ChatGPT

Study 1: Paradigm Overview and Representational Disentanglement Test

Paradigm summary

In Study 1, we explore how coordination demands shape the way reinforcement-learning agents internally represent both their **environment** and their **social partners**. The experimental paradigm is adapted from a multi-agent grid-world foraging task in which agents traverse a two-dimensional world to collect stochastically appearing resources. Each agent is controlled by a separate neural network and is assigned a unique **alien face icon** generated from a small set of latent factors (for example, colour, size and group identity). Resource icons are also generated from latent factors (e.g., colour and size), analogously to real-world entities that vary along multiple dimensions. Once an agent has collected enough resources, it can enter a "ready-to-interact" state and fire an **interaction beam** toward another agent. When two agents interact in this way, each receives a reward determined by the similarity of their inventories; the payoff matrix is calculated from the counts of resource types held by each agent [1]. This design encourages agents to align their actions around collecting similar resources so that they can maximise the interaction reward. By varying the slope of the relationship between resource similarity and reward magnitude, we can manipulate the **coordination demand**: steeper slopes impose stronger incentives to coordinate on shared resource types.

The paradigm allows us to investigate two complementary hypotheses:

1. **Disentanglement in internal representations.** When coordination requires accurate mutual understanding, agents may benefit from organising knowledge in a **factorised** way, where each latent attribute (e.g., colour or size) can be treated independently rather than as entangled combinations. Such factorisation supports generalisation and stereotyping along single dimensions. Prior work has shown that social alignment demands in communication can foster the emergence of structured language and disentangled latent codes [2]. Study 1 extends this idea to coordination in strategic interactions.
2. **Behavioural generalisation to novel partners.** After training, each agent will be evaluated with two novel agents whose icons share all but one latent factor with the focal agent. On the differing dimension, one novel partner is more similar to the focal agent than the other. We hypothesise that the focal agent will preferentially approach the more similar partner, and that this bias will be mediated by the degree of representational disentanglement measured by our test.

This document focuses on how to **measure representational disentanglement** in the trained agents. It provides a high-level description of the paradigm and omits any implementation details of the underlying game. Pseudocode for the disentanglement test is provided below.

Rationale for linear separability

The central question is whether agents' internal representations encode latent factors (such as colour or size) in separable dimensions. A common way to assess this is to see if a **linear classifier** can recover each factor from hidden activations. If a simple linear read-out can decode the value of a factor, it implies that the factor is represented in an approximately independent subspace [2]. This approach has been widely used to measure abstraction and factorisation in emergent communication and neural representations.

Procedure for the representational disentanglement test

The test proceeds after training: the agent's parameters are frozen and it is no longer interacting with the environment. We probe the agent's encoder by systematically presenting icons for resources or social partners and recording the hidden activations. These activations are then analysed to determine how linearly separable the underlying factors are.

1. Collect hidden activations

1. **Enumerate latent configurations.** Define the latent factors for the icons you wish to probe (e.g., `colour: ['red', 'green']`, `size: ['small', 'large']`, etc.). Generate all possible combinations of these factor values.
2. **Generate stimuli.** For each combination of factor values, use a function `generate_icon` to create a visual representation (icon) with those attributes. This could be an image or a one-hot encoded vector, depending on your network's input format.
3. **Forward pass through the frozen encoder.** Feed each generated icon through the agent's encoder (or the hidden layer of interest) and record the resulting activation vector.
4. **Label assignment.** Associate each activation vector with the values of its latent factors so that these values can serve as labels in subsequent decoding.

2. Compute linear separability

For each latent factor:

1. **Prepare labels.** Extract the value of that factor from the recorded labels to form a label vector `y` for classification.
2. **Split data.** Divide the activation vectors and labels into a training set and a held-out test set (e.g., 70/30 split). Use stratified sampling to ensure all factor values appear in both sets.
3. **Train a linear classifier.** Fit a logistic regression classifier (or another linear model) on the training set to predict the factor value from the hidden activations.
4. **Evaluate performance.** Compute the classifier's accuracy on the test set. High accuracy indicates that the factor is encoded in a linearly separable manner.
5. **Aggregate across factors.** Repeat for each latent factor and compute the average accuracy or record the distribution of accuracies. A higher average accuracy indicates stronger overall disentanglement.

Pseudocode for the test

The following pseudocode illustrates how to implement the representational disentanglement test. It assumes access to a trained agent with a frozen encoder, a dictionary of latent factors, and a function `generate_icon` that produces input representations from factor configurations. The code uses logistic regression as a simple linear classifier and stratifies the train/test split to ensure balanced classes.

```
from itertools import product
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# latent_factors: dict mapping each factor name to a list of possible values
# generate_icon: function that maps a configuration (dict) to an input
representation
# agent.encoder: frozen encoder that outputs hidden activations

def collect_hidden_activations(agent, latent_factors, generate_icon):
    activations = []
    labels = []
    factor_names = list(latent_factors.keys())
    # Enumerate all possible combinations of factor values
    for combo in product(*latent_factors.values()):
        config = {factor_names[i]: combo[i] for i in range(len(combo))}
        icon = generate_icon(config)
        h = agent.encoder(icon)        # forward pass through frozen encoder
        activations.append(h)
        labels.append(config)
    return np.array(activations), labels

def compute_linear_separability(activations, labels, latent_factors):
    results = {}
    for factor_name in latent_factors:
        # Extract labels for the current factor
        y = np.array([lbl[factor_name] for lbl in labels])
        # Train/test split with stratification
        X_train, X_test, y_train, y_test = train_test_split(
            activations, y, test_size=0.3, stratify=y, random_state=0
        )
        # Fit a logistic regression classifier (linear model)
        clf = LogisticRegression(max_iter=1000)
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results[factor_name] = acc
    mean_accuracy = float(np.mean(list(results.values())))
```

```
    return results, mean_accuracy

# Example usage:
# activations, labels = collect_hidden_activations(trained_agent,
latent_factors, generate_icon)
# per_factor_accuracy, overall_disentanglement = compute_linear_separability(
#     activations, labels, latent_factors)
```

Interpreting the results

Each entry in `results` gives the accuracy of the linear classifier for a particular latent factor, quantifying how well that factor can be decoded from the hidden representation. The `mean_accuracy` summarises the overall degree of disentanglement across factors. Higher values indicate that the agent's representations encode each factor in separable subspaces, consistent with the hypothesis that coordination pressures promote structured, generalisable internal categories [2].

References and further reading

- **Interaction beam and reward structure:** In the grid-world foraging task, agents receive rewards by firing an interaction beam at a partner; the reward for each agent equals the expected payoff computed from the counts of resource types in their inventories [1]. This incentivises agents to coordinate their resource-collection strategies.
- **Linear separability as a measure of modularity:** Prior research has shown that context-dependent tasks can be decomposed into linearly separable sub-tasks and that measuring linear separability is a useful way to evaluate the modularity of learned representations [2].
- **Emergent communication and factorisation:** Studies of emergent language in multi-agent systems demonstrate that social alignment demands lead to structured communication and disentangled latent codes [3]. This motivates the expectation that coordination pressures in our paradigm will promote factorised internal representations.

---

[1] Learning Through Social Interactions and Learning to Socially Interact in Multi-Agent Learning
https://naomi.princeton.edu/wp-content/uploads/sites/744/2024/01/Dissertation_Multi_agent_learning.pdf

[2] Modular representations emerge in neural networks trained to perform context-dependent tasks - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC11482777/

[3] Compositionality and Generalization In Emergent Languages - ACL Anthology
https://aclanthology.org/2020.acl-main.407/