Neural Networks 91 (2017) 76–84

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Hopfield networks as a model of prototype-based category learning: A method to distinguish trained, spurious, and prototypical attractors

Chris Gorman *, Anthony Robins, Alistair Knott

Department of Computer Science, University of Otago, 133 Union Street East, Dunedin 9016, New Zealand

ARTICLE INFO

Article history:
Received 10 November 2016
Received in revised form 25 March 2017
Accepted 14 April 2017
Available online 25 April 2017

Keywords:
Prototype theory
Cognition
Spurious attractors

ABSTRACT

We present an investigation of the potential use of Hopfield networks to learn neurally plausible, distributed representations of category prototypes. Hopfield networks are dynamical models of autoassociative memory which learn to recreate a set of input states from any given starting state. These networks, however, will almost always learn states which were not presented during training, so called spurious states. Historically, spurious states have been an undesirable side-effect of training a Hopfield network and there has been much research into detecting and discarding these unwanted states. However, we suggest that some of these states may represent useful information, namely states which represent *prototypes* of the categories instantiated in the network's training data. It would be desirable for a memory system trained on multiple instance tokens of a category to extract a representation of the category prototype. We present an investigation showing that Hopfield networks are in fact capable of learning category prototypes as strong, stable, attractors without being explicitly trained on them. We also expand on previous research into the detection of spurious states in order to show that it is possible to distinguish between trained, spurious, and prototypical attractors.

Introduction

Many models of categorization focus on supervised learning and similarity metrics and do not incorporate object prototypes, instead basing category representations on exemplar theory. Kruschke (2008) discusses several of these models in depth, but in general they are given a stimulus, determine the similarity between the stimulus and an exemplar stored in memory, and then determine the category. There are several ways to determine the category, e.g. the probability of a category given an input can be calculated based on the similarity of the stimulus to the exemplar and how frequently an exemplar is associated with a category.

Recently, we introduced an unsupervised model of categorization based on prototype theory called DPAN (Gorman & Knott, 2016). DPAN is focused on the learning of different levels of categorization (e.g. basic and subordinate levels) and attention to the properties of a token object which make it unusual as an instance of a type. Since it is a prototype-based model, DPAN does not take exemplar theory into consideration. While this model is successful in utilizing an inhibition of return operation to learn subordinate-level categories, it relies on highly localist interpretations of input data with questionable neural plausibility. For example, in an input vector an active bit may represent the presence of an arbitrary, high-level feature (e.g. "has brown fur"). Additionally, recent neuroimaging, lesion, and cognitive studies (Blonder et al., 2004; Hanson, Matsuka, & Haxby, 2004; Haxby, 2001; Martin, Chao, & Haxby, 1999; Rakison & Yermolayeva, 2010) have shown that categories are internally represented as a large, distributed feature space rather than as discrete "category units" per se. These issues necessitate a more robust, distributed model of categorization.

To that end, we have explored the use of Hopfield networks (Hopfield, 1982) as a model of category learning based on prototype theory. Hopfield networks are dynamical, recurrent, fully connected (with no self-connections) autoassociative artificial neural networks which learn a set of input patterns. During training, the network is presented with a series of bipolar (i.e. either −1 or 1) vectors as input patterns and the network then updates its weights to reproduce these patterns. A Hopfield network learns a set of stable states such that, from some starting state, the network updates its state and traverses an energy surface until it reaches a local minimum. The final state at this local minimum can either be one which was presented during training (a *trained state*) or one which was not (an *untrained state*). The states the network reaches between its initial and final states are called *intermediate*

* Corresponding author.
E-mail addresses: cgorman@cs.otago.ac.nz (C. Gorman), anthony@cs.otago.ac.nz (A. Robins), alik@cs.otago.ac.nz (A. Knott).

http://dx.doi.org/10.1016/j.neunet.2017.04.007
0893-6080/

states. These intermediate states form a *basin of attraction* around the final state. The energy of the network slides into this basin until it reaches an attractor with the lowest energy. A trained network (with asymmetrical connections) may also oscillate between intermediate states in the basin indefinitely.

Historically, untrained states have been considered undesirable. In general, a good associative memory network should have high capacity, form large basins of attraction, be content addressable, and retrieve only explicitly stored states. Hopfield networks do not have a very high capacity (Wu, Hu, Wu, Zhou, & Du, 2012) and they do allow the retrieval of untrained states. Hopfield networks are typically trained to recall tokens (rather than types). In this context, an untrained state represents an error in learning. If the network's goal is to recall a token individual from a noisy input, then recalling a state the network was never trained on is a major problem with the model. Research into these networks has produced methods to identify and differentiate between trained and untrained states as well as remove them from the network altogether (see e.g. Abe, 1993; Robins & McCallum, 2004; van Hemmen, 1997). In the typical use case, identifying and discarding untrained states is definitely beneficial, but what if some of these untrained states represent useful information?

We hypothesize that, as a distributed model of associative memory, Hopfield networks can learn representations of individual stimuli *as well as* their basic-level category (Rosch, 1973) in the form of an object prototype. That is, learning tokens *and* types from the input tokens alone. Importantly, the prototype is gleaned from the information inherent to the execution of a Hopfield network and the statistical similarities inherent to the input stimuli. Therefore, we expand our earlier definitions such that untrained states can either be *spurious* or *prototypical*. Spurious states are the classic case of a pattern which was not presented during training and is an undesirable final state for a trained Hopfield network. Prototype states are those which do not correspond to input patterns, but rather represent the categorical prototypes of those patterns. We also propose that, using the methodology outlined in Robins and McCallum (2004), we will be able to use the stability profile of a final state to determine whether it is trained, prototypical, or spurious.

Methodology

Our model is an asymmetrically connected thermal perceptron learning rule Hopfield network trained on a set of synthetic, bipolar input patterns. We explain the structure of the input data, the network architecture, and the training regime in the following sections.

Input data

The key component of our hypothesis is that the network should learn a category prototype as a stable attractor given a set of token category individuals. To facilitate that, each training item is stochastically generated from an initial prototype.

We start by generating one prototype per class, an example of which is provided in Fig. 1. When generating the prototypes we need to ensure that the category members have a high level of intra-category similarity and a high level of inter-category dissimilarity. To simulate this while maintaining neurally plausible distributed object representations, we enable features within each prototype based on a Gaussian window of probabilities (see Fig. 2). The size of the window, i.e. the number of points in the Gaussian, is set to be the number of neurons divided by the number of prototypes, and the standard deviation is set to the window size divided by ten. The Gaussian distribution is initialized within the vector such that each feature contains a value between zero and

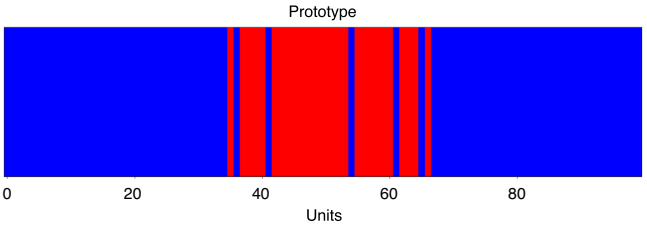

Fig. 1. An example of an object prototype. This figure is a heat map of a 100 unit vector such that −1 is blue and 1 is red. Since the prototype is a vector, the height of this figure is arbitrary for visualization purposes. Each column in the figure corresponds to one feature of the vector.

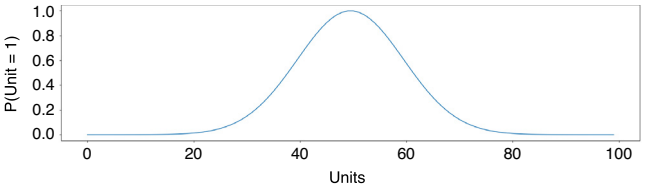

Fig. 2. The Gaussian window which was used to generate the prototype in Fig. 1. Each integer on the X axis corresponds to a feature at that index in the prototype. The Y axis represents the probability that the feature will be set to 1 in the prototype.

one representing the probability that the value of the output prototype at that index is equal to 1, otherwise it is −1. This produces prototypes which contain a central cluster of features and a small amount of noise.

For example, in Fig. 1 the center of the distribution is around unit 50. The indices surrounding it form a Gaussian distribution of probabilities. The value at each index is provided as a probability to a Bernoulli trial, the result of which is stored in the output vector. This new vector, representing a category prototype, contains a noisy cluster of features which is then used to generate individual patterns as members of that category.

When generating prototypes, we allow for a certain level of overlap. If the newly generated prototype center lies within *WindowSize*/4 of any other prototype center, it is recalculated. In practice, whether or not the prototypes overlap at all is a function of the random placement of the prototype center.

To create the actual training items (instance tokens) for the network, we simply duplicate the prototype and randomly mutate it. To do so, we select 20%[1] of the features in the vector, randomly choose the indices to alter, and then flip the bits. The resulting training items (Fig. 3) have a set of strongly correlated features, inherited from the prototype, along with a set of random features, distinguishing one object from another.

The instance tokens also typically do not contain the entirety of the prototype. That is, since the prototype bits can be flipped, the instance tokens will likely have a few features which are different from their category prototype. Importantly, the prototypes themselves are never presented to the network. When testing the network, we are able to compare its final state to the input patterns as well as the prototype to determine if it is trained, prototypical, or spurious.

Network architecture and training regime

The Hopfield model is arranged as a single layer of k McCulloch–Pitts neurons with weight matrix[2] $\mathbf{W}$. The weight

[1] See section *Discussion* regarding this decision

[2] By convention, capitalized boldface symbols are matrices and lowercase boldface symbols are vectors.