Minimizing Description Length in an Unsupervised Neural Network

Geoffrey E. Hinton
Department of Computer Science
University of Toronto
6 King's College Road
Toronto M5S 1A4, Canada

and

Richard S. Zemel
Computational Neurobiology Laboratory
The Salk Institute
10010 North Torrey Pines Road
La Jolla, CA 92037

April 9, 1996

Classification: Computer Sciences

Abstract

An *autoencoder* network uses a set of *recognition* weights to convert an input vector into a representation vector. It then uses a set of *generative* weights to convert the representation vector into an approximate reconstruction of the input vector. We derive an objective function for training autoencoders based on the Minimum Description Length (MDL) principle. The aim is to minimize the information required to describe both the representation vector and the reconstruction error. This information is minimized by choosing representation vectors stochastically according to a Boltzmann distribution. Unfortunately, if the representation vectors use distributed representations, it is exponentially expensive to compute this Boltzmann distribution because it involves all possible representation vectors. We show that the recognition weights of an autoencoder can be used to compute an approximation to the Boltzmann distribution. This approximation corresponds to using a suboptimal encoding scheme and therefore gives an upper bound on the minimal description length. Even when this bound is poor, it can be used as a Lyapunov function for learning both the generative and the recognition weights. We demonstrate that this approach can be used to learn distributed representations in which many different hidden causes combine to produce each observed data vector. Such representations can be exponentially more efficient in their use of hardware than standard vector quantization or mixture models.

1 Introduction

The goal of unsupervised neural network learning procedures is to capture the structure underlying a set of observations. A natural approach to this problem is to view the observations as samples drawn from a stochastic generative model whose behavior is controlled by a set of parameters. The goal of the learning is then to discover parameter values that maximize the probability of producing the observations by repeated drawings from the generative model.

Generative models may contain latent variables that are not directly observed. Different settings of these latent variables allow the same observation to be generated in different ways and all of these possible *representations* of an observation must be taken into account when computing how changes in the parameters affect the likelihood of the data. In a *mixture* model, the latent variables are booleans and exactly one of them is turned on when generating an observation. The number of possible representations is therefore equal to the number of latent variables and it is tractable to explicitly consider all possible representations of each observation. Unfortunately, mixture models are an exponentially inefficient way to represent the structure of the data if each observation is really generated by the combined effects of multiple simultaneous underlying causes. In such cases we need to fit generative models in which many of the latent variables can be turned on simultaneously. But then each observation has exponentially many possible distributed representations so it appears to be intractable to perform maximum likelihood fitting of the model's parameters.

Although maximum likelihood fitting is intractable for generative models that use distributed representations, we demonstrate an effective method of fitting these models to data using a Minimum Description Length (MDL) approach.

2 The Minimum Description Length Approach

Inspired by the algorithmic notion of complexity [1,2,3] as well as Akaike's work [4], Rissanen [5] proposed the description length of some observed data under competing models as a criterion to select between models. The essence of the MDL principle is that the best model of the data is the one that minimizes the summed length of the description of the data with respect to the model and the description of the parameters of the model. Intuitively, this can be thought of as a tradeoff between the *succinctness* of the model and its *accuracy*. To apply the principle, one decides in advance on a class of models, including a way of coding their parameters, and then searches within this class for parameters that minimize the total description length. The approach is interesting when the class of models is quite broad and the search is nevertheless tractable.

MDL can be formulated based on a communication game, in which a *sender* observes the data and must then communicate them to a *receiver*. Assuming that the data samples are quantized, we can ask how many bits must be sent to allow the receiver to reconstruct all the data perfectly using a model. Autoencoders arise when we restrict ourselves to generative models in which the latent variables are boolean and many of them combine to produce each observed vector. For an autoencoder, it is convenient to divide the total description length into three terms. An input vector is communicated to the receiver by sending the activities of the hidden units and the residual differences between the true input vector and the one that can be reconstructed from the hidden activities. There is a *representation cost* for the hidden activities and a *reconstruction cost* for the residual errors. In addition there is a one-time *model cost* for communicating the weights that are required to convert the hidden activities into the output of the net. The model cost is generally very important within the MDL framework, but in this paper we will ignore it and focus instead on the tradeoff between the representation cost and the reconstruction cost. In effect, we are considering the limit in which the model cost is negligible because the complexity of the models we are willing to consider is small compared with the amount of data.

Several important aspects of this formulation should be emphasized. First, the communication game is only a device that we use to derive an MDL objective function which can be used to train the neural network. Our goal is to develop a good generative model of a data set and we are not actually interested in communicating the data. Second, the network may have an arbitrary number of layers between the input and representation layers, and between the representation and output layers. In the rest of this paper, we refer to the network as if it only contains these three layers, but the formulation also applies to deeper networks.

2.1 Coding the residual errors

When the input vector is reconstructed from the representation that has been communicated, there will generally be residual errors and to achieve lossless communication these errors must also be communicated. In order to count the number of bits required to send the value, $x_{i,c}$, of component i of residual error vector c we must encode this value as a bit string. If the sender and the receiver have already agreed on a probability distribution that assigns a probability $p(x)$ to each possible quantized value, x, Shannon's coding theorem implies that x can be communicated at a cost that is bounded below by $-\log p(x)$ bits. Moreover, by using block coding techniques we can get arbitrarily close to this bound so we shall treat it as the true cost. For coding residual errors to within a quantization width of t it is often convenient to assume a Gaussian probability distribution with mean zero and standard deviation σ. Provided that σ is large compared with t, the probability mass of a strip of width t under a Gaussian can be approximated by the product of the strip's width and height. The cost of coding the value x is therefore:

$$C \approx -\log \frac{t}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} = -\log t \ + \ \log \sqrt{2\pi}\sigma \ + \ x^2/2\sigma^2 \tag{1}$$

To minimize this cost summed over the training set, σ^2 should be equal to the variance of x. For large training sets the additional cost of communicating the optimal value of σ is negligible, especially if we use the same value of σ for all the components of the residual error vector.

2.2 Ignoring the Representation Cost

An incorrect but very simple way of handling the representation cost is to ignore it. From an MDL perspective this is what is done in several familiar learning procedures and it is a reasonable approach if we place some restriction on the hidden layer that limits its representational capacity. Since we are also ignoring the model cost and coding the residual errors using a Gaussian, the full MDL framework then reduces to simply minimizing the squared reconstruction error.

An obvious restriction to place on the hidden layer is to use only m hidden units. If all of the units are linear, an autoencoder will then perform a version of principal components analysis. More precisely, the weight vectors of the m hidden units will span the same space as the first m principal components of the ensemble of input vectors. If additional layers of non-linear hidden units are introduced, the autoencoder will be an interesting non-linear generalization of principal components analysis.

A different restriction is to use a winner-take-all competition among the hidden units so that the one with the greatest input has an activity of 1 and the remainder have activities of 0. Even if the number of hidden units is large and they all win equally often, it will take at most $\log m$ bits on average to communicate the winner of the competition. When using a winner-take-all non-linearity it is usually more convenient to replace standard sigmoid units with radial units that compute the squared Euclidean distance between the input vector and the unit's weight vector. The winner is then the unit whose weight vector is closest to the input vector. This is called a *vector quantizer* because the input vectors are quantized into m clusters.

If we constrain the incoming and outgoing weight vectors of a radial hidden unit to be identical, backpropagation in an autoencoder is exactly equivalent to the version of competitive learning (or vector quantization or clustering) in which the incoming weights of the winning hidden unit are moved towards the input vector by an amount proportional to the Euclidean distance between the two. When we backpropagate through the winner-take-all non-linearity, the gradient of the non-linear function is zero if the winner does not change and infinite if it does, but when the winner changes there is no effect on the squared error because the new winner is exactly the same distance from the input vector. So there is never any error derivative for the input-to-hidden weights and they only change because they are tied to the hidden-to-output weights. Unfortunately, if there is more than one winner of the competition at a time, the error-derivatives of the input-to-hidden weights become infinite so gradient methods cannot be used.

Most of the unsupervised learning algorithms that have been suggested for neural networks can be seen as variations of either vector quantization or principal components analysis, so it is interesting that both these algorithms can be implemented in an autoencoder. It suggests that autoencoders may also be able to implement new algorithms that combine the best aspects of both. Vector quantization is powerful because it uses a very non-linear mapping from the input vector to the representation but weak because the representation is purely local. For the representation to contain an average of N bits of information about the input, there must be at least 2^N hidden units. Conversely, principal components analysis is weak because the mapping is linear but powerful because the representation is distributed over all the hidden units so the number of effectively different representations is exponential in the number of hidden units. Autoencoders that use binary hidden units should be able to combine distributed representations with a non-linear mapping. Moreover, if the hidden units are stochastic the autoencoder should be able to represent a probability distribution over representations, rather than just a single specific representation for each input vector. To see why it is important to have a probability distribution over representations, and to derive an appropriate objective function for training such networks we must first introduce the concept of stochastic complexity.

2.3 Coding Stochastic Representations

The description length of an input vector using a particular representation is the sum of the representation cost and reconstruction cost. We define this to be the *energy* of the representation, for reasons that will become clear later. If the prior probability of representation i is π_i and its squared reconstruction error is x_i^2 the energy of the representation is

$$E_i = -\log \pi_i - k \log t + k \log \sqrt{2\pi}\sigma + \frac{x_i^2}{2\sigma^2} \tag{2}$$

where k is the dimensionality of the input vector, σ^2 is the variance of the fixed Gaussian used for encoding the reconstruction errors and t is the quantization width.

Now consider the following situation: For a given input vector, two of the representations are equally good in the sense that they have equal energies. It may seem that we gain no advantage from having two equally good representations. We have to send some representation, so cold logic dictates that only the lowest energy representation is relevant. However, the fact that we have a choice of two representations should be worth something. It does not matter which representation we use so if we are vague about the choice of representation we should be able to save one bit when communicating the representation.

To make this argument precise consider the following communication game: In addition to communicating some input vectors, the sender must also communicate some additional unrelated information. This information has already been expressed as a bit string that cannot be further compressed, so it can be viewed as a string of random bits. Instead of first communicating the input vectors and then communicating the random bit string, the sender combines the two into a single message. Clearly, the description length of the input vectors is the total message length minus the length of the random bit string.

When confronted with two equally good representations of an input vector, the sender looks at the next bit in the random bit string and uses its value to choose one of the two. She then sends the chosen representation and its reconstruction error which costs a number of bits equal to the energy, E, of that representation of the input. The receiver first uses this message to reconstruct the input vector. Then he applies the very same method as the sender used to arrive at a choice of representations. The receiver now knows what the sender's choices were, and because he also knows what representation the sender actually chose he can infer what value the next random bit must have had. So even though E bits were actually sent, the description length attributable to the input vector is only $E - 1$ bits.*

More generally, if each of the possible representations has energy E_i and if the sender picks representations with probability p_i the expected message length will be $\sum_i p_i E_i$. The expected number of bits required to pick one representation is just the entropy of the probability distribution $-\sum_i p_i \log p_i$ and since these random bits are also communicated we must subtract off the entropy in order to get the description length attributable to the input vector

$$F = \sum_i p_i E_i + \sum_i p_i \log p_i \tag{3}$$

Note that F has exactly the form of a Helmholtz free energy. The advantage of the slightly baroque "bits-back" argument is that it allows us to give an MDL interpretation to the individual terms within the Helmholtz free energy. The free energy captures the tradeoff between the goodness of an explanation of the data and the number of alternative explanations. The probability distribution which minimizes F is

$$p_i = \frac{e^{-E_i}}{\sum_j e^{-E_j}} \tag{4}$$

The idea that a stochastic choice of representations is more efficient than just choosing the representation with the smallest value of E is an example of the concept of stochastic complexity [6] and can also be derived in other ways.

*The fact that the receiver needs to be able to reproduce the sender's probability distribution across the alternative representations for a given input vector suggests that the recognition weights must be communicated to the receiver. It is sufficient, however, to send only the generative weights. The receiver can discover the recognition weights by first reconstructing all of the input vectors for the entire training set and then running whatever learning procedure was used by the sender. So when the model cost is taken into account, only the generative weights need to be simple.

We illustrate the idea of stochastic complexity by applying it to a vector quantizer. In a standard vector quantizer, each input vector is represented by activating the hidden unit whose weight vector is closest to the input vector and the cost of communicating this winner, i, is $-log\pi_i$ where π_i is the fraction of the whole training set for which i is the winner. In a "stochastic vector quantizer" we define the energy of hidden unit i using Eq. 2 and choose hidden units stochastically according to Eq. 4. The probability distribution across hidden units is then exactly the same as for a mixture of Gaussians model:

$$p_i = \frac{\pi_i e^{-x_i^2}}{\sum_j \pi_j e^{-x_j^2}} \tag{5}$$

Using this distribution, the free energy in Eq. 3 is equal to the negative log probability of the data under the mixture of Gaussians model.

The concept of stochastic complexity is unnecessarily complicated if we are only interested in fitting a mixture of Gaussians. Instead of thinking in terms of a stochastically chosen representation plus a reconstruction error, we can simply use Shannon's coding theorem directly by assuming that we code the input vectors using the mixture of Gaussians probability distribution. However, when we start using more complicated coding schemes in which the input is reconstructed from the activities of several different hidden units, the formulation in terms of F is much easier to work with because it liberates us from the constraint that the probability distribution over representations must be the optimal one. There is generally no efficient way of computing the optimal distribution, but it is nevertheless possible to use F with a suboptimal distribution as a Lyapunov function for learning [7]. In MDL terms we are simply using a suboptimal coding scheme in order to make the computation tractable.

One particular class of suboptimal distributions is very attractive for computational reasons. In a factorial distribution the probability distribution over m^d alternatives factors into d independent distributions over m alternatives. Because they can be represented compactly, factorial distributions can be computed conveniently by a non-stochastic feed-forward recognition network.

3 Factorial Stochastic Vector Quantization

Instead of coding the input vector by a single, stochastically chosen hidden unit, we could use several different pools of hidden units and stochastically pick one unit in each pool. All of the selected units within the different pools are then used to reconstruct the input. This amounts to using several different stochastic vector quantizers† which cooperate to reconstruct the input. The number of possible distributed representations is m^d where d is the number of vector quantizers and m is the number of units within a vector quantizer.

The weights from the hidden units to the output units determine what output is produced by each possible distributed representation. Once these weights are fixed, they determine the reconstruction error that would be caused by using a particular distributed representation. If the prior probabilities of each representation are also fixed, Eq. 4 defines the optimal probability distribution over distributed representations, where the index j now ranges over the m^d possible representations.

Computing the correct distribution requires an amount of work that is exponential in d, so we restrict ourselves to the suboptimal distributions that can be factored into d independent distributions, one for each vector quantizer. The fact that the posterior distributions within the different vector quantizers are not really independent given the input will not lead to major problems as it does in mean field approximations of Boltzmann machines [8]. It will simply lead to an overestimate of the description length but this overestimate can still be used as a bound when learning the weights. Also the excess bits caused by the non-independence will force the generative weights towards values that cause the correct distribution to be approximately factorial.

†Instead of using radial units that compute a Euclidean distance within each vector quantizer, we used units that compute a scalar product. This is combined with a bias term to yield the energy that is used in Eq. 4 for normalizing the hidden activities within one vector quantizer.

3.1 Computing the Expected Representation Cost

Since the stochastic choices within the d vector quantizers are independent given the input, the expected representation cost for a given input vector is just the sum of d separate costs. For stochastic vector quantizer v, we assume that the sender communicates which unit, i, was chosen by coding the choice relative to the frequency, π_i^v, with which that unit is chosen over the whole training set. If the probability of picking unit i in vector quantizer v is p_i^v, the expected representation cost is

$$\text{Expected representation cost} \;=\; \sum_v \sum_{i \epsilon v} \left(p_i^v \log \frac{1}{\pi_i^v} - p_i^v \log \frac{1}{p_i^v} \right) \tag{6}$$

The first term inside the summations corresponds to the number of bits required to communicate the stochastically chosen representation and the second term subtracts the number of random bits that are also successfully communicated.

3.2 Computing the Expected Reconstruction Cost

To perform gradient descent in the description length given in Eq. 3, it is necessary to compute, for each training example, the derivative of the expected reconstruction cost with respect to the activation probability of each hidden unit. An obvious way to approximate this derivative is to use Monte Carlo simulations in which we stochastically pick one hidden unit in each pool. This way of computing derivatives is faithful to the underlying stochastic model, but it is inevitably either slow or inaccurate. Fortunately, it can be replaced by a fast exact method when the output units are linear and there is a squared error measure for the reconstruction. Given the probability, p_i^v, of picking hidden unit i in vector quantizer v, we can compute the expected reconstructed output y_j for output unit j on a given training case

$$y_j = b_j + \sum_v w_{ji}^v p_i^v \tag{7}$$

where b_j is the bias of unit j and w_{ji}^v is the generative weight from i to j in vector quantizer v. We can also compute the variance in the reconstructed output caused by the stochastic choices within the vector quantizers. Under the assumption that the stochastic choices within different vector quantizers are independent, the variances contributed by the different vector quantizers can simply be added.

$$V_j = \sum_v \sum_i p_i^v \left(w_{ji}^v - \sum_k w_{jk}^v p_k^v \right)^2 \tag{8}$$

The expected squared reconstruction error for each output unit is $V_j + (y_j - d_j)^2$ where d_j is the desired output. So if the reconstruction error is coded assuming a zero-mean Gaussian distribution the expected reconstruction cost can be computed exactly.‡ It is therefore straightforward to compute the derivatives, with respect to any weight in the network, of all the terms in Eq. 3.

Early in the learning when the representation vectors are randomly related to the input vectors that cause them, the network almost eliminates the representation cost by using an almost identical, high entropy distribution across representations for every input vector. So almost all the information is in the reconstruction cost. The high entropy across representations contributes additional variance to the output units, but this is minimized by using small hidden-to-output weights. As learning progresses and the representation vectors begin to capture the regularities in the inputs, it is worth using very different, low-entropy distributions for each input vector because the increase in representation cost is more than offset by the reduction in reconstruction cost.

‡Each vector quantizer contributes non-Gaussian noise and the combined noise is also non-Gaussian. But since its variance is known, the expected cost of coding the reconstruction error using a Gaussian prior can be computed exactly. The fact that this prior is not ideal simply means that the computed reconstruction cost is an upper bound on the cost using a better prior.

3.3 Simulation results

Zemel [9] presents several different data sets for which factorial vector quantization produces efficient encodings. We briefly describe two of those examples.

The weights of the network were adjusted using a conjugate gradient training procedure. The parameters π_i^v that are used for coding the representations were set to the mean value of p_i^v on the previous sweep through the training set. After a few experiments with different values, the variance of the Gaussian used to code the reconstruction errors was fixed at 0.1 for all the simulations described here.

The other variables in these experiments concern the architecture of the network, and particularly, the number of vector quantizers and the number of units in each. There is some flexibility in these choices because the learning algorithm can effectively eliminate excess units or entire vector quantizers. It does this by setting the outgoing weights to zero so that the activities of units do not effect the reconstruction cost and setting the incoming weights so that units have a constant activity level across the training set and thus have zero representation cost.

3.3.1 The Cartesian Product of Two Simple Tasks

Consider a set of 22-component input vectors which each contain two halves. Each half is drawn from a vocabulary of 13 different binary vectors that contain three 1's and eight 0's. The whole ensemble of 169 vectors is the Cartesian product of two much simpler ensembles, but a standard vector quantizer has no way of taking advantage of this factorial structure in the data.

We trained a factorial vector quantizer on this task using a hidden layer that contained 30 different vector quantizers each with 2 units. A 2-unit vector quantizer is equivalent to a single binary variable because the activity of one unit can be interpreted as the probability that that variable is 1, and the other unit's activity is the probability that the variable is 0. As anticipated, the network finds the underlying vocabulary. It assigns one vector quantizer to each of the 13 11-component vectors on either side, for a total of 26 useful features and the 4 remaining vector quantizers are effectively eliminated by the algorithm. The network can then represent each input vector fairly efficiently using two active features, but it is suboptimal because its representation fails to capture the fact that the 13 vectors are mutually exclusive on each half, so slightly more than one bit is wasted in specifying that the other 12 binary features for each half should be inactive.

We changed the network architecture to contain two vector quantizers each containing 15 units. This architecture consistently learns a solution where each vector quantizer learns to pay attention to one half of the input, and 13 of the representation units in each come to represent (i.e., respond to, and reconstruct on the output units) one of the 13 vectors. The other two units in each vector quantizer are eliminated. For each input example, one unit in each vector quantizer has an activity close to 1. If one of the input vectors is withheld from the training set, this network still represents it appropriately producing a squared reconstruction error of 0.00 on the test example. For this network, the total reconstruction and representation costs over the full training set are almost exactly equal to the entropy of the distribution over the input vectors.

3.3.2 Spline Images

The spline data set consists of 200 images of simple curves as shown in Figure 4. A network containing 4 vector quantizers, each with 6 hidden units, is trained on this data set. After training, the final outgoing weights for the hidden units are as shown in Figure 4. Each vector quantizer has learned to represent the height of the spline segment that connects a pair of control points. By chaining these four segments together the image can be reconstructed fairly accurately. For new images generated in the same way, the description length is approximately 18 bits for the reconstruction cost and 7 bits for the code. By contrast, a stochastic vector quantizer with 24 hidden units in a single competing group has a reconstruction cost of 36 bits and a code cost of 4 bits. A set of 4 separate stochastic vector quantizers each of which is trained on a different 8x3 vertical slice of the image also does slightly worse than the factorial vector quantizer (by 5 bits) because it cannot smoothly blend the separate segments of the curve together. A purely linear network with 24 hidden units that performs a version of principal components analysis has a slightly lower reconstruction cost but a much higher code cost.

4 Discussion

A natural approach to unsupervised learning is to use a generative model that defines a probability distribution over observable vectors. The obvious maximum likelihood learning procedure is then to adjust the parameters of the model so as to maximize the sum of the log probabilities of a set of observed vectors. This approach works very well for generative models, such as a mixture of Gaussians, in which it is tractable to compute the expectations that are required for the application of the EM algorithm. It can also be applied to the wider class of models in which it is tractable to compute the derivatives of the log probability of the data with respect to each model parameter. However, for non-linear models that use distributed representations it is usually intractable to compute these derivatives since they require that we integrate over all of the exponentially many representations that could have been used to generate each particular observed vector.

The MDL principle suggest a way of making learning tractable in these more complicated generative models. The optimal way to code an observed vector is to use the correct posterior probability distribution over representations given the current model parameters. However, we are free to use a suboptimal probability distribution that is easier to compute. The description length using this tractable, suboptimal method can still be used as a Lyapunov function for learning the parameters of the generative model because it is an upper bound on the optimal description length. The excess description length caused by using the wrong distribution has the form of a Kullback-Liebler distance and acts as a penalty term that encourages the recognition weights to approximate the correct distribution as well as possible. The penalty also pushes the parameters of the generative model towards values that cause the correct posterior distribution to be close to the distribution computed by the recognition weights.

There is an interesting relationship to statistical physics. Given an input vector, each possible representation acts like an alternative configuration of a physical system. The function E defined in Eq. 2 is the energy of this configuration. The function F in Eq. 3 is the Helmholtz free energy which is minimized by the thermal equilibrium or Boltzmann distribution. The probability assigned to each representation at this minimum is exactly its posterior probability given the parameters of the generative model. The difficulty of performing maximum likelihood learning corresponds to the difficulty of computing properties of the equilibrium distribution. Learning is much more tractable if we use the *non-equilibrium* Helmholtz free energy as a Lyapunov function [7]. We can then use the recognition weights of an autoencoder to compute some non-equilibrium distribution. The derivatives of F encourage the recognition weights to approximate the equilibrium distribution as well as they can, but we do not need to reach the equilibrium distribution before adjusting the generative weights that define the energy function of the analogous physical system.

In this paper we have shown that an autoencoder network can learn factorial representations by using the description length as an objective function. In related work [10] we apply the same approach to learning population codes. We anticipate that the general approach described here will be useful for a wide variety of complicated generative models. It may even be relevant for gradient descent learning in situations where the model is so complicated that it is seldom feasible to consider more than one or two of the innumerable ways in which the model could generate each observation.

Acknowledgements

We thank Peter Dayan, Yann Le Cun, Radford Neal and Chris Williams for helpful discussions. This research was supported by grants from the Ontario Information Technology

Research Center, the Institute for Robotics and Intelligent Systems, and NSERC. Geoffrey Hinton is the Noranda Fellow of the Canadian Institute for Advanced Research.

References

1. Solomonoff, R. (1964) *Information and Control* **7**, 1-22.

2. Kolmogorov, A. (1965) *Problems of Information Transmission* **1**, 1-11.

3. Chaitin, G. (1966) *Journal of the ACM* **13**, 547-569.

4. Akaike, H. (1974) *IEEE Transactions AC-19*, 716-723.

5. Rissanen, J. (1978) *Automatica* **14**, 465-471.

6. Rissanen, J. (1989) *Stochastic Complexity in Statistical Inquiry* (World Scientific Publishing Co., Singapore).

7. Neal, R., and Hinton, G. E. (1993) A new view of the EM algorithm that justifies incremental and other variants. *Submitted manuscript.*

8. Galland, C. C. (1993) *Network* **4**, 355-379.

9. Zemel, R. S. (1993) *A Minimum Description Length Framework for Unsupervised Learning.* PhD. Thesis, Department of Computer Science, University of Toronto.

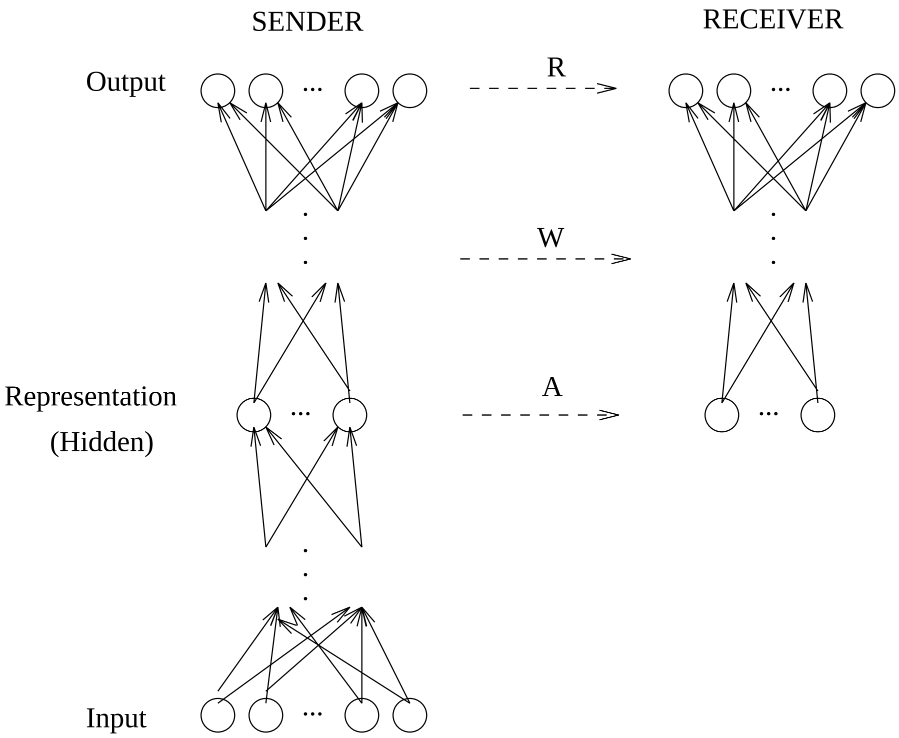
10. Zemel, R. S. and Hinton, G. E. (1994) in *Advances in Neural Information Processing Systems 6*, eds. Cowan, J., Tesauro, G., and Alspector, J. (Morgan Kaufmann, San Mateo, CA), pp. 3-10.

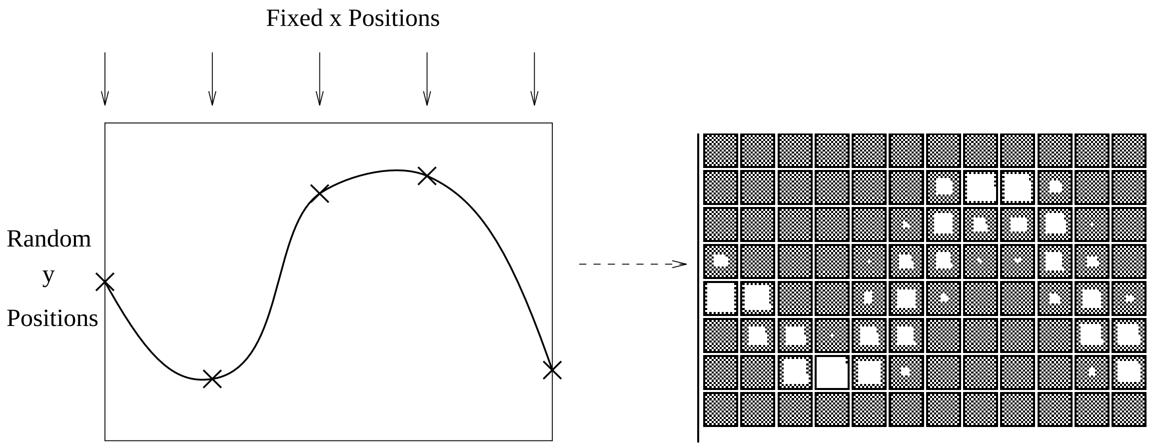
Figure Legends

Figure 1: The MDL cost function for the autoencoder is derived from the communication game illustrated above. The sender uses the autoencoder to develop representations for the training set. The receiver uses the weights, W, to convert the representation into an approximate reconstruction of the input vector and then adds the residual error vector. Ignoring the cost of communicating W, the aim of the learning is to minimize the sum of the representation cost and the reconstruction cost. The sender trains the network by back-propagating the derivatives of the reconstruction-cost from the output units, and then adding in the derivatives of the representation-cost at the representation layer.

Figure 2: Each image in the spline data set is generated by fitting a spline to 5 control points with randomly chosen y-positions. An image is formed by blurring the spline with a Gaussian. The intensity of each pixel is indicated by the area of white in the display. The resulting images are 8x12 pixels, but have only 5 underlying degrees of freedom.

Figure 3: The outgoing weights of the hidden units for a network containing 4 vector quantizers with 6 units in each, trained on the spline data set. Each 8x12 weight block corresponds to a single unit, and each row of these blocks corresponds to one vector quantizer.


SENDER
RECEIVER
Output
...
R
W
A
Representation
(Hidden)
Input


Fixed x Positions
Random
y
Positions