system must combine a prior $p(\mathbf{z})$, i.e., a world model, with the likelihood, $p(\mathbf{x}|\mathbf{z})$, in order to compute its posterior, $p(\mathbf{z}|\mathbf{x})$. Source: [Rao97].

engineered to represent the class label, y_i. This was done by simply **clamping** the appropriate node to its "on" value when an image is presented. For example, if $\mathbf{x}$ is set to the digit 8, we clamp the node representing class 8 to its on state; the remaining hidden nodes are **unclamped**. We then update our posterior over the parameters based on the new $\mathbf{x}, y$ pair by computing $p(\boldsymbol{\theta}|\mathbf{x}, y, D)$.

This technique of forcing some of the hidden nodes to represent labels can be seen as simply a generative model for supervised learning. But because it is a generative model, we can train it in an unsupervised way on large quantities of unlabeled data as well, by computing $p(\boldsymbol{\theta}|\mathbf{x}, D)$. There are two advantages of this. First, unlabeled data is easier to acquire in large quantities (no human is required to label the images, one simply feeds a video stream into the system). Second, because the model is required to explain all the data, and not just the labels, it is less likely to overfit, which means we can use more complex models. Put another way, there is much more information content in a video stream to constrain our parameter estimates than there is in the captions at the bottom of the video stream.[3] Consequently, this book will focus more on generative models than on discriminative models. (For a textbook that focuses on discriminative models, see [HTF01].)

References

[HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[Rao97] P. Rao. Kalman filter model of the visual cortex. *Neural Computation*, 9(4):721–763, 1997.

[3]This observation is due to Geoff Hinton (personal communication).