Channel Associative Networks for Multiple-Valued Mappings

Per-Erik Forssén
Laboratory for Computational Intelligence
Department of Computer Science
201-2366 Main Mall, BC, V6T 1Z4, Canada
perfo@cs.ubc.ca

Björn Johansson and Gösta Granlund
Computer Vision Laboratory
Department of Electrical Engineering
Linköping University, SE-581 83 Linköping
bjorn@isy.liu.se, gosta@isy.liu.se

Abstract

This paper introduces a novel artificial neural network (ANN) structure which can learn multiple valued, non-linear mappings. This is accomplished by expanding both input and output domains using a set of localised functions called channels. *In the channel space the learning problem becomes a linear mapping, which can be made sparse using a non-negative constraint. By applying this ANN to an object view recognition problem, we demonstrate that the network is able to learn efficiently under* perceptual aliasing. *This has applications for cognitive vision systems where learning has to occur at several abstraction levels simultaneously. If a subsystem is supplied with ambiguous inputs, learning will not break down, instead the subsystem will learn to pass the ambiguity to the output side, where the next subsystem can hopefully resolve it using additional context.*

1. Introduction

Perceptual aliasing [4], also known as *partial observability* [3] is a problem that occurs in learning under incomplete knowledge. It has been extensively studied in reinforcement learning [22], but it also occurs in a supervised setting when a one-to-one mapping from input to output does not exist. For many real-world learning problems it is easy to see that the inputs give evidence of what the output should be, but it is often difficult to decide whether this evidence is always non-ambiguous.

In general, perceptual aliasing occurs when we have two training samples $(\mathbf{x}_m, \mathbf{y}_m)$ and $(\mathbf{x}_n, \mathbf{y}_n)$, where the inputs $\mathbf{x}_m, \mathbf{x}_n \in \mathcal{X}$ are near identical, but the desired outputs $\mathbf{y}_m, \mathbf{y}_n \in \mathcal{Y}$ are different. Learning of a single-valued mapping $f : \mathcal{X} \mapsto \mathcal{Y}$ will in such situations be impossible. If attempted it typically results in an output which is somewhere in-between $\mathbf{y}_m$ and $\mathbf{y}_n$. This could be disastrous, e.g. if the system is an obstacle avoiding robot, and $\mathbf{y}_m$ and $\mathbf{y}_n$ are motor controls that steer the robot respectively to the left and to the right of an obstacle.

The standard solution to the perceptual aliasing problem is to add an *internal state* $\mathbf{s} \in \mathcal{S}$, and instead learn two mappings: The state estimation mapping $f_1 : \mathcal{S} \times \mathcal{X} \mapsto \mathcal{S}$, which updates the state estimate according to $\hat{\mathbf{s}}_n = f_1(\mathbf{s}_{n-1}, \mathbf{x}_n)$, and the output mapping[1] $f_2 : \mathcal{S} \mapsto \mathcal{Y}$, which estimates the response from the state estimate $\hat{\mathbf{y}}_n = f_2(\mathbf{s}_n)$. This solution works fine if we know what state representation $\mathbf{s}$ we should use, but in general, finding out the state representation is part of the learning problem at hand, see e.g. [2]. When we have to learn the state space shape, estimation of f_1 and f_2 will be quite slow, since the state space will change during learning.

In some learning problems it is possible to resolve perceptual aliasing by detecting it and adding suitable inputs, see e.g. [11]. Such approaches, in essence assume that we have a non-aliased set of percepts to choose from, which is not the case in general.

In this paper we instead propose to learn a, possibly, multiple-valued mapping $\mathcal{X} \mapsto \mathcal{Y}$, where a single input $\mathbf{x}_n$ could produce multiple outputs $\mathbf{y}_n$. We can then, if needed, learn the shape of the state space in a separate step. This has the potential of speeding up open state space learning problems by orders. When learning the mapping, we will make use of the *channel representations* of $\mathbf{x}_n$ and $\mathbf{y}_n$, $\mathbf{u}_n = \text{enc}(\mathbf{y}_n)$, and $\mathbf{a}_n = \text{enc}(\mathbf{x}_n)$, and learn a linear mapping

$$\hat{\mathbf{u}}_n = \mathbf{C}\mathbf{a}_n \,. \tag{1}$$

The *channel representation* is a way to represent single and multiple values in a unified manner, and is the topic of the next section.

2. Channel Representation

The *channel representation* [8, 1, 17] is an *encoding* of a signal value y, and an associated confidence $r \geq 0$. This is

[1] or *policy* if we do reinforcement learning

done by passing y through a set of non-negative, localised *kernel functions* $\{\mathrm{B}^k(y)\}_{k=1}^K$, and weighting the result with the confidence r. Each output signal is called a *channel*, and the vector consisting of a set of channel values

$$\mathbf{u} = \mathrm{enc}(y, r) = r \begin{pmatrix} \mathrm{B}^1(y) & \mathrm{B}^2(y) & \dots & \mathrm{B}^K(y) \end{pmatrix}^T \quad (2)$$

is said to be the *channel representation* of the signal–confidence pair (y, r), provided that the channel encoding is *injective* for $r \neq 0$, i.e. there should exist a corresponding decoding that reconstructs y and r from the channel values.

The confidence r can be viewed as a measure of reliability of the value y. When no confidence is available, it is simply taken to be $r = 1$.

Examples of suitable kernels for channel representations include Gaussians [20, 21], and B-splines [19]. Here we will use the windowed $\cos^2$-functions introduced in [17]:

$$\mathrm{B}^k(y) = \begin{cases} \cos^2(\omega(y-k)) & \text{if } \omega|y-k| < \pi/2 \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

Here the kernel index k is also the kernel centre, and ω is the kernel scale (typically set to $\pi/3$). See figure 1 for an example of a set of $\cos^2$-kernels.

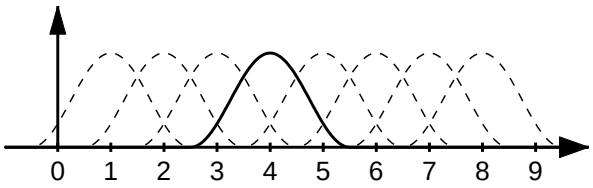


Figure 1. Regular channel arrangement. The kernel with $k = 4$ is plotted in solid, the others are dashed. All kernels have scale $\omega = \pi/3$.

The kernels we use are normally defined a priori, and this restricts us to represent values y that are *bounded*, and the bounds should be known, i.e. we should know values b_l, b_u such that

$$y \in [b_l, b_u] \cong \{y : b_l \leq y \leq b_u\} \quad \forall y\,. \quad (4)$$

Then we should also ensure that the domain of y, i.e. $[b_l, b_u]$ coincides with the represented domain of the channel set $\{\mathrm{B}^k(y)\}_1^K$. This can e.g. be achieved by a linear mapping

$$y' = k_0 + k_1 y \quad (5)$$

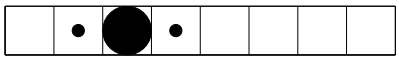
for some choice of coefficients k_0 and k_1.

2.1. Representation of Multiple Values

If we e.g. used the channel set in figure 1 to encode the value $y = 3$ with $r = 1$, we would get

$$\mathrm{enc}(3, 1) = \begin{pmatrix} 0 & 0.25 & 1 & 0.25 & 0 & 0 & 0 & 0 \end{pmatrix}^T . \quad (6)$$

This vector can also be visualised as

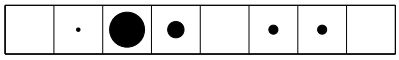


As can be seen here, the channel vector is typically *sparse*, i.e. it contains mostly zeroes. This means that the actual amount of memory needed is not as large as it may appear, since only the non-zero values need to be stored. A lot of empty space in a channel vector also allows us to optionally store more than one value, provided that the values we want to store are sufficiently different. As an example consider

$$\mathbf{u} = \mathrm{enc}(3.2, 0.75) + \mathrm{enc}(6.5, 0.25) \quad (7)$$

$$= \begin{pmatrix} 0 & 0.072 & 0.718 & 0.336 & 0 & 0.186 & 0.186 & 0 \end{pmatrix}^T , \quad (8)$$

visualised as



The interpretation of such a channel vector could be "The value 3.2 with likelihood 0.75, and 6.5 with likelihood 0.25". That is, the channel representation allows us to represent ambiguous statements, which is exactly what we would like to give as a response from a neural network, when perceptual aliasing is present.

2.2. Channel Decoding

In order to decode several signal values from a channel vector, we have to make a *local decoding*, i.e. a decoding that assumes that the signal value lies in a specific limited interval (see figure 2).

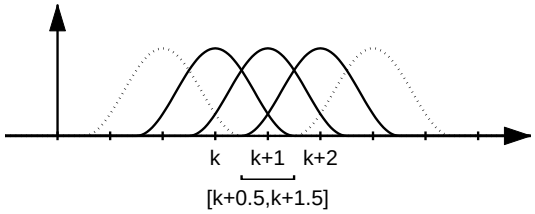


Figure 2. Interval for local decoding ($\omega = \pi/3$).

For the $\cos^2$-kernel, and a kernel scale $\omega = \pi/N$, where $N \in \{3, 4, 5 \dots\}$ the decoding becomes [7]:

$$\hat{y} = k + \frac{1}{2\omega} \arg \left[\sum_{l=k}^{k+N-1} u_l e^{\mathbf{i} 2\omega(l-k)} \right] . \quad (9)$$

The confidence can be retrieved from the sum of the channel values used in the decoding:

$$\hat{r} = \sum_{l=k}^{k+N-1} u_l \,. \quad (10)$$

In order to decode multiple valued channel vectors, we simply go through all windows $\{u_k, u_{k+1}, u_{k+2}\}$ for different values of k, decode, and check if the decoding lies within the represented interval, see figure 2.

We will denote the process of obtaining one, or possibly several value-confidence pairs by

$$\{(\hat{y}_k, \hat{r}_k)\} = \text{dec}(\mathbf{u}_k)\,. \tag{11}$$

An important aspect of this decoding scheme is that, if we encode two values y_1, and y_2, sum their channel representations, and then decode, we will obtain:

- The average $m = (y_1 + y_2)/2$ when $|y_1 - y_2| \leq 1$
- Both y_1 and y_2 when $|y_1 - y_2| \geq 2$
- Two values close to y_1 and y_2, but shifted toward the mean $(y_1 + y_2)/2$, otherwise.

This kind of representation, where two similar values are confused with their average is common in biological systems, and is called a *metameric representation* [20].

3. A Channel Associative Network

We will below describe the associative network using two simple one-dimensional examples that are of an illustrative nature, and we will concentrate on general ideas rather than giving detailed performance measurements. Although these examples do not tell the whole story, they serve to give a quick insight. Another example will be given later in the experiment section 4.

Assume that we have a set of training samples, $\{(x_n, y_n)\}_{n=1}^N$, from which we want to learn a mapping $x \mapsto y$. As the first example we consider the special case of single valued mappings, see left plot in figure 3. However, a more general and interesting class of problems is multiple valued mappings, as shown in the second example in figure 3.

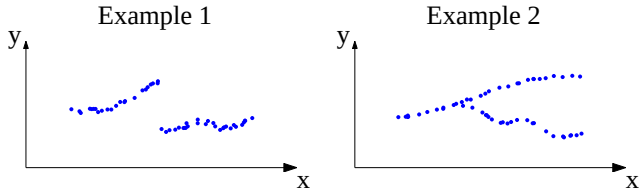


Figure 3. Two mapping examples.

Let $\mathbf{a}$ and $\mathbf{u}$ denote channel representations of x and y respectively, i.e. $\mathbf{a} = \text{enc}(x)$ and $\mathbf{u} = \text{enc}(y)$. Note that the encoding functions are in general different for the two domains, since the number of channels and the mapping (5) differs. We will refer to the vectors $\mathbf{a}$ and $\mathbf{u}$ as the *feature vector* and the *response vector* respectively, and their elements as *feature channels* and *response channels*. Furthermore, let $\mathbf{a}_n = \text{enc}(x_n)$ and $\mathbf{u}_n = \text{enc}(y_n)$. The single valued mapping problem is often solved using *kernel regression*, one example is RBF-networks [10]. This approach is basically equivalent to a linear mapping from a channel representation (possibly normalised) of the input domain x to the output domain y, i.e.

$$\hat{y} = \mathbf{c}^T \mathbf{a}\,. \tag{12}$$

The vector $\mathbf{c}$ containing the model parameters can for example be computed using the training samples $\{(\mathbf{a}_n, y_n)\}$ and least squares. The result of using this approach is shown in figure 4. We see that the model displays a ringing behaviour near the discontinuity in the first example, which is typical for linear methods (see e.g. [18]). The model naturally fails completely in the multiple valued region in the second example.

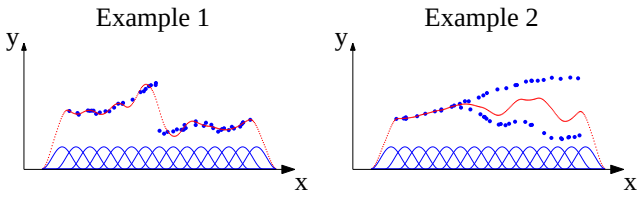


Figure 4. Results for the two examples using model (12) and evaluation points sampled with a resolution higher than for the training points. The figure also shows the positions and widths of the feature channels.

A more powerful model is a linear mapping from a channel representation of x to a channel representation of y, i.e.

$$\hat{\mathbf{u}} = \mathbf{C}\mathbf{a}\,. \tag{13}$$

The matrix $\mathbf{C}$ that contains the model parameters can again be found by solving a least squares problem,

$$\min_{\mathbf{C}} \|\mathbf{U} - \mathbf{C}\mathbf{A}\|^2\,, \tag{14}$$

where

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_N \end{pmatrix}\,, \tag{15}$$

$$\mathbf{U} = \begin{pmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_N \end{pmatrix}\,. \tag{16}$$

We will refer to $\mathbf{C}$ as a *linkage matrix* and its elements as *links*.

The channel vector $\hat{\mathbf{u}}$ can in usage mode be decoded using the scheme in section 2.2 to give one or several estimated values of y. Figure 5 shows the result using model (13) on the two examples. The performance is much better than using model (12) in both cases. Note the metameric effect (c.f. section 2.2) in the second example, when the two values are confused with their average if they lie too close to each other.

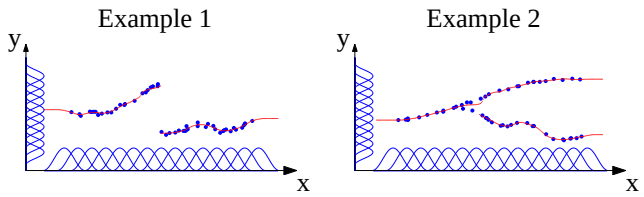


Figure 5. Results for the two examples using model (13). Only the most confident decoding is plotted in the first example, while all decodings with confidence above a certain threshold are plotted in the second example. The figure also shows the positions and widths of the feature channels and the response channels.

3.1. Monopolar Constraint

So far we have created a model which seems to have a good performance, but the cost is a large increase in model parameters, $\mathbf{C}$, leading to more memory requirements and a higher computational complexity. We propose a solution to reduce the number of parameters that is fairly simple and which has some nice properties: a *non-negative constraint on* $\mathbf{C}$, also referred to as a *monopolar constraint*. In a fuzzy rule framework this corresponds to using only positive rules, which would make the model easier to interpret [15]. Furthermore, the non-negative constraint on $\mathbf{C}$ will ensure a non-negative response vector $\hat{\mathbf{u}}$. We modify (14) into

$$\min_{\mathbf{C}\geq\mathbf{0}} \|\mathbf{U} - \mathbf{C}\mathbf{A}\|^2 . \tag{17}$$

Problem (17) is solved using the simple gradient based iterative algorithm

$$\mathbf{C}(i+1) = \max\left(\mathbf{0}, \mathbf{C}(i) - (\mathbf{C}(i)\mathbf{A} - \mathbf{U})\mathbf{A}^T\mathbf{D}\right) , \tag{18}$$

where $\mathbf{C}(0) = \mathbf{0}$ and $\mathbf{D} = \text{diag}^{-1}(\mathbf{A}\mathbf{A}^T\mathbf{1})$. The sequence (18) is sometimes called a *projected Landweber method* and the matrix $\mathbf{D}$ a *preconditioner* which can, if properly chosen (not necessarily as a diagonal matrix), allow a considerable acceleration of the ordinary gradient search, see e.g. [5, 18]. In practise, the present choice gives significant speedups compared to ordinary gradient search, see [13].

Note that the error function (17) is based on a training data set, and the error on other data may very well increase during the iterative process, sometimes referred to as *semi-convergence* or *over-fitting*. We use early termination as a means for regularization. Semiconvergence is also occuring in the sense that we optimize using an error function in the channel space rather than in the original space, which means that the error in the original space may also increase during iterations. Further details, such as convergence of (18) are discussed and proven in [13, 12].

The constrained solution (17) is *sparse*, i.e. it contains much fewer non-zero elements than the unconstrained solution (14). To further illustrate the network, figure 6 shows the matrices $\mathbf{A}$, $\mathbf{U}$ from the second example, the solution $\mathbf{C}$ from problem (17), and the estimated responses $\hat{\mathbf{U}} = \mathbf{C}\mathbf{A}$.

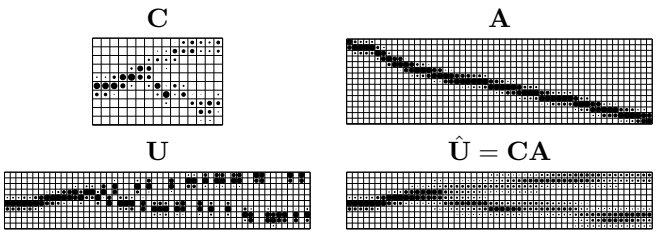


Figure 6. The matrices $\mathbf{A}$ and $\mathbf{U}$ in example two, the solution $\mathbf{C}$ from (17), and the estimate $\hat{\mathbf{U}}$.

The performance in the two examples is visually unaffected by the use of the monopolar constraint (therefore not shown). In fact, the constraint can actually improve the performance in many cases. As shown in [12, 13], the monopolar constraint actually gives an upper limit on the linkage matrix values which makes it a form of regularisation.

3.2. Model Aspects

As always, models contain a number of user parameters that affect the result. In this case we have to specify channel widths and positions for both input and output domain, which controls the behaviour of the network. For example, the distinction between a discontinuity and a steep slope is not well defined when we only have access to a set of samples. This is mainly controlled by the distance and overlap between the response channels, see figure 7.

We emphasise that an increase in the number of response channels does not cause an explosion in the number of used links. In [7] it is demonstrated that for the case of a single valued mapping the number of links remains fairly stable at approximately two to three times the number of links required for the simpler model (12). This is a direct consequence of the monopolar constraint. For the mappings in figure 5, the number of links are 3.1 and 4.1 times higher than the corresponding mappings in figure 4.

3.3. Discussion

To summarise the associative network, the basic idea is quite simple if we ignore all the surrounding notation; a non-linear, single or multiple valued mapping $x \mapsto y$ can be effectively implemented as a simple linear mapping between a channel representation of each domain, $\mathbf{a} \mapsto \mathbf{u}$. The linkage matrix $\mathbf{C}$ that defines the mapping can be made sparse due to a monopolar constraint.

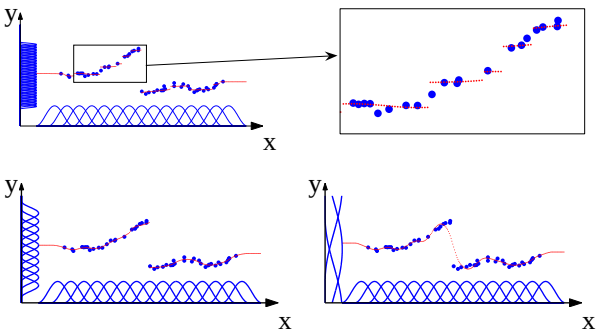


Figure 7. Model behaviour for varying width of the response channels. The width controls the distinction between a discontinuity and a steep slope. Note the "staircase effect" for the very dense response channels in top graph, and the loss of discontinuity preservation for very large channels in the bottom right graph.

Computer vision problems often involve high-dimensional mappings $\mathbf{x} \in \mathbb{R}^{M_x} \mapsto \mathbf{y} \in \mathbb{R}^{M_y}$, but the fundamental idea is similar to the one described above, namely, to decompose the space and represent the problem by locally linear problems. The feature channels are usually not of the simple kind described above, but of a very complex nature and usually not regularly positioned. The feature channels a^h will in a typical case describe local properties of a signal, or local properties in an image of an object, such as local orientation, local curvature, colour, etc.

4. Experiments

As a practical illustration of a multiple-valued mapping we will use the mapping from appearance of a 3D object, to its orientation. When two views of an object have the same appearance, a mapping from appearance to view angle should output both view angles. We will use images from the COIL-100 database [16]. This database (see `http://www.cs.columbia.edu/CAVE`) contains 100 objects of varying shape and colour. Each object has been placed on a motorised turntable, and has been photographed at 72 different views, 5° apart, and stored as 128×128 pixel RGB bitmaps (see figure 8). The COIL-100 database has previously [16, 23] been used to evaluate object recognition systems, but we will instead use it as an example of a situation where multiple-valued mappings can be used.

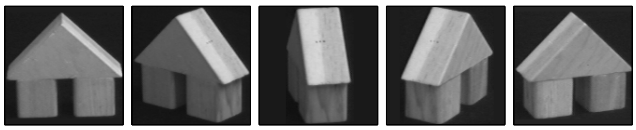


Figure 8. Five views of COIL-100 object #51.

4.1. Network Inputs

The inputs to the associate network are computed from responses from the *first order rotational symmetry operator* s_1 [14, 12] designed to find high curvature points from gradient images in double-angle representation. The s_1 operator responses are well suited to associative learning, since they are sparse and relatively robust.

The gradient field which is the input to the s_1 operator is computed with Gaussian derivatives of $\sigma = 1.0$. The s_1 operator itself is computed with a Gaussian window of $\sigma = 5.5$, and the result is subsampled to a 32×32 grid. The output $\mathbf{s_1}(\mathbf{x})$ is a complex number in each position, $\mathbf{x}$, where the argument represents the direction of the curvature. We then compute four non-negative feature maps from $\mathbf{s_1}(\mathbf{x})$ by separating each complex number into real, and imaginary parts, and further into positive and negative parts. This gives us in total $32 \times 32 \times 4 = 4096$ features per view. The s_1 response magnitude, $|\mathbf{s_1}(\mathbf{x})|$, and the four non-negative components of $\mathbf{s_1}(\mathbf{x})$, are shown in figure 9.

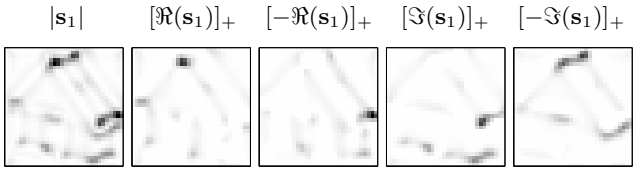


Figure 9. Left to right: Magnitude $|\mathbf{s_1}|$ for the second view in figure 8, and the four non-negative components of $\mathbf{s_1}$.

4.2. Network Outputs

The desired response from the associative network is the view angle ϕ. The view angle is channel encoded with 12 channels placed along the periodic interval $[0^\circ, 360^\circ]$, as shown in figure 10. Note that the three channels at the $0^\circ \Leftrightarrow 360^\circ$ discontinuity in figure 10 continue on the other side. This is accomplished by modifying (3) to:

$$\mathrm{B}^k(y) = \begin{cases} \cos^2(\omega d(y,k)) & \text{if } \omega d(y,k) < \pi/2 \\ 0 & \text{otherwise.} \end{cases} \quad (19)$$

By setting $d(y,k) = |y-k|$ we again get (3), but we will now instead use a periodic distance:

$$d_K(y,k) = \min(\mathrm{mod}(y-k,K), \mathrm{mod}(k-y,K)) \,. \quad (20)$$

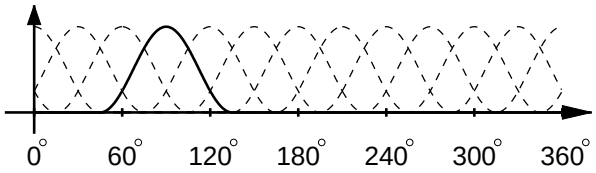


Figure 10. Modular $\cos^2$ channel arrangement for view angle representation. Channel centred at $90°$ is plotted in solid, others are dashed.

4.3. Learning Object View

We now supply features and responses for all 72 available views of object #51 (see figure 8) to the optimisation algorithm (18). Since this object is symmetric, its appearances from opposite sides are near identical, and the optimisation algorithm will thus get samples $(\mathbf{a}_m, \mathbf{u}_m)$, and $(\mathbf{a}_n, \mathbf{u}_n)$, where the features $\mathbf{a}_m$, $\mathbf{a}_n$ are very similar, but the desired responses $\mathbf{u}_m$, $\mathbf{u}_n$ are quite different, i.e. we have the perceptual aliasing problem mentioned in section 1. Figure 11, left, shows what the optimised linkage matrix $\mathbf{C}$ will output when it is again fed with the feature samples. Compare this with the desired responses in figure 10.

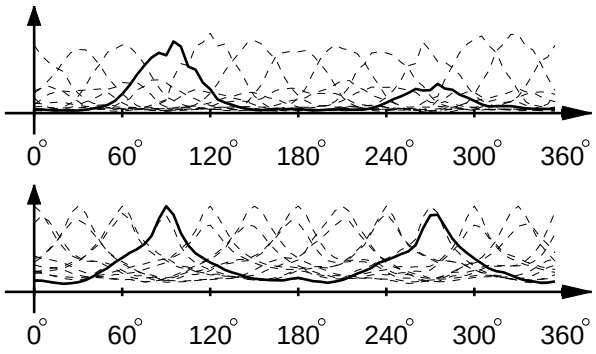


Figure 11. Top: Obtained responses when using object #51. Channel centred at $90°$ is plotted in solid, others are dashed. Bottom: Feature correlation functions for object #51. Correlation for feature at $90°$ is plotted in solid, others are dashed.

As can be seen in figure 11, left, the single peaks in the desired response functions have been split into two peaks with lower amplitude. Most of the energy ended up in the desired place, but a fraction ended up at the samples where the directly opposing view was presented. The network response thus reflects the high degree of similarity between the two views.

Note however that the obtained response is quite different from a direct correlation of the feature vectors. Consider the feature correlation function

$$r_k(n) = \frac{\mathbf{a}_k^T \mathbf{a}_n}{||\mathbf{a}_k|| ||\mathbf{a}_n||} \,. \tag{21}$$

Figure 11, bottom is a plot of the 12 different $r_k(n)$ functions corresponding to the responses in figure 10, and 11, top. By comparing the top and bottom plots in figure 11, we can see that the feature correlation functions have nearly equal strength at the opposite views, while the associative network manages to suppress a great deal of the features that also correlate with the opposite view. The associative network can be said to try its best to use only features that uniquely describe a certain view. When this is not possible, it will pick those features that have the least amount of activity outside the desired response region. This is what makes the associative network avoid the DC offset present in the feature correlation functions.

4.4. Decoding Performance

The output functions from figure 10, and figure 11 are also visualised in a different way in the two first plots in the top row of figure 12. The individual outputs are quite noisy, but the noise is significantly reduced when we combine the responses using the local decoding (9), as can be seen in figure 12, top right. Here we can also see that the extra output, although slightly noisier than the specified one, lies consistently at an offset of about $180°$. Mean absolute errors (MAE) for the two decodings are $0.88°$, and $3.58°$ respectively. Compare this with the $5°$ view distance. The bottom row of figure 12 shows the result of the same experiment, when only every fourth sample is shown to the associative network (i.e. $20°$ view distance). As can be seen, the result is similar, although slightly more noisy. The MAE values for these curves are $1.42°$ and $4.59°$ respectively.

4.5. Other objects

We have also applied the associative networks to the other objects in the COIL-100 database. The decoded responses for a selection of other objects are shown in figure 13. We have also included the corresponding feature correlation functions for comparison. The objects in the database can be divided into the following categories:

1. Objects where all views are sufficiently different. This is case for e.g. the frog in the first row.
2. Objects with a single symmetry, which is not directly the opposite view. See e.g. tin can in second row.
3. Objects with three-fold symmetry. See e.g. the tin can in the third row.

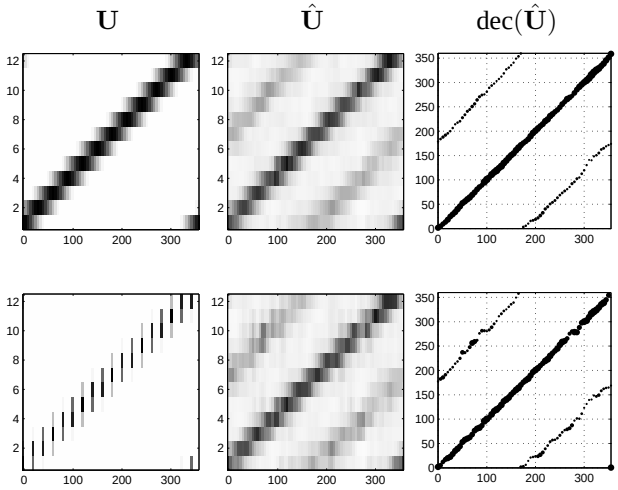


Figure 12. Learning results for object #51. Top row: results for training with all samples. Bottom row: results for training with every fourth sample. Left: The desired response functions from figure 10 shown as rows in an image. White is 0, black is 1. Centre: The obtained responses from figure 11, left. Right: Decoded network responses. Each valid decoding $(\hat{y}, \hat{r})$ is visualised as a dot at $(y, \hat{y})$ with a size proportional to $\hat{r}$ (only decodings with $\hat{r} > 0.15$ are shown).

4. Objects with a four-fold symmetry. See e.g. sandwich in fourth row.
5. Objects where there is confusion along part of the trajectory, but where some views are sufficiently different. See e.g. the dental-floss box in the fifth row.
6. Objects where all views are very similar. See e.g. the plastic jar in the sixth row.

5. Discussion

This paper has introduced a novel neural network architecture that is able to learn multiple valued mappings. The aim of the paper has been to demonstrate what the network can do, rather than presenting a complete application. Instead of directly learning the desired response function, the network learns a set of channel functions, which can be decoded to one or several output values with associated confidences. We have shown that the network automatically switches between a single response and several depending on the problem at hand. The price of this is a higher memory requirement. When the network learns single valued mappings, the memory requirements is roughly 2 to 3 times higher than in a conventional kernel regression network.

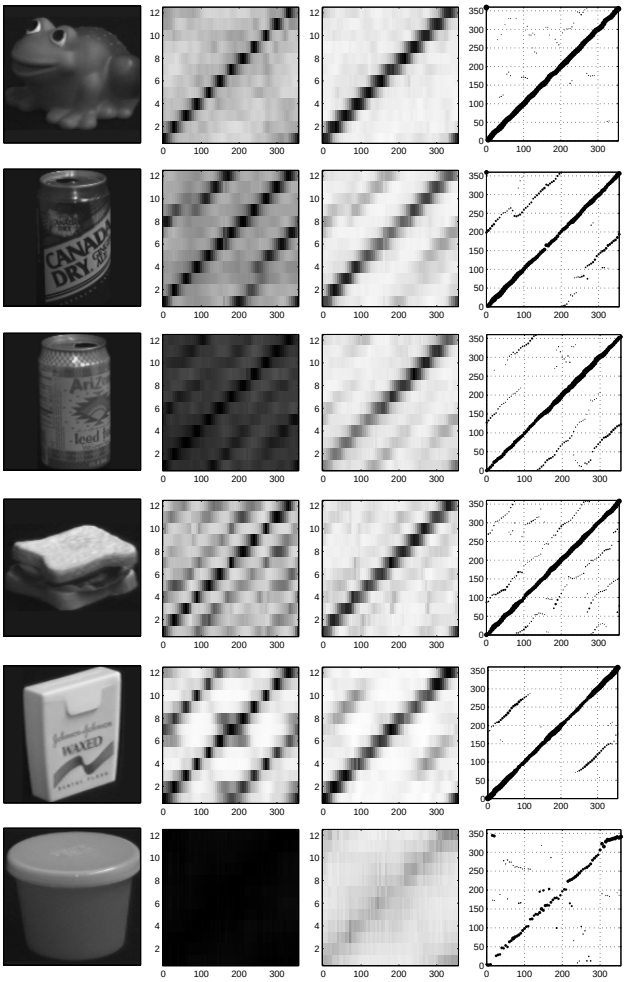


Figure 13. Results for a selection of COIL-100 objects. Each row describes a different object. Columns left to right: Object at $320°$ view. Correlation functions, Network responses, Decoded responses (only decodings with $\hat{r} > 0.15$ are shown).

The experiment section demonstrates how multiple valued mappings can be used to learn under perceptual aliasing. The experiments are meant to give an idea of what kinds of learning problems can be solved with channel associative networks. We have compared the network to feature correlation and showed that the behaviour is qualitatively different. The network can, in contrast to the correlation, select the appropriate features to model the desired responses. When only ambiguous features are available, the network will select the least ambiguous features and use these to propagate the ambiguity to the output. We also measured the accuracy of the obtained mapping, and demonstrated the

ability of the network to generalise to novel samples.

We believe that the field of cognitive vision can benefit from the use of channel associative networks, since cognitive vision often requires adaptation at several abstraction levels simultaneously. The associative networks also allow segmentation of large complex learning problems into several smaller ones without requiring that each mapping is always non-ambiguous. In fact, we have already employed the networks to a cognitive vision task in [6]. Here we used the networks to learn a mapping from local image patches to relative object location, orientation, and object type. For such a mapping it is advantageous to have the option of multiple outputs, since several objects can have near identical appearance locally. We resolved the resultant ambiguity on the output side by combining the responses from several patches using a clustering algorithm. Earlier, we also employed the multiple response property of the network for learning in 3D-object recognition [9], where a multiple valued output indicated the locations of the multiple objects present in the scene.

Acknowledgements

This work was supported by the Swedish Research Council through grants for the projects *Active Exploration of Surroundings and Effectors for Vision Based Robots* and *A New Structure for Signal Processing and Learning*, and the European Community through EC Grant IST-2003-004176 COSPAL. It reflects only the authors' views and the Community is not liable for any use that may be made of the information contained therein.

References

[1] M. Borga. *Learning Multidimensional Signal Processing*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 1998. Dissertation No 531, ISBN 91-7219-202-X.

[2] R. I. Brafman, G. Shani, and S. E. Shimony. Partial observability under noisy sensors–from model-free to model-based. In *Workshop on Rich Representations for Reinforcement Learning*, August 2005. http://www.cs.waikato.ac.nz/∼kurtd/rrfrl/.

[3] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI-94*, volume 2, pages 1023–1028, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[4] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *National Conference on Artifi cial Intelligence*, pages 183–188, 1992.

[5] B. Eicke. Iteration methods for convexly constrained ill-posed problems in Hilbert spaces. *Numerical Function Analysis and Optimization*, 13(5&6):413–429, 1992.

[6] M. Felsberg, P.-E. Forssén, A. Moe, and G. Granlund. A cospal subsystem: Solving a shape-sorter puzzle. In *AAAI Fall Symposium: From Reactive to Anticipatory Cognitive Embedded Systems*, number FS-05-05 in AAAI Technical Report Series, pages 65–69, Virginia USA, November 2005.

[7] P.-E. Forssén. *Low and Medium Level Vision using Channel Representations*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, March 2004. Dissertation No. 858, ISBN 91-7373-876-X.

[8] G. H. Granlund. The complexity of vision. *Signal Processing*, 74(1):101–126, April 1999. Invited paper.

[9] G. H. Granlund and A. Moe. Unrestricted recognition of 3-D objects for robotics using multi-level triplet invariants. *Artifi cial Intelligence Magazine*, 25(2):51–67, 2004.

[10] S. Haykin. *Neural Networks–A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 1999. ISBN 0-13-273350-1.

[11] S. Jodogne and J. Piater. Interactive learning of mappings from visual percepts to actions. In *22nd ICML*, Bonn, Germany, 2005.

[12] B. Johansson. *Low Level Operations and Learning in Computer Vision*. PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, December 2004. Dissertation No. 912, ISBN 91-85295-93-0.

[13] B. Johansson, T. Elfving, V. Kozlov, Y. Censor, P.-E. Forssén, and G. Granlund. The application of an oblique-projected landweber method to a model of supervised learning. *Mathematical and Computer Modelling*, 43:892–909, 2006.

[14] B. Johansson and G. Granlund. Fast selective detection of rotational symmetries using normalized inhibition. In *6th ECCV*, volume I, pages 871–887, Dublin, Ireland, June 2000.

[15] B. Kosko. *Fuzzy Engineering*. Prentice Hall, 1996. ISBN 0-13-124991-6.

[16] S. K. Nayar, S. A. Nene, and H. Murase. Real-Time 100 Object Recognition System. In *Proc. of ARPA Image Understanding Workshop*, 1996.

[17] K. Nordberg, G. Granlund, and H. Knutsson. Representation and Learning of Invariance. In *1st ICIP*, Austin, Texas, November 1994. IEEE.

[18] M. Piana and M. Bertero. Projected Landweber method and preconditioning. *Inverse Problems*, 13:441–463, 1997.

[19] H. Scharr, M. Felsberg, and P.-E. Forssén. Noise adaptive channel smoothing of low-dose images. In *CVPR Workshop: Computer Vision for the Nano Scale*, June 2003.

[20] H. Snippe and J. Koenderink. Discrimination thresholds for channel-coded systems. *Biological Cybernetics*, 66:543–551, 1992.

[21] H. Spies and P.-E. Forssén. Two-dimensional channel representation for multiple velocities. In *13th SCIA*, LNCS 2749, pages 356–362, Gothenburg, Sweden, June-July 2003.

[22] R. S. Sutton and A. G. Barto. *Reinforcement Learning, An Introduction*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-19398-1.

[23] M.-H. Yang, D. Roth, and N. Ahuja. Learning to Recognize 3D Objects with SNoW. In *6th ECCV*, volume I, pages 439–454, Dublin, Ireland, June 2000.