Tutorial 9: Unsupervised learning; uninformed search

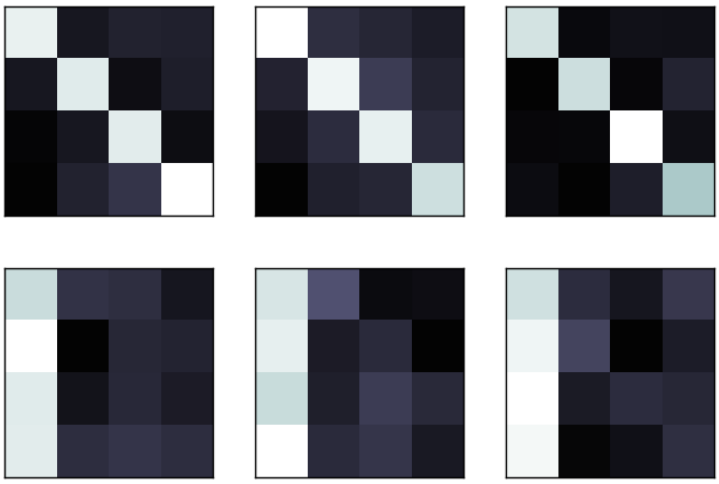
Unsupervised learning: Concepts

1. Explain what **principal component analysis** does.
2. Explain how k**-means clustering** works.
3. Explain what an **autoassociative network** is.

Unsupervised learning: Lab

4. **Unsupervised learning with principal component analysis (PCA)**
   In this exercise, you'll get some experience with the PCA function defined in Python's `scikit-learn` library.

   Open the file `pca.py` (from the Tutorials web page). This script defines some training data in 16 dimensions that can be visualised as 4×4 images: examples are shown below.



   There are two categories of training item: a diagonal bar (top row) and a vertical bar (bottom row). A component of noise is added to each item, which can be changed with the variable `noise_level`. (This is set to 0.1 in the above images.)

   (a) The function `show_training_images` displays example training items as images. Run this function for various values of `noise_level` to get a feel for how noise obscures the two categories of training item.

(b) The function `plot_raw_data_in_sampled_dimensions` plots the training items in randomly-selected pairs of dimensions. Set `noise_level` to 0.7 and run this function. As you will see, the categories are hard to distinguish in the presence of this level of noise.

(c) Principal component analysis (PCA) finds the dominant correlations in a dataset and defines new axes for the data based on these correlations. What are the dominant correlations in the training set?

(d) The `pca()` function finds the principal components of variance in the dataset and plots the data in terms of the first two principal components. Run the function to see what happens. Why are the patterns easier to distinguish from one another when plotted this way?

5. **K-means clustering**
`pca.py` also includes a function for running k-means clustering on the data represented in the two new axes, and plotting the result (in black and white). Run `kmeans_test()` to show that an unsupervised clustering technique does a reasonable job of identifying the two patterns in the space found by PCA.

6. **Autoassociative networks**
A backprop network can find correlations in a dataset without supervision, if it is trained to map each datapoint onto itself. A network of this type is called an **autoassociative network**. The file `backprop-network.py` has another implementation of the backprop network you saw with Lech. The `demo()` function in that file defines a network with two inputs and one output, and trains it on the logical function AND, displaying its error on the training set.

(a) Run the `demo()` function, so you can see what it does.

(b) Then alter the `demo()` function, to create a simple autoassociative network for yourself. If your network has n input units, it should have n output units, and the number of hidden units should be less than n, so that the hidden layer can learn a 'compressed' version of the inputs. Create some training data for your network by hand, where the outputs are the same as the inputs. Train the network, and see if it can learn to map the datapoints onto themselves.

(c) Show that your autoassociative network's error on the training set increases the more 'compression' the hidden layer has to do.

(d) **(Optional, but nice.)** Look again at `pca.py`. This file also contains a function `train_autoassociative_network_on_patterns()` that trains an autoassociative network with two hidden units on the image data.

i. Set `noise_level` to zero, and run the function a few times. Its performance is somewhat variable—why do you think this is?

ii. The function also shows images representing the *weights* of the two hidden units at the end of training. When the network's training is successful, what do these weights represent?

Uninformed search

1. Give an example of a problem that requires some **look-ahead search**. For your example, you should specify a **state space**, including an **initial state**, a **goal state**, and a set of possible **intermediate states**.

2. In AI search algorithms, the agent's actions are represented as operations that *transition from one state to another.*

   (a) Explain how this is formalised by the SUCCESSOR-FN described on p9 of Lecture 17.

   (b) Explain how the successor function, plus an initial state, formally define a state space.

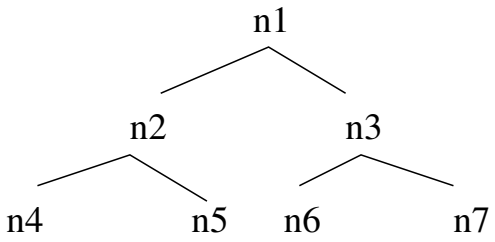
3. Here's an old puzzle. A man with a boat needs to cross a river with a hen, a fox and some corn. He can only take one thing with him in the boat at a time. If he leaves the fox and the hen by themselves, the fox will eat the hen. If he leaves the hen and the corn, the hen will eat the corn. What should he do?

   (a) Solve the problem.

   (b) Formulate a state space for the problem.

   (c) When you solved the problem in (a), did it feel at all like you were building and searching a state space?

4. **Quiz:** Consider the state space below.



   (a) What order would the nodes be encountered if the graph was searched

      i. depth first left-to-right?

      ii. breadth-first right-to-left?

   (b) For each of the above search strategies, define an algorithm that searches the graph using the notion of a **fringe** of nodes to be expanded.