We propose an Artificial Neural Net (ANN) architecture for discovering common hidden variables and for learning of invariant representations through synchronicity, coincidence and concurrence. In the common variable discovery problem, the ANN uses measurements from two distinct sensors to construct a representation of the common hidden variable that is manifested in both sensors, and discards sensor-specific variables. In the invariant representation learning problem, the network uses multiple observations of objects under transformations to construct a representation which is invariant to the transformations. Unlike classic regression problems, the network is not presented with labels to learn, instead, it must infer a proper representation form the data; unlike classic signal processing problems, the algorithm is not given the invariant features to compute, and it must discover proper features.

Common Variable Discovery and Invariant Representation Learning using Artificial Neural Networks

Uri Shaham[†] and Roy R. Lederman[‡],

Technical Report YALEU/DCS/TR-1506

[†] Department of Statistics, Yale University, New Haven CT 06511
[‡] Applied Mathematics Program, Yale University, New Haven CT 06511

Keywords: *common variable, hidden variable, invariance learning, equivalence learning, representation learning, coincidence modeling, neural networks*

1 Introduction

Measurements of phenomena have many sources of variability; in a given context, some of the variability is of interest and some is superfluous. Often, these sources of variability are modeled, and the measurements are processed to remove the unwanted sources of variability; for example, in the processing of audio signal, filters are used to suppress some frequency components in order to best recover a signal in other frequencies. In other scenarios, the model does not allow to recover the original signal, but there is a representation that captures the similarity between two instances of the signal; for example, the proper rotation-invariant transforms allow to estimate whether two images are identical up to arbitrary rotation although the original image is known only up to rotation.

When the model for data generation is unknown, but there are examples for which the values of the relevant variable are known, the problem of recovering the relevant source of variability is a regression problem. In other words, if we have a dataset of measurements, each labeled with the true value of the underlying variable X that we would like to recover, we can potentially use some method of regression to construct an estimator for X.

In this manuscript we are interested in a case were the phenomenon is not modeled and the sources of variability are unknown; in other words, we are interested in recovering an unknown variable from unlabeled measurements of nonlinear functions of a mixture of relevant and superfluous variables. Clearly, we cannot expect to construct a filter or perform regression in the absence of labeled data or a model. Instead, we use synchronicity, coincidence and concurrence as additional sources of information which reveal the structure of the phenomenon; we assume two (or more) sets of measurements that share the same underlying "relevant" variable, to which we refer as *the common variable*, but each having its own superfluous variables. Our goal is to obtain a parametrization of the "relevant" variable, i.e. a representation of the data that depends only on the variable of interest, and that is invariant to the superfluous variables; in this manuscript we introduce an Artificial Neural Net (ANN) architecture and an algorithm to obtain this goal.

We discuss several formulations of the problem and demonstrate that the the approach is applicable to *common variable learning* problem, to *equivalence class learning*, and to *invariant representation learning.*

The organization of this manuscript is as follows. In Section 1 we formulate the common variable discovery problem and present an example to illustrate the problem. In Section 2 we present an ANN algorithm for the common variable discovery problem. In Section 3 we discuss some of the properties of the maps learned by the ANN. We also discuss the invariant representation learning problem and its relation to the common variable discovery problem via the equivalence learning problem. In Section 4 we present experimental results and discuss some of the properties illustrated by these results. Brief conclusions are presented in Section 5.

1.1 Illustrative toy problem

In this section, we present a toy example (adapted from [1]) to demonstrate the motivation for common variable learning. For simplicity, we chose an example that is easy to model and visualize, however, we note that the algorithm is not "aware" of the model, so the concepts illustrated in this example apply to more complex scenarios, where the user is not able to construct a model for the data.

We consider the experimental setup in Figure 1, where three objects, Yoda (a green action figure), a bulldog and a bunny are placed on spinning tables and each object spins independently of the other objects. Two cameras are used to take simultaneous snapshots: Camera 1, whose field of view includes Yoda and the bulldog, and Camera 2, whose field of view includes the bulldog and the bunny.

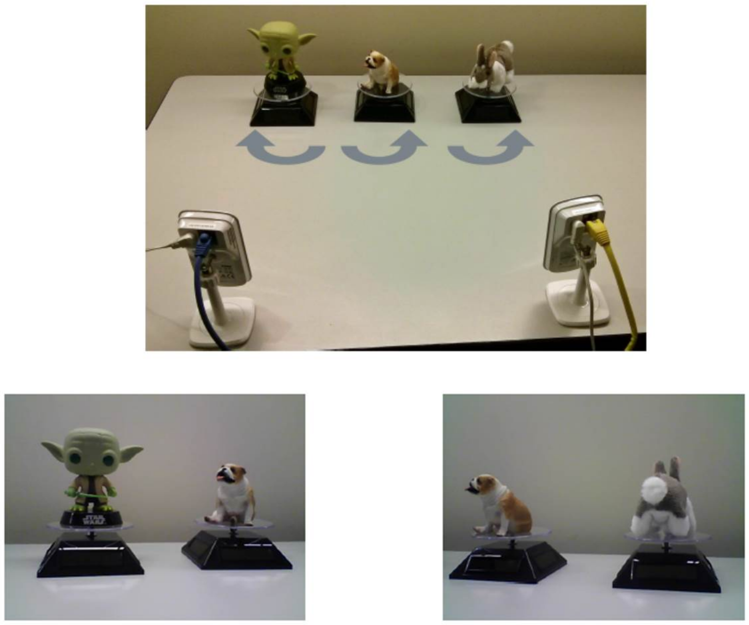


Figure 1: Top: Yoda, the bulldog and the bunny on spinning tables and the two cameras. Bottom left: a sample snapshot taken by Camera 1 at a given point in time. Bottom right: a snapshot taken at the same time by Camera 2.

In this setting, the rotation angle of the bulldog is a common hidden variable which we will denote by X; this is the common variable is manifested in the snapshot taken by both cameras. The rotation angle of Yoda, which we will denote by Y, is a sensor-specific source of variability manifested only in snapshots taken by Camera 1, and the rotation angle of the bunny, which we will denote by Z, is a sensor-specific source of variability manifested only in Camera 2. The three rotation angles are "hidden," in the sense that they are not measured directly, but only through the snapshots taken by the cameras. Given snapshots from both cameras, our goal is to obtain a parametrization of the "relevant" common hidden variable X, i.e., the rotation angle the bulldog, and ignore the “superfluous” sensor-specific variables Y and Z.

If the model were known, we would crop the images from Camera 1 so that Yoda is not visible; this would give us a representation of X, which we could process further. Alternatively, if some of the images had a label of the true value of X, we could conceivably construct an estimator for X. Here we have neither a model, nor labeled samples, but we know that the snapshot taken by Camera 1 has the same value of X as the snapshot taken at the exact same time by Camera 2.

The goal of the ANN architecture described in Section 2 is to learn a representation that depends only on the state of the common hidden variable, discarding the sensor specific variables. Optional dimensionality reduction methods, described in Section 2.6.3, are used to simplify the representation and to visualize it. An example of a representation obtained by the algorithm, visualized using these dimensionality reduction methods, is presented in Figure 2; each point in the figure corresponds to a snapshot; the color of each point corresponds to the value of the common variable X, i.e. the rotation angle of the bulldog. Figure 2 demonstrates that all the snapshots with the same value of X have been represented in essentially the same way; in other words, all the points with the same color have been mapped to approximately the same place.

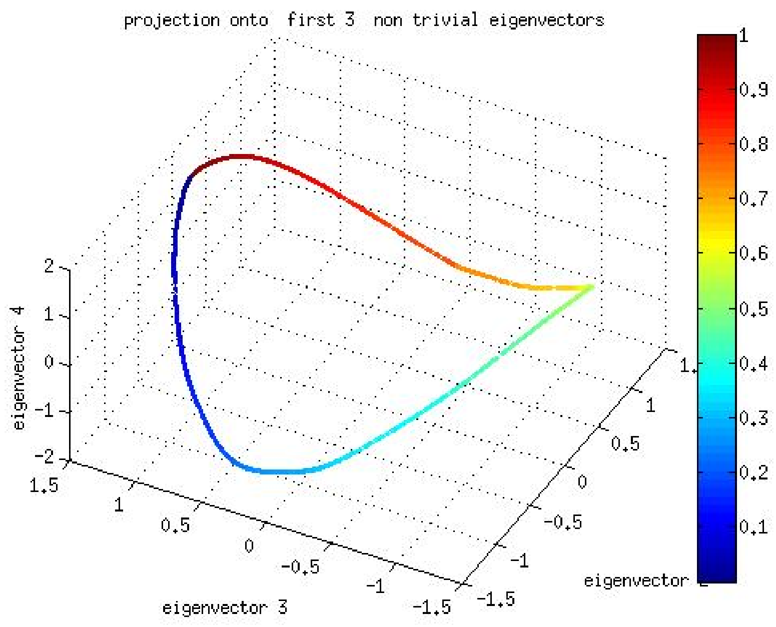


Figure 2: A representation of snapshots taken by Camera 1. Each point corresponds to a snapshot; the color of each point corresponds to the value of the common variable X.

1.2 Problem formulation: common variable discovery

In this section we present a more formal definition of the common variable discovery problem. We consider three hidden random variables $(X, Y, Z) \sim \pi_{x,y,z}(X, Y, Z)$, from the (possibly high dimensional) spaces $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, where, given X, the variables Y and Z are independent.

We have access to these hidden variables through two observable random variables

$$S^{(1)} = g_1(X, Y) \tag{1}$$

and

$$S^{(2)} = g_2(X, Z) \tag{2}$$

where g_1 and g_2 are bi-Lipschitz. We denote the range of g_1 and g_2 by $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$, respectively; these ranges may be embedded in a high dimensional space. We refer to the random variables $S^{(1)}$ and $S^{(2)}$ as the measurement in Sensor 1 and the measurement in Sensor 2, respectively.

In our toy example in Section 1.1, X is the rotation angle of the bulldog, Y is the rotation angle of Yoda, and Z is the rotation angle of the bunny; these angles are not measured directly, but through the snapshots taken by the two cameras. Sensor 1, which is Camera 1, produces the snapshot $S^{(1)}$, which is a function $S^{(1)} = g_1(X, Y)$ of the direction of the bulldog and the direction of Yoda. Sensor 2, which is Camera 2, produces the snapshot $S^{(2)} = g_2(X, Z)$ which is a function of the direction of the bulldog and the direction of the bunny.

The i-th realization of the system consists of the hidden triplet (x_i, y_i, z_i) and the corresponding measurements $(s_i^{(1)}, s_i^{(2)})$; while x_i, y_i and z_i are hidden and not available to us directly, $s_i^{(1)} = g_1(x_i, y_i)$ and $s_i^{(2)} = g_2(x_i, z_i)$ are observable. We note that both $s_i^{(1)}$ and $s_i^{(2)}$ are functions of the same realization x_i of X. Our dataset is composed of n pairs of corresponding measurements $\left\{(s_i^{(1)}, s_i^{(2)})\right\}_{i=1}^{n}$, which are functions of the n hidden realizations of the system $\{(x_i, y_i, z_i)\}_{i=1}^{n}$.

In the toy experiment, the pair of measurements $(s_i^{(1)}, s_i^{(2)})$ is a pair of snapshots taken simultaneously by the two cameras, so that the bulldog is in the same state in both $s_i^{(1)}$ and $s_i^{(2)}$. Interestingly, the two cameras observe the bulldog from different directions, so although the bulldog is in the same state x_i when the two images are taken, it does not look the same to the two cameras, as illustrated in the two simultaneous snapshots shown in Figure 1.

Ideally, we would like to construct a function $\phi : \mathcal{S}^{(1)} \to \mathcal{X}$ that recovers x from $g_1(x, y)$, so that for every $x \in \mathcal{X}$, and every $y \in \mathcal{Y}$, we would have $x = \phi(g_1(x, y))$. However, since $\mathcal{X}$ and g_1 are unknown, we cannot expect to recover x precisely, and we are interested in a function $f_1 : \mathcal{S}^{(1)} \to \mathbb{R}^d$ that recovers x up to some scaling and bi-Lipschitz transformation. In particular, we require that for all $x \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$

$$f_1(g_1(x, y)) = f_1(g_1(x, y')) \tag{3}$$

and for all $x \neq x' \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$

$$f_1(g_1(x, y)) \neq f_1(g_1(x', y')). \tag{4}$$

In the toy example, it is clear that the X cannot be recovered exactly because, for example, there is no reason to prefer the range $[0, 360)$ to the range $[0, 2\pi)$ or to determine the phase of the rotation. However, the reduced representation obtained by the algorithm and presented in Figure 2 is an example of a simple representation of X up to scaling and bi-Lipschitz transformation.

1.3 A neural network approach

The purpose of this section is to provide a brief overview of the architecture of the neural network proposed in this manuscript. The detailed description is found in Section 2.

We construct two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ which accept samples from $\mathcal{S}^{(1)}$ and $\mathcal{S}^{(2)}$, respectively, with both networks having outputs in $\mathbb{R}^d$. Once the networks are trained, $\mathcal{N}_1$ implements our proposed function f_1 while $\mathcal{N}_2$ implements a similar function on $\mathcal{S}^{(2)}$. The two networks may have different numbers of layers and otherwise different configurations; however the two networks have the same number of output units.

In order to train the networks, we form a joint network $\mathcal{N}$ by connecting $\mathcal{N}_1$ and $\mathcal{N}_2$ to a single output unit, which computes the difference between the outputs of the two networks; the output layers of $\mathcal{N}_1$ and $\mathcal{N}_2$ are therefore hidden layers of $\mathcal{N}$. The joint network is then trained as described in Section 2.4; the training of $\mathcal{N}$ implicitly includes training its components $\mathcal{N}_1$ and $\mathcal{N}_2$. The architecture of the joint network is illustrated in Figure 3.

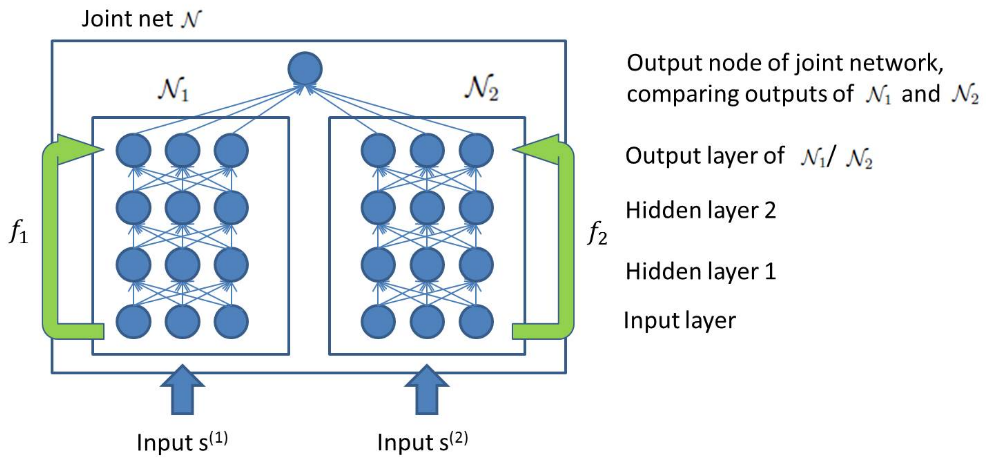

Joint net $\mathcal{N}$
$\mathcal{N}_1$
$\mathcal{N}_2$
Output node of joint network, comparing outputs of $\mathcal{N}_1$ and $\mathcal{N}_2$
Output layer of $\mathcal{N}_1$/ $\mathcal{N}_2$
Hidden layer 2
Hidden layer 1
Input layer
f_1
f_2
Input $s^{(1)}$
Input $s^{(2)}$

Figure 3: A diagram of the network structure.

1.4 Related work

Several works focus on neural nets that learn representations of inputs that are measured via two sources (e.g., [2, 3]), possibly of different modalities, for example audio and video, or images and texts. However, these works mostly deal with learning cross modality and single modality representation, while we aim to ignore modality-specific attributes, and learn the common hidden variable that affects both modalities.

Several works develop neural networks which are invariant to specific input transformations, such as translation and rotation (see, for example [4, 5]). However, these networks are designed to be invariant to specific, well modeled transformations, rather than to learn an unknown transformation.

Our architecture is related to the the architecture proposed in [6] for a related problem of non linear Canonical Correlation Analysis (CCA) using a Deep Neural Network. In this manuscript we follow a different approach in the use of the dataset, in the comparison of the two networks, and in the design of the loss function; our experiments suggest that the approach presented here is more suited to the common variable discovery problem.

The common variable problem has been studied in [1] from the manifold learning and kernel perspective. While the approach presented in [1] and the approach presented in this manuscript share some conceptual relations, they have fundamentally different properties and they use fundamentally different approaches to the problem, a comparison of the two approaches will be presented at a later date.

2 Algorithms

2.1 Rationale

The goal of the algorithm introduced in this manuscript is to construct a function $f_1 : \mathcal{S}^{(1)} \to \mathbb{R}^d$ that maps samples $s_i^{(1)} = g_1(x_i, y_i)$ to a representation that reflects X up to some bi-Lipschitz transformation; in particular, we would like $f_1(s_i^{(1)})$ to depend on x_i and be invariant to the value of y_i and we would like f_1 to satisfy (3) and (4).

We assume that the data are provided as pairs of measurements $D_{pos} = \{(s_i^{(1)}, s_i^{(2)})\}_{i=1}^{n_{pos}}$, where the measurements $s_i^{(1)} = g_1(x_i, y_i)$ and $s_i^{(2)} = g_2(x_i, z_i)$ are functions of x_i, y_i and z_i, which are hidden realizations of the random variables X, Y and Z. The crucial information is provided in the dataset through the fact that both $s_i^{(1)}$ and $s_i^{(2)}$ in the i-th pair are functions of the same value of x_i. The idea is to use this information to learn maps f_1 and f_2 such that for all i,

$$f_1(s_i^{(1)}) = f_2(s_i^{(2)}). \tag{5}$$

Assuming the appropriate independence of Y and Z given X, the requirement in equation (5) makes f_1 and f_2 functions of x_i, which gives the property that for every $x \in \mathcal{X}$, $y, y' \in \mathcal{Y}$ and $z, z' \in \mathcal{Z}$,

$$f_1(g_1(x, y)) = f_1(g_1(x, y')), \tag{6}$$

and similarly

$$f_2(g_2(x, z)) = f_2(g_2(x, z')). \tag{7}$$

The trivial solution that satisfies (6) and (7) is constant functions f_1 and f_2; to avoid the trivial solution, we add the requirement that for all $x_i \neq x_j$,

$$f_1(s_i^{(1)}) \neq f_2(s_j^{(2)}), \tag{8}$$

so that f_1 and f_2 cannot simply "ignore" the value of x.

We implement the function f_1 by a network which we denote by $\mathcal{N}_1$, and the function f_2 by a network which we denote by $\mathcal{N}_2$; the algorithm is designed to train $\mathcal{N}_1$ and $\mathcal{N}_2$ to satisfy a relaxed version of the requirements (5) (thus, implicitly, (6) and (7)) and (8). We construct a joint network $\mathcal{N}$, described in Figure 3, which compares the outputs of $\mathcal{N}_1$ and $\mathcal{N}_2$.

The idea is that when we use sample $s_i^{(1)}$ from the pair $(s_i^{(1)}, s_i^{(2)}) \in D_{pos}$ as input to $\mathcal{N}_1$, the output of $\mathcal{N}_1$ should be identical to the output of $\mathcal{N}_2$ when $s_i^{(2)}$ is used as input. We train the joint network $\mathcal{N}$ to minimize the difference between the two outputs, thereby approximating (5) and thus, implicitly, (6) and (7).

In order to approximate the condition (8) too, we use a second dataset $D_{neg} = \{(\tilde{s}_i^{(1)}, \tilde{s}_i^{(2)})\}_{i=1}^{n_{neg}}$ composed of "false" pairs of samples; this dataset is constructed from D_{pos} or obtained in some other way. The idea is that when we use sample $\tilde{s}_i^{(1)} \in \mathcal{S}^{(1)}$ from the pair $(\tilde{s}_i^{(1)}, \tilde{s}_i^{(2)}) \in D_{neg}$ as input to $\mathcal{N}_1$, the output of $\mathcal{N}_1$ should be different from the output of $\mathcal{N}_2$ when $\tilde{s}_i^{(2)} \in \mathcal{S}^{(2)}$ is used as input. We train the joint network $\mathcal{N}$ to maximize the difference between the two outputs, thereby approximating (8).

At the end of the process, the map f_1 implemented by $\mathcal{N}_1$ is our approximate representation; as a useful "side effect," we also obtain the map f_2 which approximates a similar function for the samples obtained from Sensor 2.

In the context of the toy problem described in Section 1.1, we have $\mathcal{N}_1$ a network which accepts snapshots from Camera 1 and $\mathcal{N}_2$ a network which accepts snapshots from Camera 2; the idea is to have these two networks give the same output when they are presented with snapshots that were taken simultaneously and to give a different output when the images were taken at different times. The best that the two networks can be expected to do is to "conspire" to produce an output that depends only on the rotation angle of the bulldog, which they can both observe; if the bulldog is in the same position, the snapshots could have been taken at the same time and the two networks would give the same output, and if the bulldog is in different positions, then the snapshots were clearly taken at different times and the two networks would give different outputs. Since the bunny is only viewed by Camera 2, $\mathcal{N}_1$ cannot take its position into consideration, and since Yoda is only viewed by Camera 1, $\mathcal{N}_1$ should prefer to ignore it when it attempts to "guess" what the output of $\mathcal{N}_2$ would be.

2.2 A summary of the algorithm

The purpose of this section is to provide a brief summary of the algorithm, with references to the more detailed discussion in the following sections.

```
Data: $\{(s_i^{(1)}, s_i^{(2)})\}_{i=1}^n$.
Result: $\mathcal{N}_1$ and $\mathcal{N}_2$: implementation of maps $f_1 : \mathbb{R}^{d_1} \to \mathbb{R}^d$ and $f_2 : \mathbb{R}^{d_2} \to \mathbb{R}^d$.
Construct datasets $D_{pos}$ and $D_{neg}$ (see Section 2.3).
Optional: pre-train each layer of $\mathcal{N}_1$ and $\mathcal{N}_2$ using autoencoders (see Section 2.6.1).
Optimize the parameters of the joint network $\mathcal{N}$ (see Section 2.4).
Optional: dimensionality reduction of the learned representation (see Section 2.6.3).
```

Algorithm 1: Common variable learning using an ANN

2.3 Constructing a dataset of positive pairs and a dataset of negative pairs

The algorithm is given a dataset of n pairs corresponding to n realizations of X, Y, Z. We refer to this dataset as the *positive dataset*,

$$D_{pos} = \left\{ \left(s_i^{(1)}, s_i^{(2)} \right) \right\}_{i=1}^{n_{pos}}. \tag{9}$$

We construct a second dataset, referred to as the *negative dataset*, which contains "false pairs":

$$D_{neg} = \left\{ \left(\tilde{s}_i^{(1)}, \tilde{s}_i^{(2)} \right) \right\}_{i=1}^{n_{neg}}, \tag{10}$$

where $\tilde{s}_i^{(1)} = g_1(\tilde{x}_i, \tilde{y}_i)$ and $\tilde{s}_i^{(2)} = g_2(\tilde{x}'_i, \tilde{z}_i)$. Ideally, $\tilde{s}_i^{(1)}$ and $\tilde{s}_i^{(2)}$ in the pair $\left(\tilde{s}_i^{(1)}, \tilde{s}_i^{(2)}\right)$ should be different realizations with different values of X, so that $\tilde{x}_i \neq \tilde{x}'_i$; in practice, it suffices that $\tilde{x}_i \neq \tilde{x}_i$ with sufficiently high probability. When D_{neg} is not explicitly available, an approximation is constructed from D_{pos} by randomly mixing pairs as follows:

$$D_{neg} = \left\{ \left(s_{r_1(i)}^{(1)}, s_{r_2(i)}^{(2)} \right) \right\}_{i=1}^{n_{neg}}, \tag{11}$$

with $s_j^{(1)}$ and $s_j^{(2)}$ elements in pairs in the dataset D_{pos}, and $r_1, r_2 : \{1, ..., n\} \to \{1, ..., n\}$ some random functions, such as random permutations.

2.4 The Neural Network Architecture

The architecture of our network $\mathcal{N}$ is presented in Figure 3. The network is composed of two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ and a single output unit which is connected to the output layer of both networks. In our experiments, both $\mathcal{N}_1$ and $\mathcal{N}_2$ have an input layer and one to three additional layers; however, various network configurations are applicable and the two networks need not have the same structure; the two networks are only required to have the same number of output units.

The output node Q of $\mathcal{N}$ compares the output of $\mathcal{N}_1$ and $\mathcal{N}_2$,

$$Q(s^{(1)}, s^{(2)}) = \sigma\left(\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2\right) \tag{12}$$

with $s^{(1)}$ and $s^{(2)}$ the inputs of $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively, $f_1(s^{(1)})$ and $f_2(s^{(2)})$ the outputs of $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively, and σ the sigmoid function.

The network is optimized to minimize the loss function $L(\theta)$, defined by the formula

$$\begin{aligned} L(\theta) =& \frac{\alpha}{n_{pos}} \sum_{(s^{(1)},s^{(2)})\in D_{pos}} \left(\frac{1}{2} - \sigma\left(\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2\right)\right) + \\ & \frac{\beta}{n_{neg}} \sum_{(s^{(1)},s^{(2)})\in D_{neg}} \left(1 - \sigma(\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2)\right) + \lambda\|\theta\|_2^2, \end{aligned} \tag{13}$$

where θ is a vector containing the weight parameters (but not bias parameters) of $\mathcal{N}_1$ and $\mathcal{N}_2$. For the positive pairs in D_{pos}, we would like $\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2$ to be close to zero, thus $\sigma\left(\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2\right)$ close to $\sigma(0) = \frac{1}{2}$; for the negative pairs in D_{neg}, we would like to maximize $\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2$, thus have $\sigma\left(\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2\right)$ close to $\sigma(\infty) = 1$.

2.5 Generalization to k sensors

Our design can be generalized to k sensors, for example, by constructing k networks $\mathcal{N}_1, ..., \mathcal{N}_k$ and connecting them to the output unit. The training data consists of paired observations where each pair corresponds to two of the k sensors, i.e., $(s_i^{(j)}, s_i^{(l)})$, where $j, l \in \{1, .., k\}$. During training, for each such example the output unit computes the Euclidean distance between the outputs of the j-th and l-th networks and the error is back-propagated only through these networks.

2.6 Implementation

The purpose of this section is to describe some technical aspects of the algorithm proposed in Section 2.

2.6.1 Weight initialization and pre-training

The weights are initialized using sampling from normal distribution with zero mean and small variance. In experiments where we have more than a single hidden layer in each network, we pre-train every hidden layer in $\mathcal{N}_1$ and $\mathcal{N}_2$ as a Denoising Autoencoder (DAE)[7] with sparsity loss (see, inter alia, [8]); we use the loss function defined by the formula

$$L(\theta) = \frac{1}{2n}\sum_{i=1}^{n} \|s_i - \hat{\hat{s}}_i\|_2^2, +\lambda\left(\|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2\right) + \beta\sum_{j=1}^{d_H} KL\left(\rho\|\hat{\rho}_j\right), \tag{14}$$

where n is the number of training points, s_i is an input point, $\tilde{s}$ is s by corrupted by setting a random subset of entries to zero, $\hat{\tilde{s}}$ is the reconstruction of the s by the autoencoder, $W^{(1)}$ and $W^{(2)}$ are the weight matrices of the encoder and the decoder respectively, $\|\cdot\|_F$ is Frobenius norm, d_H is the number of hidden units, ρ is some small positive number (e.g. 0.05), $\hat{\rho_j}$ is the average activation of the j-th unit, $KL\left(\rho\|\hat{\rho_j}\right)$ is the Kullback-Leibler divergence between the probability mass functions of Bernoulli random variables with parameters ρ and $\hat{\rho_j}$, and β, λ are some nonnegative constants. The first term in (14) is the squared reconstruction error, the middle term is an l_2 penalty over the encoder and decoder weights, and the last term encourages the average activation of every hidden unit to be near ρ, so that the hidden representation of the data is sparse.

2.6.2 Optimization

The optimization of the network $\mathcal{N}$ is performed using standard Stochastic Gradient Descent (SGD) with momentum (see, for example, [9]) and dropout (see, for example, [10]), or using L-BFGS (see, for example, [11]); in both optimization algorithms, we compute gradients using standard backpropagation (see, for example, [12]). In the optional pre-training using autoencoders, we use L-BFGS.

Remark 2.1. In all our experiments, the learning rate of the SGD needed to be fairly large, typically in the range $1-10$, compared to $0.01-0.001$ used in many ANN applications. This might imply flatness of our loss function.

2.6.3 Dimensionality reduction and visualization

Suppose that $\{s_i^{(1)}\}_{i=1}^n$ is a set of samples from Sensor 1, the learned network $\mathcal{N}_1 : \mathcal{S}^{(1)} \to \mathbb{R}^d$ maps each sample $s_i^{(1)}$ to the representation $q_i^{(1)}$ as follows:

$$q_i^{(1)} = f_1(s_i^{(1)}). \tag{15}$$

When the representation is low dimensional (d between 1 and 3), it is often easy to visualize the representation of the samples using a scatter plot of the vectors in $\{q_i^{(1)}\}_{i=1}^n$. However, in our experiments we observed that the algorithm performs poorly when we set $d \leq 3$ compared to a larger d; typically $50 \leq d \leq 200$.

We use two methods of dimensionality reduction to analyze the representation $\{q_i^{(1)}\}_{i=1}^n$ obtained for datasets in the experiments in this manuscript. The first method is the standard Principal Components Analysis (PCA), where we use the first three principal components of $\{q_i^{(1)}\}_{i=1}^n$ to produce a three-dimensional scatter plot of the representation; the second method is diffusion maps [13] (for brief description of diffusion maps see Appendix A.1).

The diffusion maps algorithm typically produces simpler reduced representations as can be seen, for example, in Figure 4; therefore, we prefer to use it in most of our experiments.

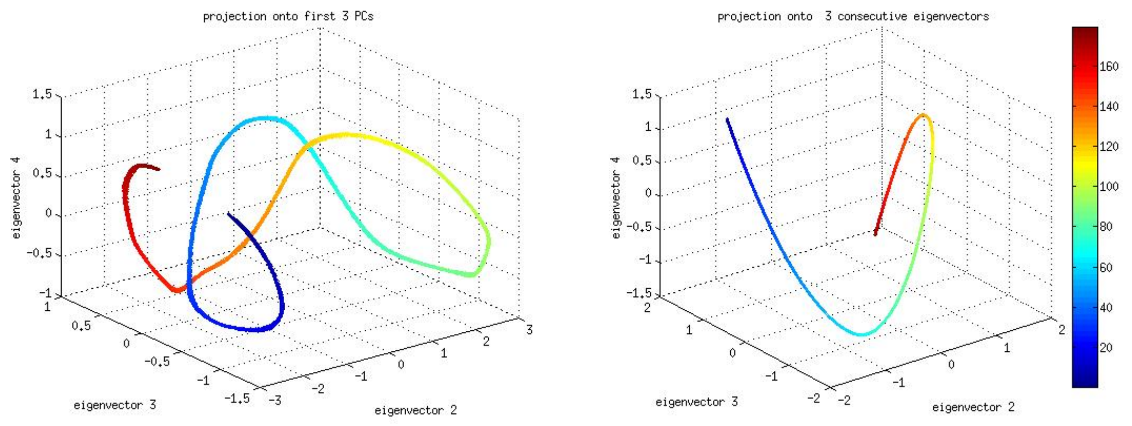


Figure 4: A three dimensional reduced representation obtained in one of our experiments. Left: PCA. Right: diffusion map. The color of each point corresponds to the true value of the common variable.

2.7 Using a test set to evaluate the performance of the ANN

The output of $Q(s^{(1)}, s^{(2)})$ (defined in (12)) of the joint network $\mathcal{N}$ can be interpreted as a classification prediction. We set 0.75 as a classification threshold for estimating whether $(s^{(1)}, s^{(2)})$ is a "real" or "fake" pair, so that if $Q(s^{(1)}, s^{(2)}) < 0.75$ the network estimates that the pair $(s^{(1)}, s^{(2)})$ is a positive pair like those in D_{pos} and if $Q(s^{(1)}, s^{(2)}) \geq 0.75$ the network classifies the pair as a negative pair, like those in D_{neg}.

Although the network is not designed as a classifier, we use its classification error on a test set as a rough estimate of the quality of the learned representation. Since the choice of the threshold is arbitrary, we do not expect the test error to be zero.

3 Properties of the maps f_1 and f_2

3.1 The quotient set and equivalence learning

The purpose of this section is to describe the problem of finding a common variable in terms of finding equivalence classes; a brief reminder of the related definitions appears in Appendix A.2. In the setting described in Section 1.2, we introduce the equivalence relation $\sim$ and we say that two samples are equivalent if and only if they share the same value of X. In other words, the realization (x_i, y_i, z_i) of the hidden variables (X, Y, Z) is equivalent to the realization of the realization (x_j, y_j, z_j) if and only if $x_i = x_j$, regardless of the values of Y and Z:

$$(x_i, y_i, z_i) \sim (x_j, y_j, z_j) \text{ iff } x_i = x_j. \tag{16}$$

In other words, the quotient set $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z} / \sim$ is isomorphic to $\mathcal{X}$.

Since g_1 is invertible, this equivalence relation extends to $\mathcal{S}^{(1)}$, the space of measurements in Sensor 1; with a minor abuse of notation:

$$s_i^{(1)} \sim s_j^{(1)} \text{ iff } x_i = x_j, \tag{17}$$

and similarly in Sensor 2:

$$s_i^{(2)} \sim s_j^{(2)} \text{ iff } x_i = x_j. \tag{18}$$

Therefore, we have the quotient set $\mathcal{S}^{(1)}/\sim$, where the equivalence class of $s^{(1)} = g_1(x, y)$ is

$$[s^{(1)}] = \left\{ g_1(x, y') | y' \in \mathcal{Y} \right\}. \tag{19}$$

We observe that a function f_1 that satisfies (3) yields the same value for any member of an equivalence class $[s^{(1)}]$. In other words, with a minor abuse of notation, there is a natural way to define $f_1 : \mathcal{S}^{(1)}/\sim \to \mathbb{R}^d$ on the quotient set $\mathcal{S}^{(1)}/\sim$ rather than on $\mathcal{S}^{(1)}$. Moreover, a function f_1 that satisfies (4) also yields a different value for members of different equivalence classes $[s_i^{(1)}] \neq [s_j^{(1)}] \in \mathcal{S}^{(1)}/\sim$. In other words, such f_1 is an injective function, defined on the quotient set $\mathcal{S}^{(1)}/\sim$.

3.2 Alternative formulation: invariant representation learning

In this section we present the closely related problem of learning representations that are invariable to certain transformations; a brief reminder of the related definitions appears in Appendix A.3.

Suppose that G is a group that acts on a set $\mathcal{S}$. We say that $s \in \mathcal{S}$ is equivalent to $s' \in \mathcal{S}$ up to G if there is $g \in G$ such that

$$g.s = s'. \tag{20}$$

we denote the equivalence relation by $s \sim s'$. A representation f_1 of $\mathcal{S}$ that is invariant to G satisfies for all $s \in \mathcal{S}$ and $g \in G$

$$f_1(s) = f_1(g.s) \tag{21}$$

and for all $s \nsim s'$ and all $g \in G$

$$f_1(s) \neq f_1(g.s'). \tag{22}$$

In the invariant representation learning problem, we have examples of pairs $(g.s, g'.s)$ with different randomly selected group actions $g, g' \in G$ operating on a randomly selected element $s \in \mathcal{S}$ and in some cases we may have examples of "negative pairs" $(g.s, g'.s')$ with $s \neq s' \in \mathcal{S}$ (or, we can generate such "negative pairs" using (11)). We would like to use these datasets to find a function f_1 that satisfies (21) and (22).

The invariant representation learning problem is very similar to the equivalence learning problem as formulated in Section 3.1; in both cases we aim to construct functions that distinguish between equivalence classes. The description of the algorithm applies to either formulation with small changes; for simplicity, the discussion in this manuscript follows the formulation in Section 1.2.

Remark 3.1. Using the network design to learn invariant representation can be also done while enforcing $f_1 = f_2$, for example by using same architecture of the two networks $\mathcal{N}_1$ and $\mathcal{N}_2$, and keeping their weights tied during the optimization.

3.3 The topology of the mapped data and comparing measurements from the same sensor

The goals stated in (3) and (4) require the function f_1 to give the exact same value for samples with the exact same X and strictly different values for samples with different values of X. When such a function is obtained, it can be used to compare the samples and determine if they have the same value of X, i.e. if they are equivalent in the sense defined in Section 3.1.

In practice, because of the continuity of the functions g_1 and g_2 and the continuity of the computation operations in the networks that we use here, samples that are "close" in X would have similar representations, so that the representation of X is smooth. Informally,

$$x_i \approx x_j \Leftrightarrow f_1(g_1(x_i, y_i)) \approx f_1(g_1(x_j, y_j)), \tag{23}$$

therefore, the function f_1 can be used to estimate if two samples $s_i^{(1)}$ and $s_j^{(1)}$ in Sensor 1 have "close" values x_i and x_j.

3.4 Comparing measurements from different sensors

The algorithm treats the measurements in Sensor 1 and the measurements in Sensor 2 symmetrically, in the sense that it aims to construct maps $f_1 : \mathcal{S}^{(1)} \to \mathbb{R}^d$ and $f_2 : \mathcal{S}^{(2)} \to \mathbb{R}^d$ that map into the same codomain $\mathbb{R}^d$ and have similar properties. Moreover, the algorithm aims to find such f_1 and f_2 that agree in the sense defined in equations (5) and (8).

Following the same argument as in Section 3.3, the two functions f_1 and f_2 can be used to compare a sample $s_i^{(1)} = g_1(x_i, y_i)$ from Sensor 1 to a sample $s_j^{(2)} = g_2(x_j, z_j)$ from Sensor 2 to estimate whether the two samples are obtained from "close" values of X; informally,

$$x_i \approx x_j \Leftrightarrow f_1(g_1(x_i, y_i)) \approx f_2(g_2(x_j, z_j)). \tag{24}$$

The two sensors might measure different modalities, such as audio signals in one and images in the other, so that the framework proposed here allows to compare two different modalities in terms of the common variable.

4 Experimental results and discussion

In this section we present experimental results of common variable learning and invariant representation learning, and discuss properties of the algorithm that have been observed in the experiments.

In each of the experiments below, we used a positive dataset D_{pos} and a negative dataset D_{neg} of equal size to train the network, each sample in each dataset is a pair $(s_i^{(1)}, s_i^{(2)})$. For testing, we generated a different positive dataset and a different negative dataset, independently of the ones used for training. In each experiment, the testing datasets were of equal size as well.

In each experiment, we construct two networks, $\mathcal{N}_1$ which takes samples from Sensor 1 as input, and $\mathcal{N}_2$ which takes samples from Sensor 2 as input, and combine them in the joint network $\mathcal{N}$. The networks $\mathcal{N}_1$ and $\mathcal{N}_2$ have either three layers (two hidden layers and an output layer) or one layer (output layer) composed of sigmoid units; the output layers of $\mathcal{N}_1$ and $\mathcal{N}_2$ serve as a hidden layer before the single output node of the joint network $\mathcal{N}$. The three-layer networks are pre-trained with a DAE (see section 2.6.1).

As the network $\mathcal{N}$ is trained, its components $\mathcal{N}_1$ and $\mathcal{N}_2$ are also trained, and implement the maps $f_1 : \mathcal{S}^{(1)} \to \mathbb{R}^d$ and $f_2 : \mathcal{S}^{(2)} \to \mathbb{R}^d$. Once the optimization is completed, we break every test example $(s_i^{(1)}, s_i^{(2)})$ into its elements $s_i^{(1)}$ and $s_i^{(2)}$ and map each of these elements to $\mathbb{R}^d$ using the corresponding map. Having obtained this representation for the observations from both sensors, we compute the diffusion embedding of the test data and embed each element into $\mathbb{R}^n$ based on the first three diffusion coordinates. We present figures of these diffusion embeddings, where each point represents an element $s_i^{(1)}$ or $s_i^{(2)}$; the color assigned to each point represents the true value of the underlying X, which we are attempting to recover. In addition, we measure the accuracy of the network in the sense defined in Section 3.4.

4.1 Common variable learning: the Person dataset (rotated images)

In this section we present an example for automatic construction of a representation for a common variable from two sets of images. Each sample $(s_i^{(1)}, s_i^{(2)})$ in the positive dataset D_{pos} is a function of three parameters $x_i, y_i, z_i \in [0, 180]$, chosen uniformly at random. The samples are generated from these parameters as follows: we denote by I_θ the image I, rotated by θ; $s_i^{(1)}$ is a concatenation of two rotated images, the image I rotated by x_i and the image I rotated by y_i:

$$s_i^{(1)} = (I_{x_i}, I_{y_i}). \tag{25}$$

Similarly, $s_i^{(2)}$ is a concatenation of two rotated images, the image I rotated by x_i and the image I rotated by z_i:

$$s_i^{(2)} = (I_{x_i}, I_{z_i}), \tag{26}$$

as illustrated in Figure 5. The common variable is hence x_i, which is manifested in the left side of $s_i^{(1)}$ and $s_i^{(2)}$, and the superfluous variables are y_i and z_i, which are manifested in the right side of $s_i^{(1)}$ and $s_i^{(2)}$. Figure 5 presents a positive example from the dataset. Figure 6 presents a negative example $(\tilde{s}_i^{(1)}, \tilde{s}_i^{(2)})$ of the negative dataset D_{neg}, where

$$\tilde{s}_i^{(1)} = (I_{\tilde{x}_i}, I_{\tilde{y}_i}) \tag{27}$$

and

$$\tilde{s}_i^{(2)} = (I_{\tilde{x}'_i}, I_{\tilde{z}_i}). \tag{28}$$

Both $s_i^{(1)}$ and $s_i^{(2)}$ are 968-dimensional.

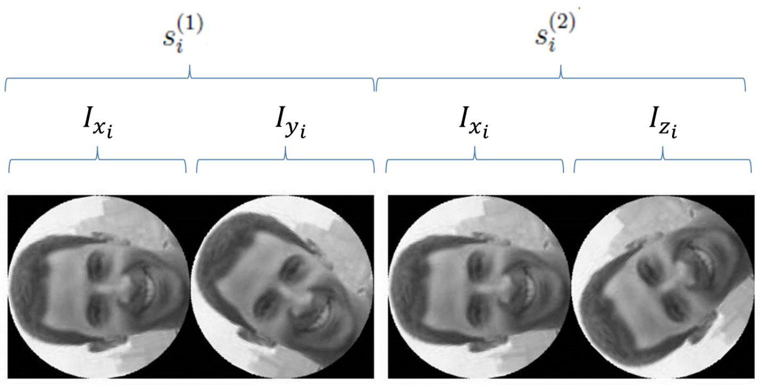


Figure 5: Each sample in the positive dataset is composed of two images, $s_i^{(1)}$ and $s_i^{(2)}$. $s_i^{(1)}$ is a concatenation of I_{x_i} and I_{y_i}. $s_i^{(2)}$ is a concatenation of I_{x_i} and I_{z_i}.

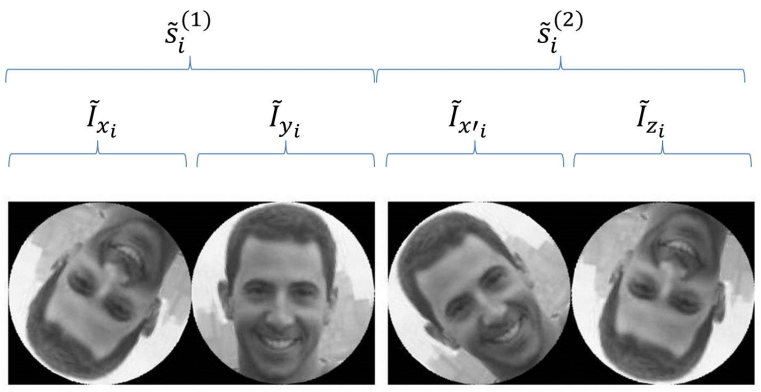


Figure 6: Each sample in the negative dataset is composed of two images, $\tilde{s}_i^{(1)}$ and $\tilde{s}_i^{(2)}$. $\tilde{s}_i^{(1)}$ is a concatenation of $I_{\tilde{x}_i}$ and $I_{\tilde{y}_i}$. $\tilde{s}_i^{(2)}$ is a concatenation of $I_{\tilde{x}'_i}$ and $I_{\tilde{z}_i}$.

Each of the training datasets D_{pos} and D_{neg} contained 1,000 pairs of samples. Both $\mathcal{N}_1$ and $\mathcal{N}_2$ had a single layer of size 100, the network was fine-tuned using SGD. The network achieved 96.2% accuracy on a test set (as defined in Section 3.4). The diffusion embedding of the test data is presented in Figure 7; the smooth transition of colors in the one dimensional curve indicates that the curve indeed represents the common variable.

A visual representation of the weights of the units in the first hidden layer of $\mathcal{N}_1$ and $\mathcal{N}_2$ is presented in Figure 8; the figure demonstrates that both networks capture features from the left side of the images $s^{(1)}$ and $s^{(2)}$, which is influenced by the variable X, and virtually ignore the right side of $s^{(1)}$, which is influenced by Y and right side of $s^{(2)}$, which is influenced by Z.

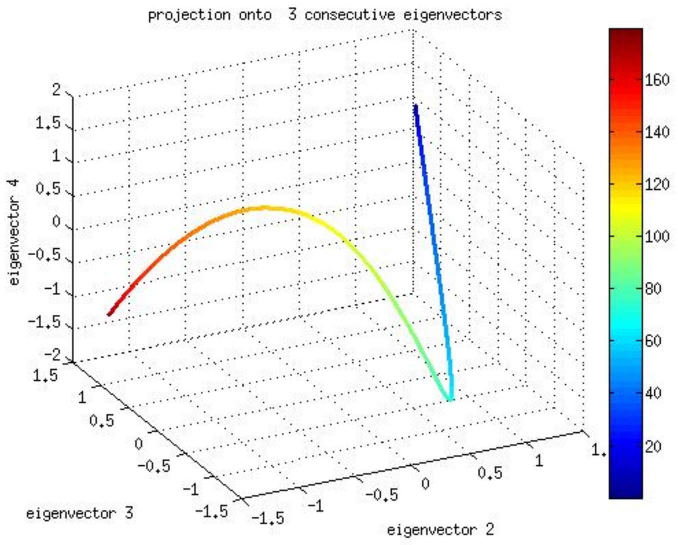
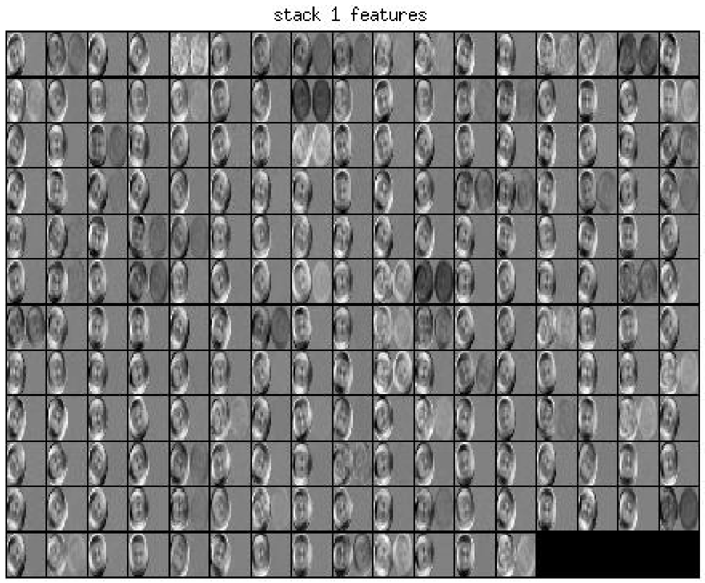


Figure 7: Embedding of the test set of the Person dataset: projection onto the first three non trivial eigenvectors. Color gradient corresponds to the value of common hidden variable X (rotation angle of the left image, in degrees).

4.1.1 Sensitivity analysis

In this section we describe variants of the experiment in Section 4.1, in which we test the performance in three different regimes:

- Case 1: small dataset: 200 examples, 100 hidden units in each output layer. The accuracy on the test set is worse: 74.9%.
- Case 2: limited network capacity: 2,000 training examples, 10 hidden units. The accuracy on the test set is 86.3%.
- Case 3: more hidden layers: 2,000 training examples, three layers, each with 100 units. The accuracy on the test set is 91.96%.



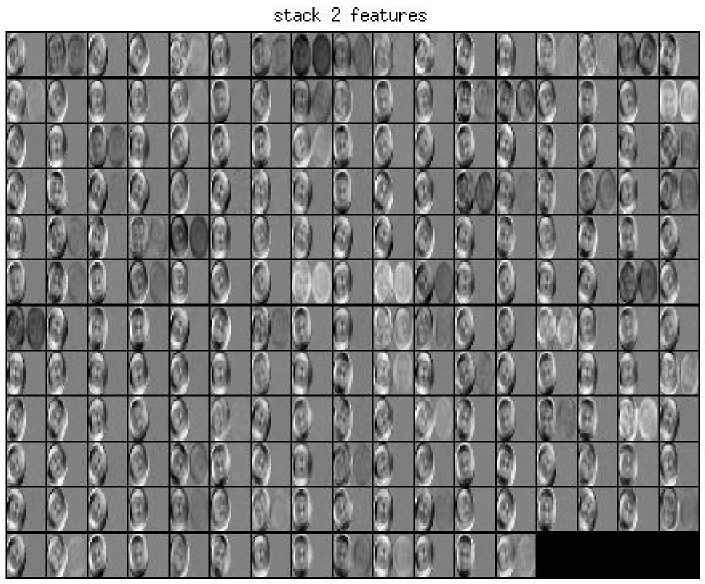


Figure 8: the weights of the first hidden layer on $\mathcal{N}_1$ (top) and $\mathcal{N}_2$ (bottom), learned from the Person dataset.

The diffusion embeddings of the representations obtained in the three cases are presented in Figure 9.

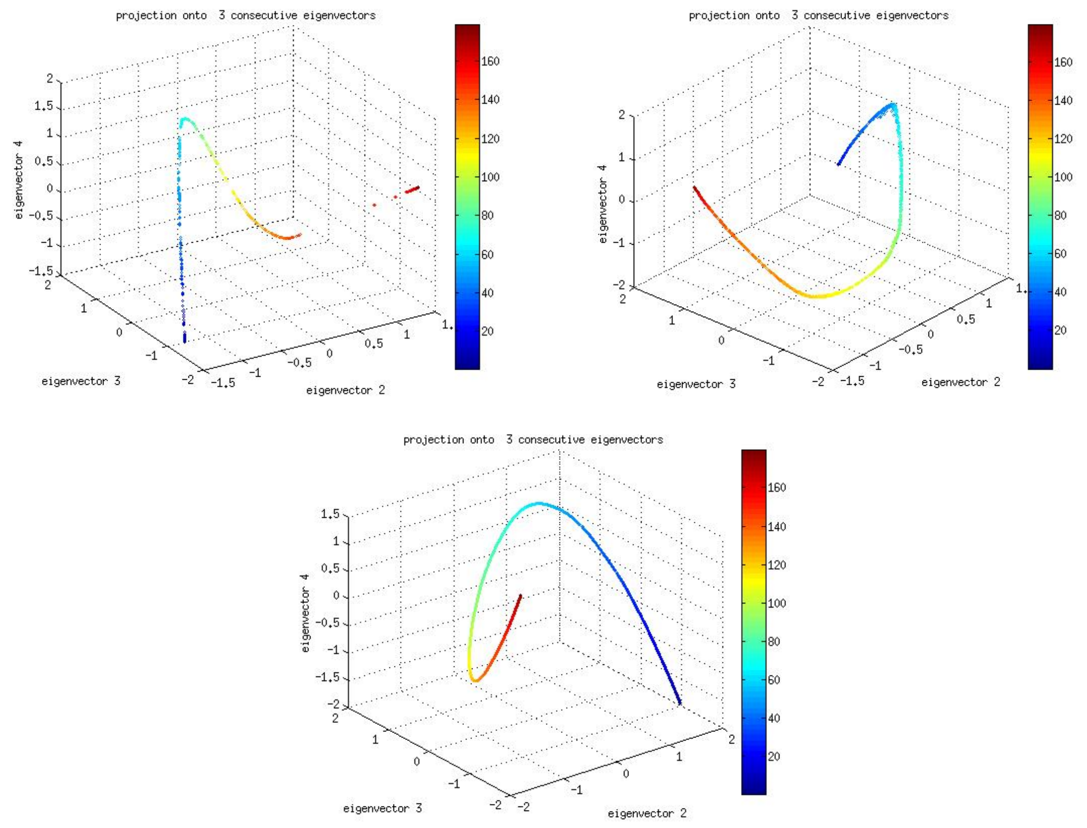


Figure 9: Sensitivity analysis experiments on the Person dataset. Top left: small dataset. Top right: limited network capacity. Bottom: more hidden layers.

4.2 Common variable learning: the Toy dataset (spinning figures)

In this example, we revisit the toy experiment described in Section 1.1 and in Figure 1. Here, $S^{(1)}$ is a snapshot taken by Camera 1 and $S^{(2)}$ is a snapshot taken by Camera 2. The dataset D_{pos} is composed of pairs of snapshots $(s_i^{(1)}, s_i^{(2)})$ where $s_i^{(1)}$ and $s_i^{(2)}$ are taken simultaneously by Camera 1 and Camera 2, respectively. The dataset D_{neg} is constructed by pairing snapshots that where taken at different times. The samples $s_i^{(1)}$ and $s_i^{(2)}$ are 60×80 color images (hence the dimensionality is 14,400). Examples for elements in D_{pos} and D_{neg} are presented in Figure 10.

In this experiment, the training sets D_{pos} and D_{neg} consisted of $10,000$ examples each. Both $\mathcal{N}_1$ and $\mathcal{N}_2$ had three layers, the two hidden layers in each network had 150 units, and the

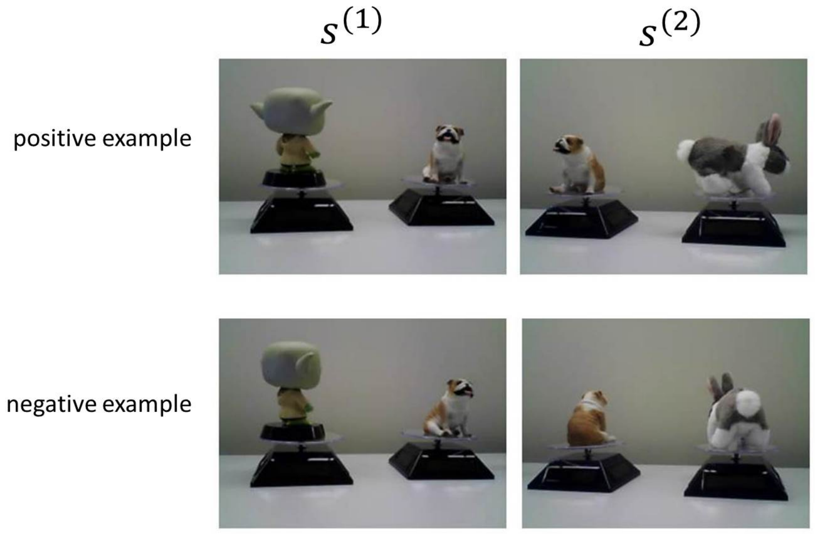


Figure 10: Two sample examples from the Toy dataset. Top row: a positive example from D_{pos}: snapshots taken simultaneously, containing two different views on the bulldog at the same rotation angle. Bottom row: a negative example from D_{neg}. Left column: the snapshot $s^{(1)}$ taken be Camera 1. Right column: $s^{(2)}$ taken by Camera 2.

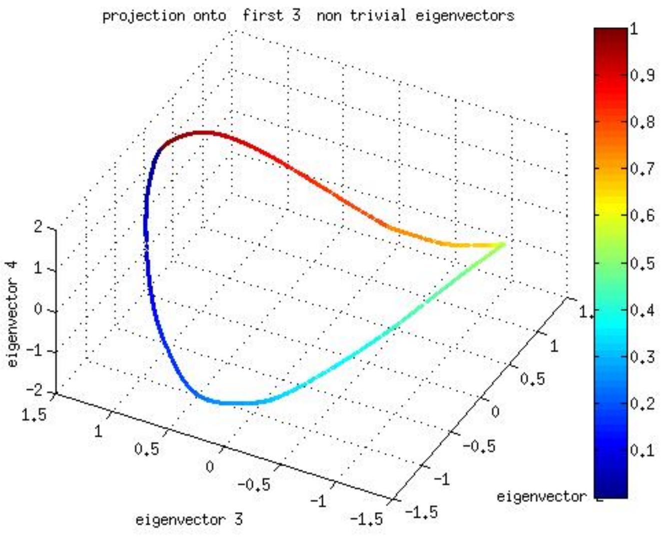


Figure 11: Embedding of the Toy dataset. The color of each point corresponds to the true common hidden variable, i.e. the rotation angle of the bulldog.

output layers had 100 units. The joint network $\mathcal{N}$ was trained using L-BFGS. The classification accuracy on the test set (as defined in Section 3.4) was 95.96%. Figure 11 presents the diffusion map of the computed representation; the closed curve and the smooth transitions in color found in Figure 11 demonstrate that the algorithm recovered a good representation of the common variable X.

4.3 Common variable learning: the Bunny dataset (revolving cameras)

In the previous experiments, the hidden variables were manifested as separable objects in the images; in this experiment, we consider a different type of map from the variables to the images. We constructed a dataset of images of a bunny on a spinning table, and then digitally rotated the images; the digital rotation can be viewed as a revolution of the camera.

We denote by $I_{x,y} = g(x, y)$ an image that was taken by a camera rotated in angle y when the bunny is rotated in direction x. In the positive dataset $s_i^{(1)} = I_{x_i,y_i}$ and $s_i^{(2)} = I_{x_i,z_i}$, so that $s_i^{(1)}$ and $s_i^{(2)}$ are images in which the bunny is rotated in the same direction while the cameras are in arbitrary angles. In this case the functions g_1 and g_2 that generate the sensor measurements are identical $g_1 = g_2 = g$. The dataset D_{neg} is constructed from D_{pos} by pairing $I_{\tilde{x}_i,\tilde{y}_i}$ and $I_{\tilde{x}'_i,\tilde{z}'_i}$, so that the bunny and the camera in $\tilde{s}_i^{(1)}$ are both rotated independently of those in $\tilde{s}_i^{(2)}$. Both $S^{(1)}$ and $S^{(2)}$ are 30×40 gray-scale images. Examples from D_{pos} and D_{neg} are presented in Figure 12.

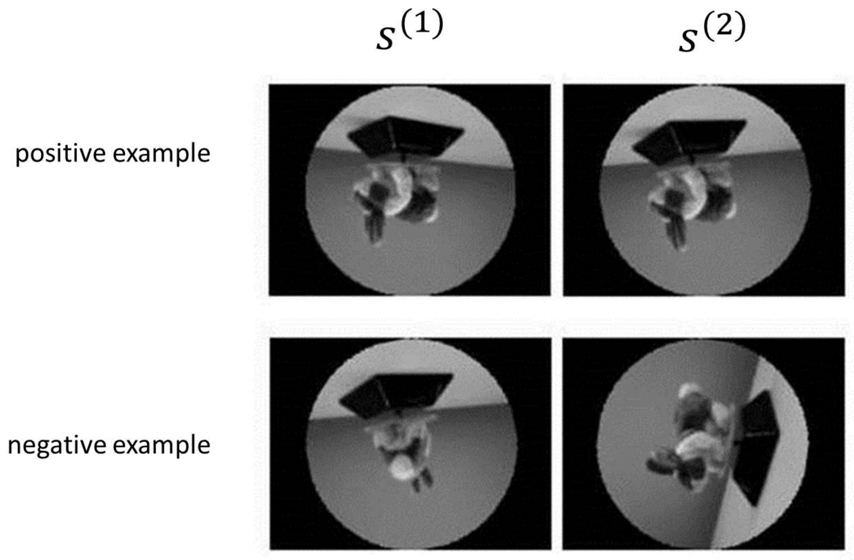


Figure 12: Two sample examples from the Bunny dataset. Top row: a positive example: the bunny is rotated in the same direction in both images, whereas the cameras are rotated in arbitrary angles. Bottom row: a negative example: both the bunny and the camera are rotated in arbitrary angles. Left column: $s^{(1)}$. Right column: $s^{(2)}$.

In this experiment, the training sets D_{pos} and D_{neg} consisted of $10,000$ examples each. The networks $\mathcal{N}_1$ and $\mathcal{N}_2$ had a single layer of 200 units. The training was performed using SGD. The classification accuracy on the test set was 94.7%. The diffusion embedding of the test set is presented in Figure 13; again, the learned representation corresponds to the value of the common variable.

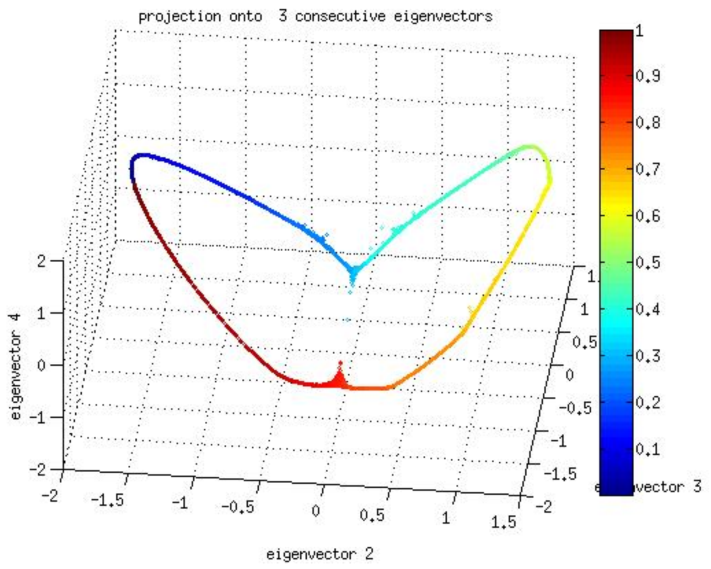


Figure 13: Diffusion embedding of the bunny dataset. The smooth transition of the color along the curve implies that the learned representation is a parametrization of the common variable, i.e., the rotation angle of the bunny.

4.4 Common variable learning: two different modalities

In the previous experiments we used the same type of input in both sensors; in this experiment we used a different data modality in each sensor: images in one sensor and audio signals in the other. Specifically, a measurement from Sensor 1 is a concatenation of two rotated images

$$s_i^{(1)} = (I_{x_i}, I_{y_i}), \tag{29}$$

as in Section 4.1, whereas $s_i^{(2)}$ is a T dimensional vector $v_{x_i,z_i}(t),\ t = 1,..,T$ with entries

$$v_{x_i,z_i}(t) = \sin(2\pi\omega(x_i)t + z_i), \tag{30}$$

where $\omega(\cdot)$ is a deterministic function, so that x_i determines the frequency of the sine, and z_i determines the phase. In other words, the common variable X determines the rotation of the left image in the first sensor and the frequency of the sine in the second sensor; the sensor specific variables are the rotation angle of the right image in $s_i^{(1)}$ and the phase of the sine in $s_i^{(2)}$.

Both $\mathcal{N}_1$ and $\mathcal{N}_2$ had three layers, with 100 units in each. D_{pos} and D_{neg} consisted of $10,000$ examples each. The accuracy on the test set was 93.5%. The diffusion embedding of the test measurements from both sensors is presented in Figure 14. The smooth transition of the color along the manifold implies that that the learned representation corresponds to the common variable.

This experiment demonstrates that the ANN can represent measurements from two different modalities in the same space, with their embedding determined by the value of the variable of interest.

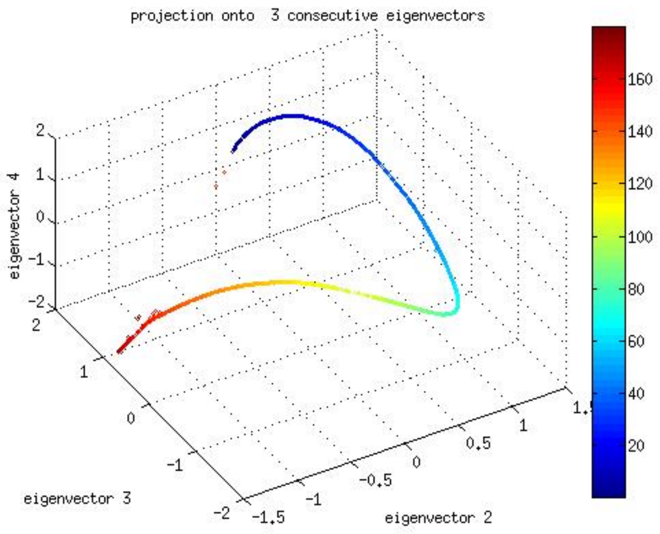


Figure 14: Embedding of the images and audio signals from the test set in the two modalities experiment. Data from two different modalities is mapped to the same space, and is parametrized by the common variable.

4.5 Learning a rotation-invariant representation

The following experiment demonstrates the proposed algorithm's applicability to the invariant representation learning problem, discussed in Section 3.2. The goal here is to learn maps f_1 and f_2 that are rotation-invariant.

The data used for this experiment was generated from images from the Caltech-101 dataset [14]; we pre-processed all images by converting them to 50×50 pixels gray-scale images. Given a set S of images, we define G to be the group of rotations, so that $g.s$ is a rotation of s by g degrees. As discussed in Section 3.2, to generate a sample $(s_i^{(1)}, s_i^{(2)})$ in the positive dataset, we sampled an image s_i from the dataset and two arbitrary rotations $g_{1,i}, g_{2,i} \in G$ and set

$$s_i^{(1)} = g_{1,i}.s_i \tag{31}$$

and

$$s_i^{(2)} = g_{2,i}.s_i, \tag{32}$$

i.e., $s_i^{(1)}$ and $s_i^{(2)}$ are two rotated instances of the same image. The negative dataset was generated in the same way, with two different randomly selected images $\tilde{s}_i$ and $\tilde{s'}_i$ in each negative sample $(\tilde{s}_i^{(1)}, \tilde{s}_i^{(2)})$:

$$\tilde{s}_i^{(1)} = \tilde{g}_{1,i}.\tilde{s}_i \tag{33}$$

and

$$\tilde{s}_i^{(2)} = \tilde{g}_{2,i}.\tilde{s'}_i. \tag{34}$$

A positive example and a negative example are presented in Figure 15. The test set contained a different subset of Caltech-101 then the training set.

The networks $\mathcal{N}_1$ and $\mathcal{N}_2$ had three layers each; the joint network was trained using L-BFGS. The weights of the units in the first hidden layer of $\mathcal{N}_1$ and $\mathcal{N}_2$ are presented in Figure 17. The learned functions achieved a high accuracy score of 99.44%; of the 20,000 test examples, half positive and half negative, we found 112 classification errors, 109 of which were false positives. Some of the errors are presented in Figure 16.

To check whether the hidden representation we obtained is indeed invariant to rotation, we perform the following analysis: we randomly selected an image and rotated it in two arbitrary angles; we denote the resulting images by a and a'. We then selected a different image and rotated it in an arbitrary angle; we denote the resulting image b. If the map f_1 is indeed invariant to rotations, then we expect to have

$$\|f_1(a) - f_1(a')\|_2 \ll \|f_1(a) - f_1(b)\|_2. \tag{35}$$

Histograms of $\|f_1(a) - f_1(a')\|_2$ and $\|f_1(a) - f_1(b)\|_2$ for 10,000 repetitions of the above procedure are presented in Figure 18; as evident from the histograms, $\|f_1(a) - f_1(a')\|_2$ is indeed significantly smaller than $\|f_1(a) - f_1(b)\|_2$, as expected.

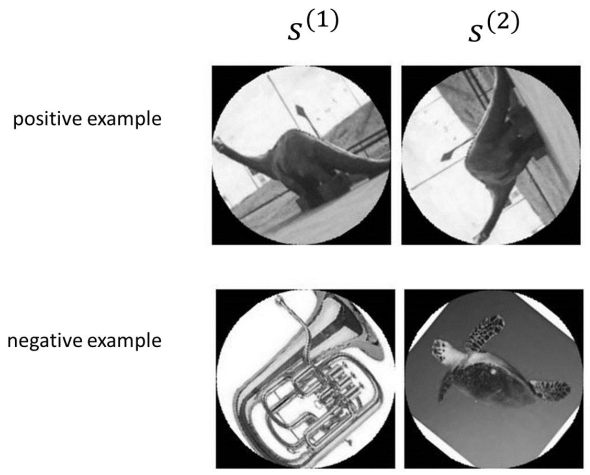


Figure 15: Two sample examples generated from the Caltech-101 dataset for the rotation-invariance experiment. Top row: a positive example ($s^{(1)}$ and $s^{(2)}$ are the same image, up to rotation). Bottom row: a negative example ($s^{(1)}$ and $s^{(2)}$ are the different images, rotated in different angles).

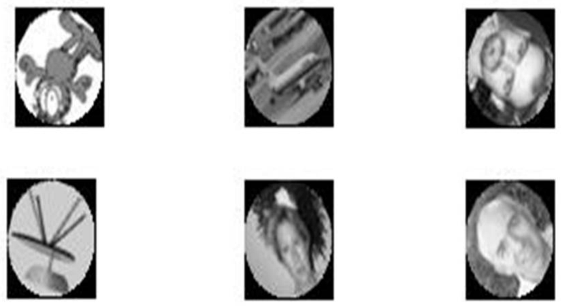
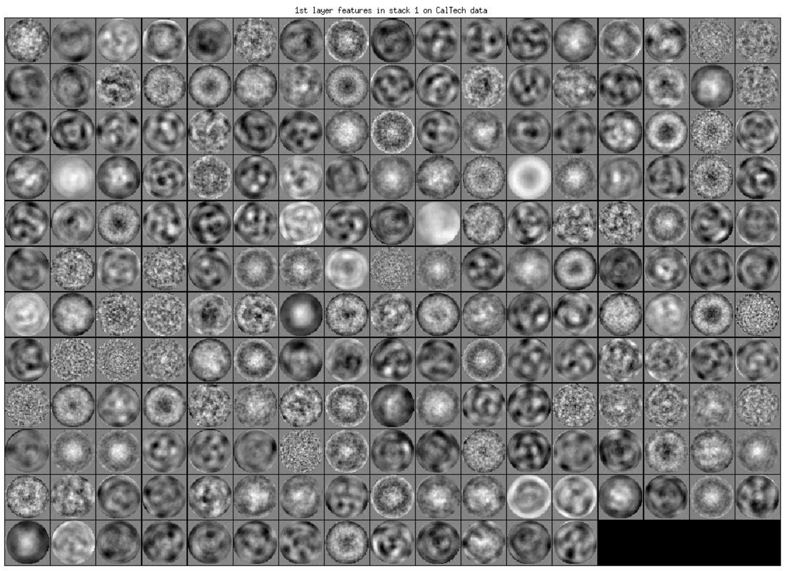


Figure 16: Three errors in the rotation invariance experiment. Each column is a negative example which was classified as positive.



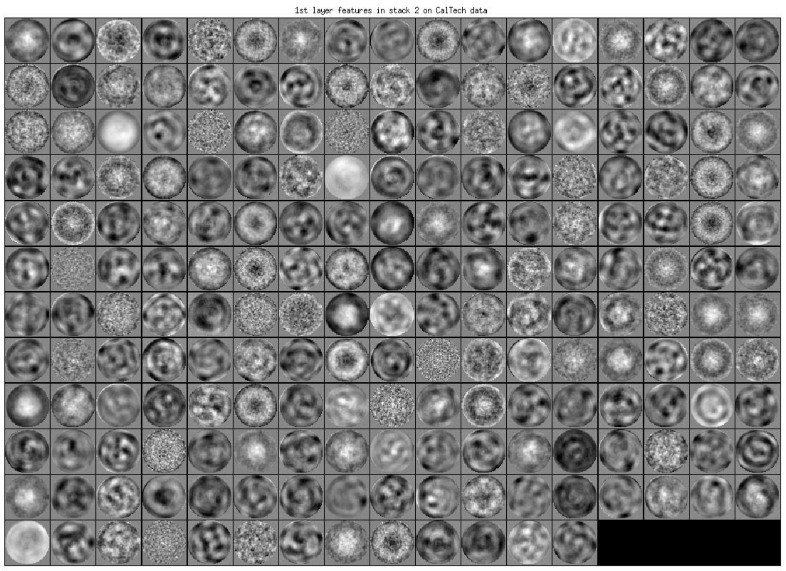


Figure 17: Layer 1 Features in the rotation-invariance experiment.

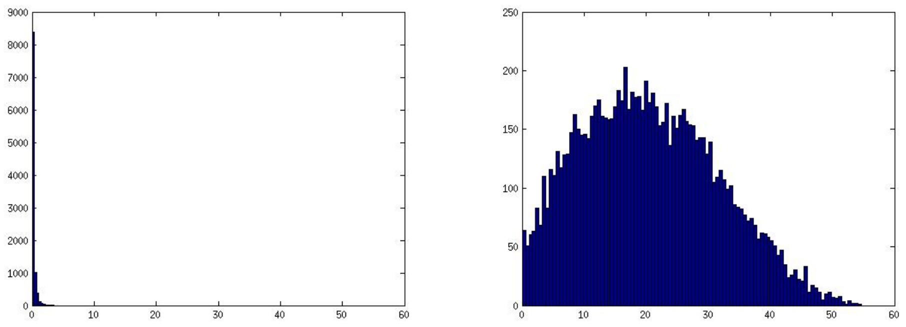


Figure 18: Histograms of $\|f_1(a) - f_1(a')\|_2$ (left) and $\|f_1(a) - f_1(b)\|_2$ (right) in the rotation-invariance experiment.

4.6 Learning a translation-invariant representation

In this section we repeat the rotation-invariance experiment of Section 4.5, but we replace the rotation with a circular horizontal translation; examples of from the datasets are presented in Figure 19.

The accuracy achieved in this experiment was 98.75%. The first layer weights of $\mathcal{N}_1$ and $\mathcal{N}_2$ are presented in Figure 20. Histograms of $\|f_1(a) - f_1(a')\|_2$ and $\|f_1(a) - f_1(b)\|_2$ (defined in Section 4.5) are presented in Figure 21. As in the rotation-invariance experiment, $\|f_1(a) - f_1(a')\|_2$ is significantly smaller than $\|f_1(a) - f_1(b)\|_2$ as desired.

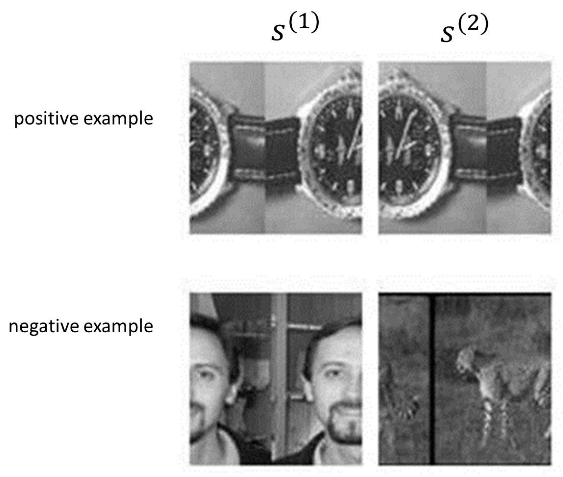
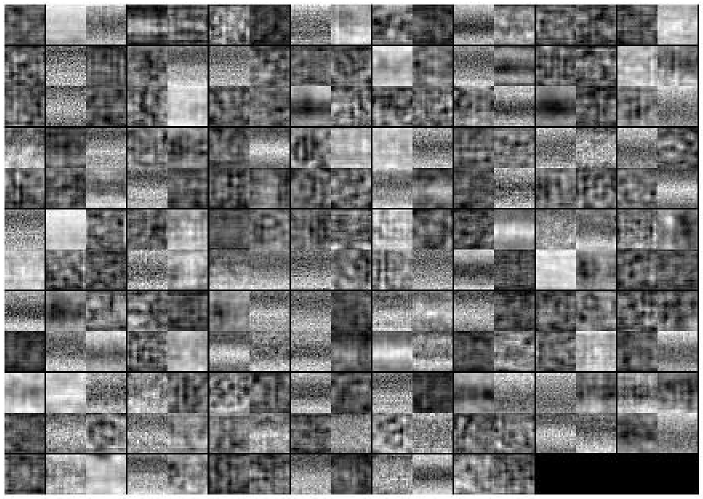


Figure 19: positive (top) and negative (bottom) examples generated from the Caltech-101 dataset in the translation-invariance representation.



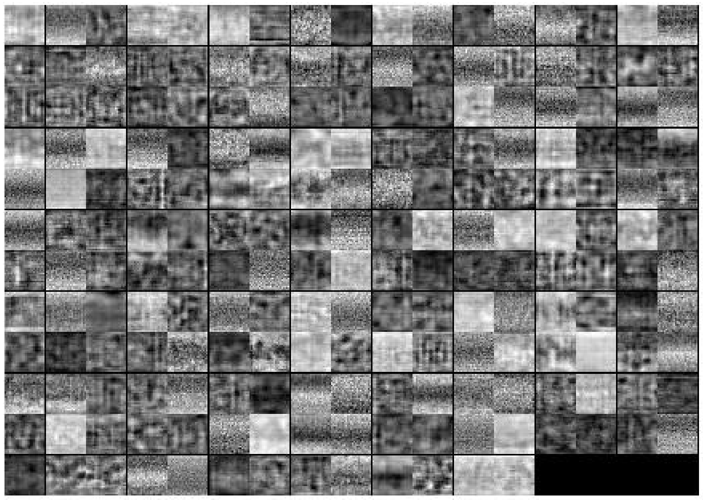


Figure 20: Layer 1 weights in the translation-invariance experiment.

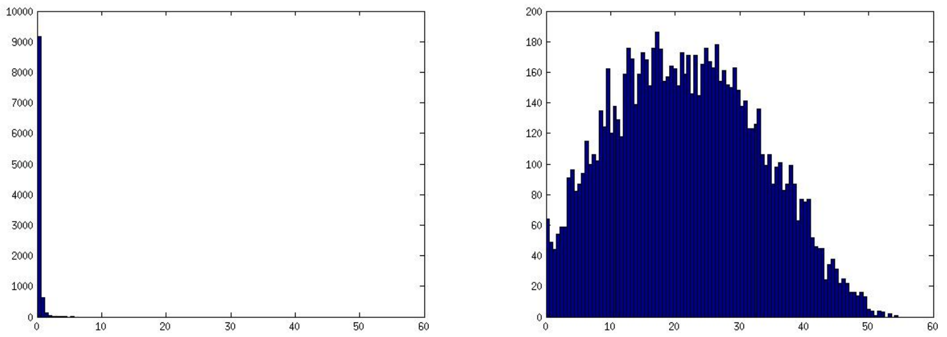


Figure 21: Histograms of $\|f_1(a) - f_1(a')\|_2$ (left) and $\|f_1(a) - f_1(b)\|_2$ (right) in the translation-invariance experiment.

4.7 Comparison to Deep CCA

Given realizations $\{(s_i^{(1)}, s_i^{(2)})\}_{i=1}^n$ of random variables $S(1)$ and $S(2)$, the deep CCA algorithm (see [6]) computes maps f_1' and f_2' so that the cross correlation between $f_1'(S^{(1)})$ and $f_2'(S^{(2)})$ is maximized. We implemented the deep CCA network described in [6] and applied it on the Toy dataset of Section 4.2, with the same network structure used in our experiment in Section 4.2. The diffusion map of the output of the deep CCA algorithm, presented in Figure 22, demonstrates that in this experiment the deep CCA algorithm fails to recover the common variable (the rotation angle of the bulldog); additional analysis indicates that the representation obtained by deep CCA in this experiment reflects the sensor specific superfluous variables (rotation angles of Yoda and the bunny), which we would like to discard.

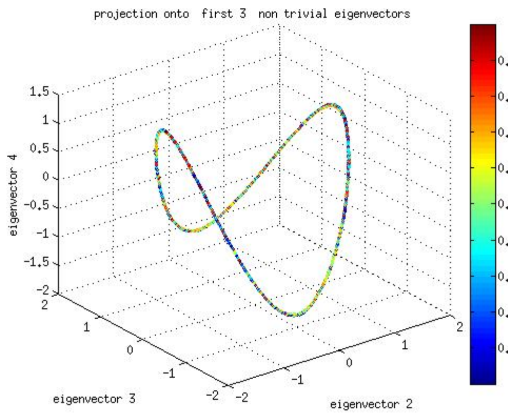


Figure 22: Diffusion embedding of the representation obtained by a deep CCA network for images from Camera 2 in the Toy dataset. The embedding does not appear to follow the value of the common variable.

4.8 Random projections and complex data models

In this section we discuss complex models of data generation; we replace the images that we have in the experiments above with unknown random projections of the images onto a lower dimensional space. These random projections "scramble" the data and remove much of the structure in it, the original images cannot be recovered without a good model of the data and the projections. The heavily distorted data is an example of complex data that can not be visualized as an image and that does not have the structure of an image; such complex data are obtained in compressed sensing (without the information that allows reconstruction) and in applications such as measurement of neural activity in the brain.

We repeated the experiment in Section 4.1 using the Person dataset, with each image replaced by a 100 dimensional random projection (the same random projections for all images in the same camera). The classification accuracy in this experiment was 96.6%, similar to the accuracy obtained on the original dataset before the random projections. The diffusion map of the representation is presented in Figure 23.

We also repeated the experiment in Section 4.2 using the Toy dataset, with each image replaced by a 100 dimensional random projection. The accuracy in this experiment was 93.66%, a small decrease compared to 95.9% in the original dataset before the random projections. However, in this case the embedding in Figure 24 (top) does not reflect a smooth transition of the color; a more detailed investigation in Figure 24 (bottom) reveals that each point in the embedding corresponds to two values of X. In other words, the algorithm has recovered the common-variable up to some ambiguity; it can still distinguish between different values of X, but for almost every x there is some very different $\tilde{x}$ that cannot be distinguished from x in the embedding. This phenomenon, to which we refer as "folding," is discussed in Section 4.9.

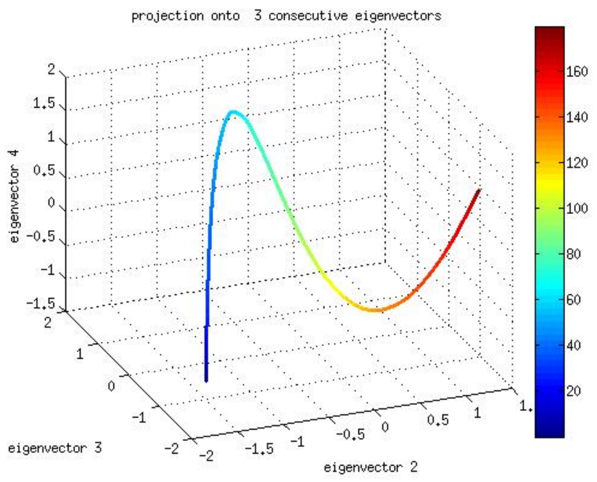
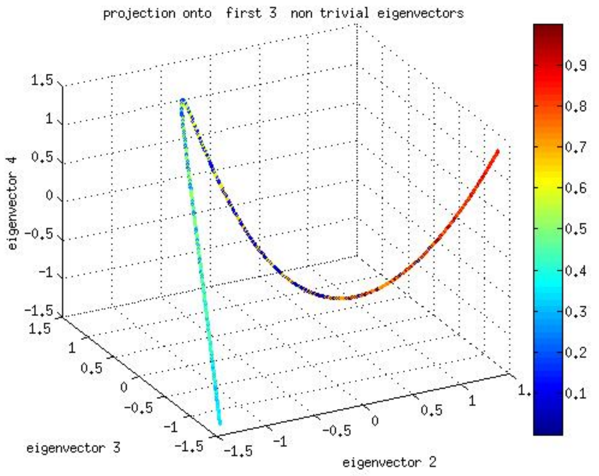


Figure 23: Embedding of images from the randomly projected Person dataset. The learned representation corresponds to the value of the common hidden variable.



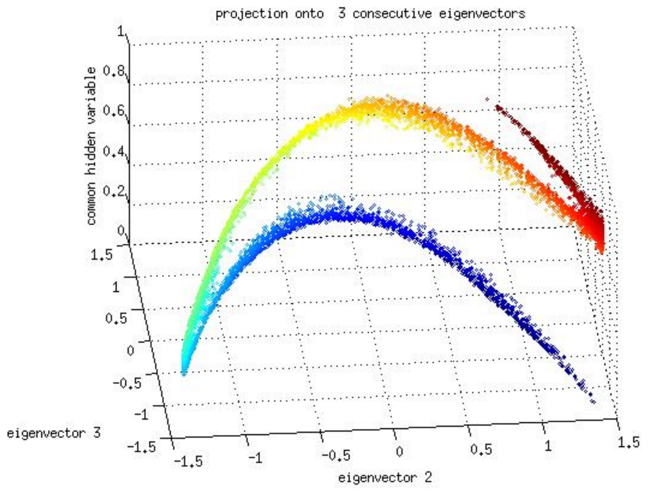


Figure 24: Top: embedding of the randomly projected Toy dataset. Bottom: foldings in the randomly projected Toy dataset: two axes are diffusion embedding coordinates as before, and one axis is the true value of the common variable X.

4.9 Non-injective maps and "folding"

The algorithm attempts to construct an approximate classifier which minimizes the loss function (13), and due to the special structure of the joint network $\mathcal{N}$, it also builds an approximate representation in $\mathcal{N}_1$. The minimization of the loss function defined in (13) on the datasets D_{pos} and D_{neg} does not strictly require the map f_1 to be injective in X (in the sense defined in Section 3.1); in some cases, the network may "fold" the representation so that it assigns the same representation to two (or more) values of X, but still distinguish between each value of X and most other values. In other words, a "folded" representation recovers the the common variable up to some ambiguity.

We only observed the "folding" phenomenon in the randomly projected Toy dataset in Section 4.8, and in the variation on the Bunny dataset in Section 4.3, when the number of layers was increased to three (not presented here).

4.10 Using additional information: Δx

In the formulation and examples presented thus far, data are presented to the algorithm in the form of "positive" pairs and "negative" pairs; the positive pairs are often simultaneous measurements which are related through coincidence and therefore share the same value of the common variable X, while the negative pairs are often measurements that were taken at different times and are therefore assumed to have different values of X with high probability. When additional information about the problem and the data is available, such information can be integrated into the algorithm to improve the results, as demonstrated in this section.

Suppose that we have a dataset of pairs of measurements $\{(s_i^{(1)}, s_i^{(2)})\}_{i=1}^n$ where $s_i^{(1)} = g_1(x_i, y_i)$ and $s_i^{(2)} = g_2(x_i', y_i)$, and that for each pair we have some distance in the common variable $\Delta x_i = \|x_i - x_i'\|$. We assign to each such pair a label $l(\Delta X)$ with the function $l : \mathbb{R}^+ \to [\frac{1}{2}, 1]$, a non decreasing function such that $l(0) = \frac{1}{2}$ and $l(\infty) = 1$. The "soft" labels l_i replace the strict distinction between "positive" pairs and "negative" pairs, so that the new labeled dataset $D = \{(s_i^{(1)}, s_i^{(2)}, l_i)\}_{i=1}^n$ replaces the datasets D_{pos} and D_{neg}. The network is trained to minimize the loss function $L(\theta)$ defined by the formula

$$L(\theta) = \sum_{(s^{(1)}, s^{(2)}, l) \in D} \left(l - \sigma \left(\|f_1(s^{(1)}) - f_2(s^{(2)})\|^2 \right) \right). \tag{36}$$

To demonstrate this scenario, we modified the Bunny dataset, and generated random pairs of images. Let $x_i, x_j \in [0, 1]$ be the rotation angles of the bunny in $s_i^{(1)}$ and $s_j^{(2)}$, we define the distance Δx_i by the formula

$$\Delta x_i = \min\{|x_i - x_i'|, 1 - |x_i - x_i'|\}, \tag{37}$$

so that $\Delta x_i \in [0, 0.5]$. We define the function l by the formula

$$l(\Delta x) = \frac{1}{2} + \min\left\{\frac{1}{2}, 4\Delta x^2\right\}, \tag{38}$$

a plot of the function l is presented in Figure 25.

Figure 26 presents the diffusion embedding of the representation, obtained from a three-layer network on this modified Bunny dataset with "soft labels".

.

.

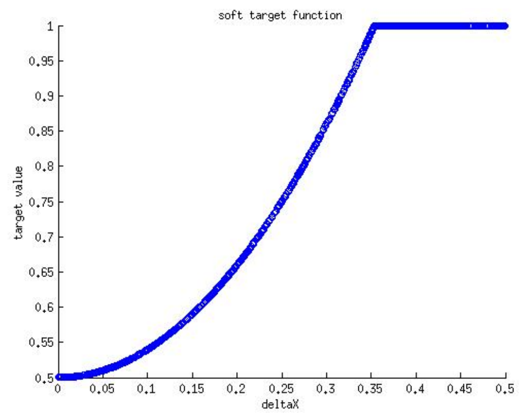


Figure 25: The label function $l(\Delta x)$.

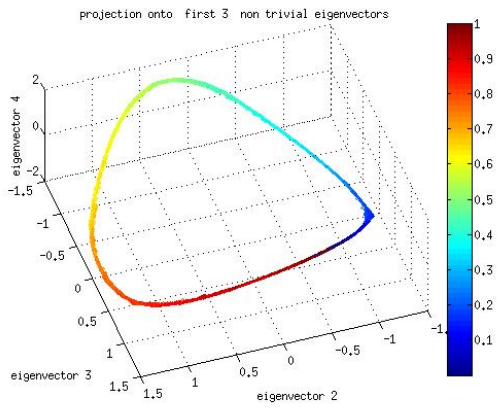


Figure 26: Diffusion embedding of the modified Bunny dataset.

5 Conclusions

A neural network based approach has been presented for the construction of invariant representations and for the recovery of common variables, in the absence of a model or labeled data. The joint network described in this manuscript learns the appropriate features and constructs the appropriate representation from examples of measurements that are "equivalent" or "related" via an appropriate form of coincidence and examples of measurements that are "not equivalent" or "unrelated"; a typical example for such sets of measurements are "related" measurements that are taken simultaneously by different sensors and "unrelated" measurements that are taken at different times. The joint network is designed to recover a meaningful representation in its sub-networks; in other words, the desired representation is obtained in what would typically be considered a "hidden layer" of a deep network.

Most of the experiments presented in this manuscript have been designed for ease for illustration and visualization; however, these experiments have many properties of complex data analysis problems. The potential application to much more complex data has been demonstrated in experiments with datasets that have been heavily "scrambled"' by random projections to lower dimensional spaces. In addition, we have not used standard preprocessing techniques, such as filtering, and we have not used standard network designs, such as convolution networks, that take advantage of known structure in the data; when partial information about the data is available, it can be incorporated into preprocessing steps and into th e network design to improve the performance of the network.

6 Acknowledgments

The authors would like to thank Raphy Coifman, Sahand N. Negahban, Andrew R. Barron and Ronen Talmon, for their help.

References

[1] R. R. Lederman and R. Talmon, "Common manifold learning using alternating-diffusion," 2014.

[2] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 689–696, 2011.

[3] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *Advances in neural information processing systems*, pp. 2222–2230, 2012.

[4] E. Oyallon and S. Mallat, "Deep roto-translation scattering for object classification," *arXiv preprint arXiv:1412.8659*, 2014.

[5] K. Sohn and H. Lee, "Learning invariant representations with local transformations," *arXiv preprint arXiv:1206.6418*, 2012.

[6] G. Andrew, R. Arora, J. Bilmes, and K. Livescu, "Deep canonical correlation analysis," in *Proceedings of the 30th International Conference on Machine Learning*, pp. 1247–1255, 2013.

[7] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

[8] A. Ng, "Unsupevised feature learning and deep learning, stnanford class cs294." `http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial`. Accessed: 2010-09-30.

[9] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1139–1147, 2013.

[10] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8609–8613, IEEE, 2013.

[11] S. J. Wright and J. Nocedal, *Numerical optimization*, vol. 2. Springer New York, 1999.

[12] R. Rojas, *Neural networks: a systematic introduction.* Springer Science & Business Media, 1996.

[13] S. S. Lafon, *Diffusion maps and geometric harmonics.* PhD thesis, Yale University, 2004.

[14] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," 2004.

A Appendix: technical background

The purpose of the following sections is to provide a brief review of certain standard definition and algorithms used in this paper.

A.1 The diffusion maps algorithm

A brief description of the procedure to obtain a diffusion map of a dataset is presented in Algorithm 2; a more detailed description can be found, inter alia, in [13].

```
**Data**: $\{x_j\}_{j=1}^n,\ x_j \in \mathbb{R}^d$.
**Result**: diffusion embedding $\{\tilde{x}_j\}_{j=1}^n,\ \tilde{x}_j \in \mathbb{R}^{n-1}$.
Compute affinity matrix $W$, such that $W_{i,j} = \exp\left(-\frac{\|x_i - x_j\|_2^2}{\epsilon}\right)$.
Compute the diagonal matrix $D$ such that $D_{i,i} = \sum_{j=1}^n W_{i,j}$ .
Compute the symmetric graph Laplacian $A = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$.
Compute the eigenvectors $v_0, ..., v_{n-1}$ and eigenvalues $\lambda_0, ..., \lambda_{n-1}$ of $A$.
Compute $n-1$ weighted vectors $\varphi_1, ..., \varphi_{n-1}$, such that $\varphi_i(j) = \frac{v_i(j)}{v_0(j)}$.
Compute the diffusion embedding $\tilde{x}_j = (\varphi_1(j), ..., \varphi_{n-1}(j))$.
```

Algorithm 2: diffusion maps

A.2 Equivalence Classes and quotient space

Definition A.1. *Let S be a set, $a \in S$, and $\sim$ be an equivalence relation over S. The equivalence class of a with respect to $\sim$ is*

$$[a] := \{x \in S : a \sim x\}. \tag{39}$$

Definition A.2. *The quotient set of S by $\sim$ is the set of all equivalence classes of elements in S:*

$$S/\sim = \{[a] : a \in S\}. \tag{40}$$

A.3 Group actions and invariant representations

Definition A.3. *Let $(G, \cdot)$ be a group and X be a set. A (left) group action of G on X is a function $\varphi : G \times X \to X$ such that*

- $\forall a, b \in G, x \in X,\ \varphi(ab, x) = \varphi(a, \varphi(b, x))$
- $\forall x \in X, \varphi(1, x) = x$.

Usually $\varphi(a, x)$ is denoted by $a.x$.

Let S be a set and G be a group that acts on S. Let $h : S \to S$ be a map with the property that for every $g \in G, h(g.X) = h(1.X)$. In this case we say that $h(S)$ is a G-invariant representation.