Understanding Deep Learning

Chapter 13: Graph Neural Networks

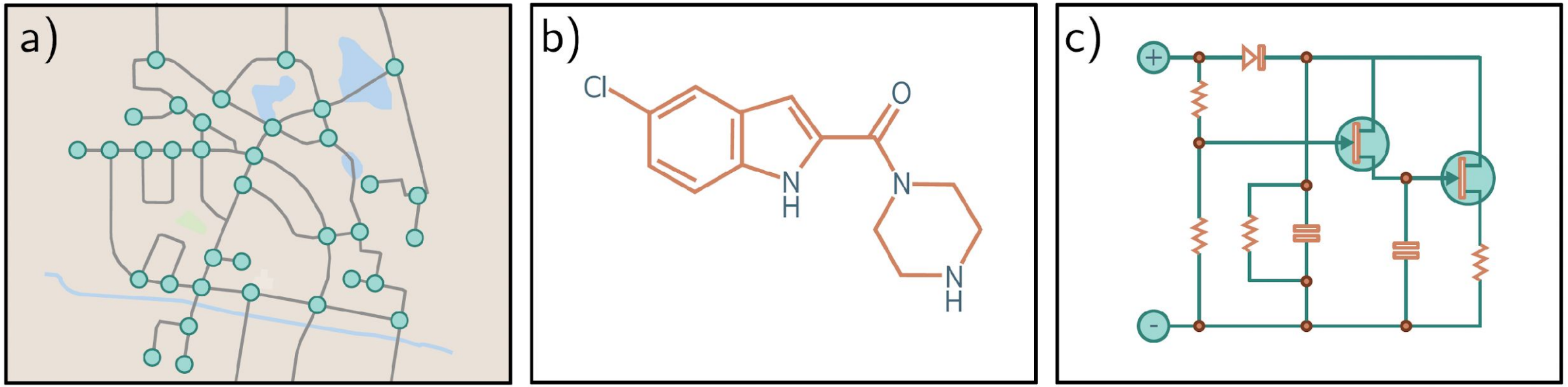

Figure 13.1 Real-world graphs. Some objects, such as a) road networks, b) molecules, and c) electrical circuits, are naturally structured as graphs.

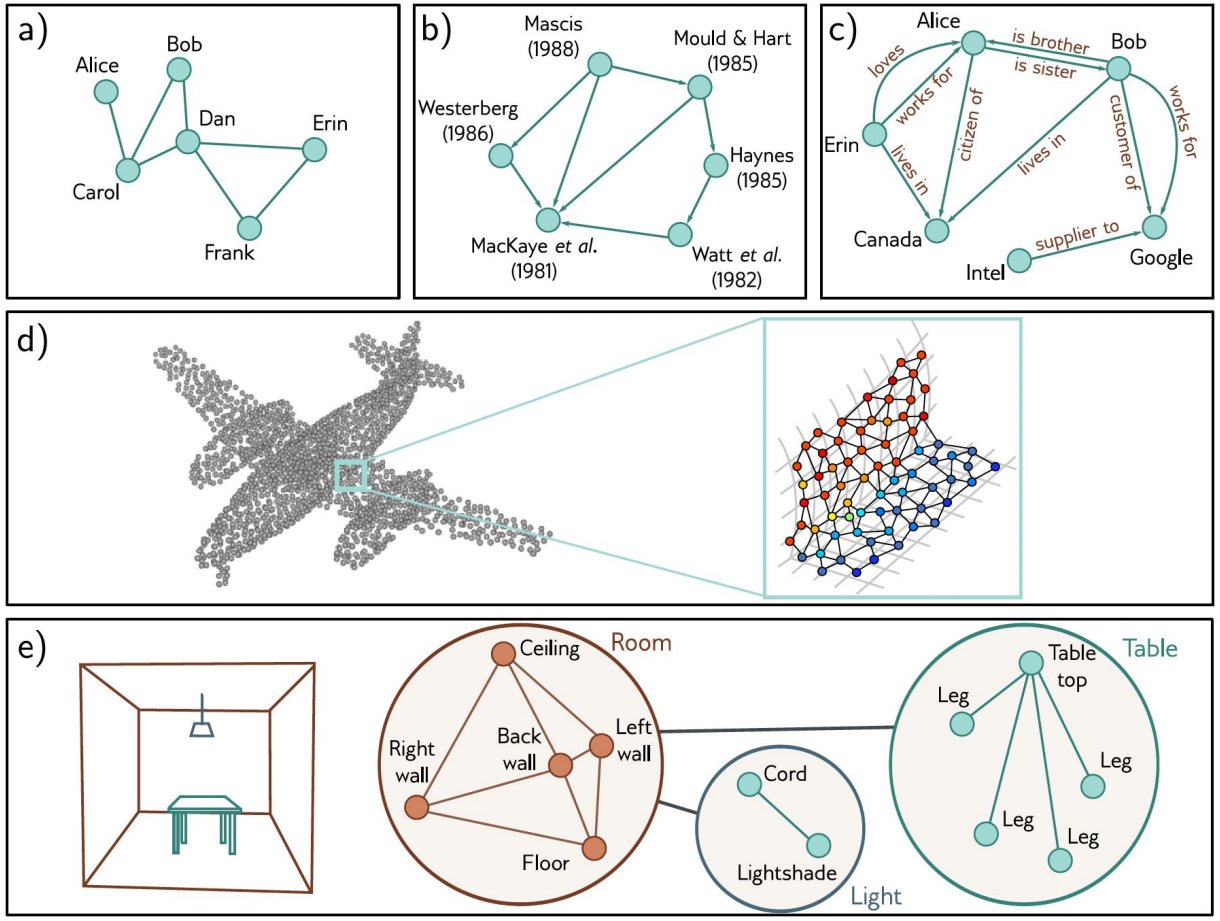

Figure 13.2 Types of graphs. a) A social network is an undirected graph; the connections between people are symmetric. b) A citation network is a directed graph; one publication cites another, so the relationship is asymmetric. c) A knowledge graph is a directed heterogeneous multigraph. The nodes are heterogeneous in that they represent different object types (people, places, companies) and multiple edges may represent different relations between each node. d) A point set can be converted to a graph by forming edges between nearby points. Each node has an associated position in 3D space, and this is termed a geometric graph (adapted from Hu et al., 2022). e) The scene on the left can be represented by a hierarchical graph. The topology of the room, table, and light are all represented by graphs. These graphs form nodes in a larger graph representing object adjacency (adapted from Fernández-Madrigal & González, 2002).

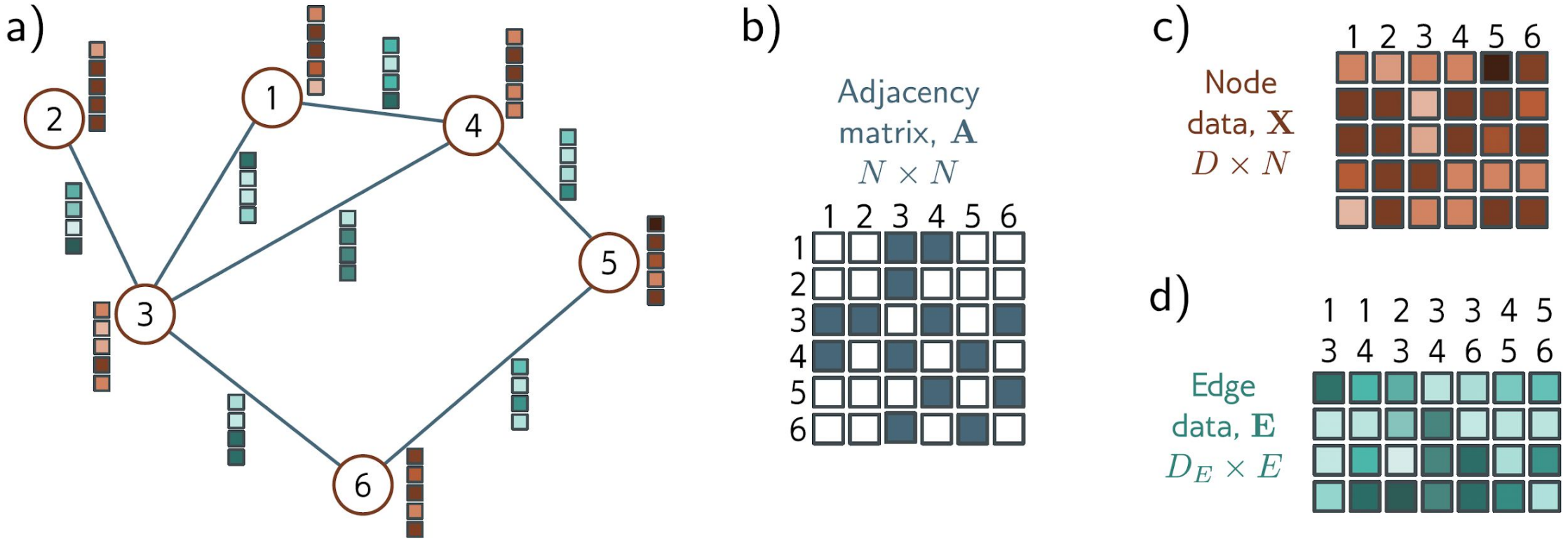

Figure 13.3 Graph representation. a) Example graph with six nodes and seven edges. Each node has an associated embedding of length five (brown vectors). Each edge has an associated embedding of length four (blue vectors). This graph can be represented by three matrices. b) The adjacency matrix is a binary matrix where element (m, n) is set to one if node m connects to node n. c) The node data matrix $\mathbf{X}$ contains the concatenated node embeddings. d) The edge data matrix $\mathbf{E}$ contains the edge embeddings.

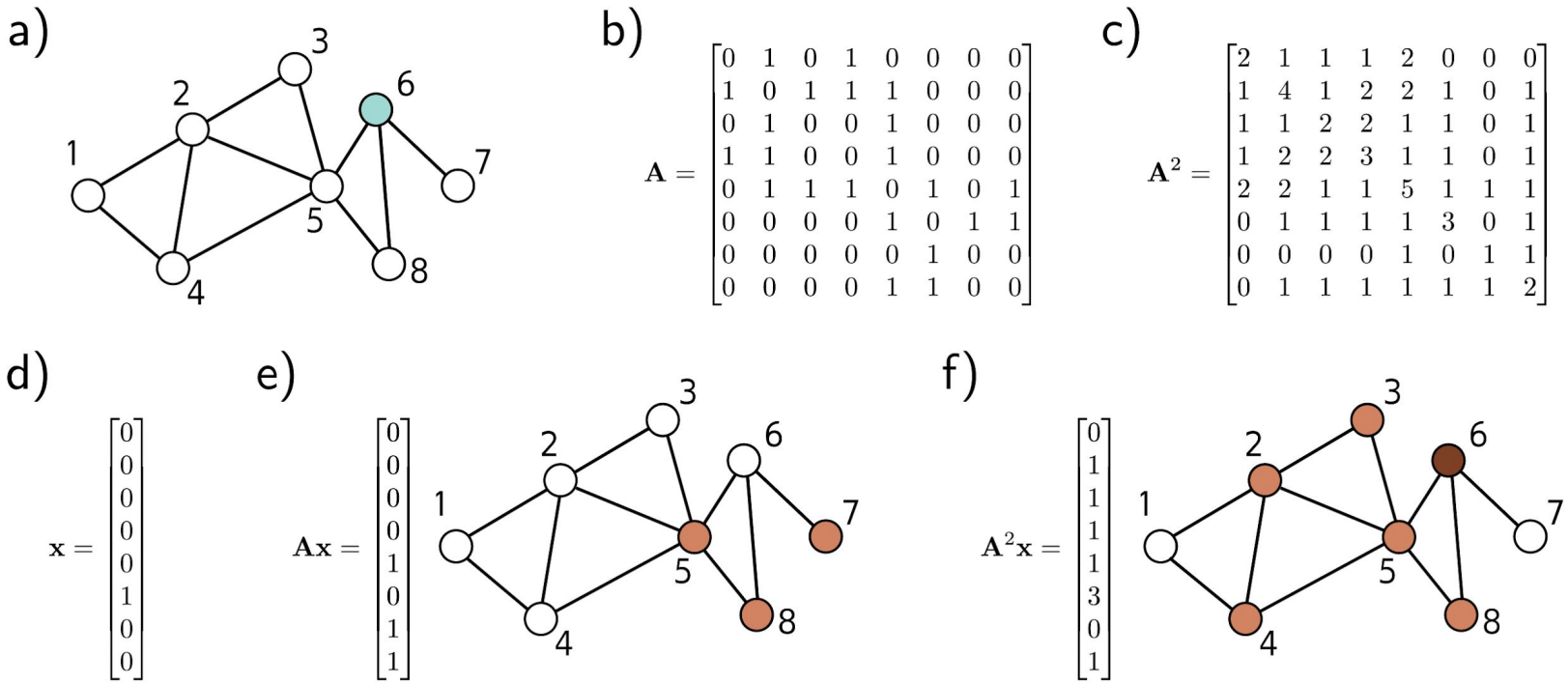

Figure 13.4 Properties of the adjacency matrix. a) Example graph. b) Position (m, n) of the adjacency matrix $\mathbf{A}$ contains the number of walks of length one from node m to node n. c) Position (m, n) of the squared adjacency matrix $\mathbf{A}^2$ contains the number of walks of length two from node n to node m. d) One hot vector representing node six, which was highlighted in panel (a). e) When we pre-multiply this vector by $\mathbf{A}$, the result contains the number of walks of length one from node six to each node; we can reach nodes five, seven, and eight in one move. f) When we pre-multiply this vector by $\mathbf{A}^2$, the resulting vector contains the number of walks of length two from node six to each node; we can reach nodes two, three, four, five, and eight in two moves, and we can return to the original node in three different ways (via nodes five, seven, and eight).

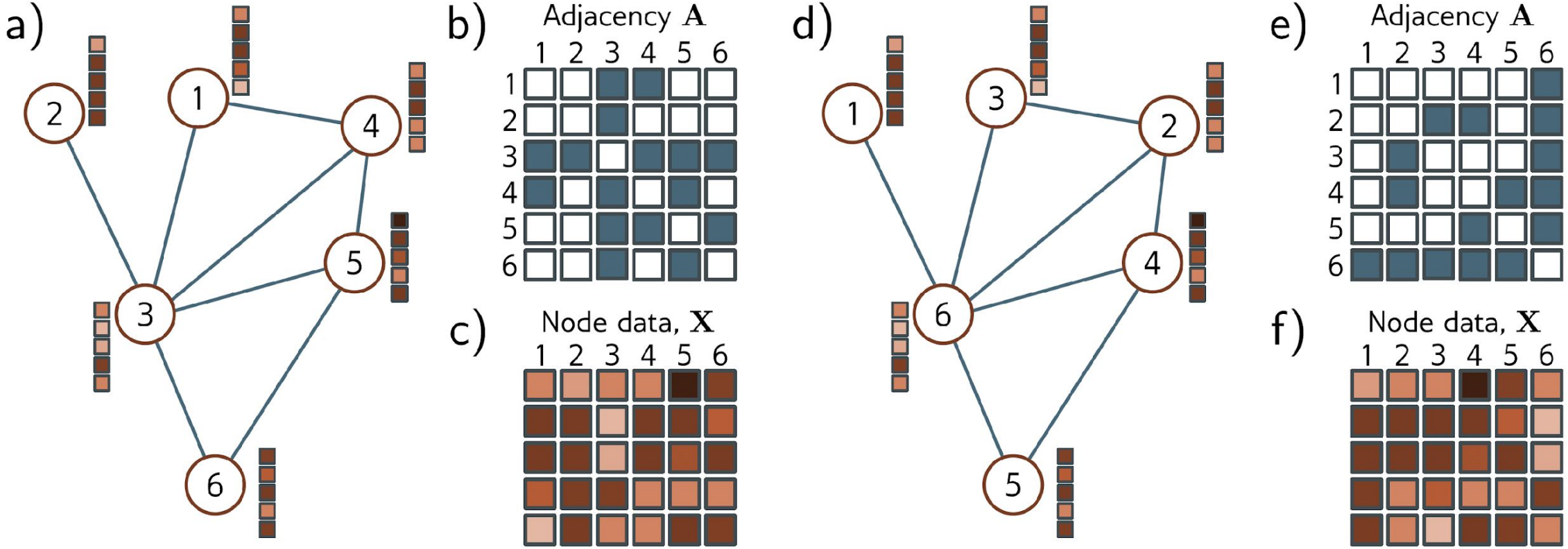

Figure 13.5 Permutation of node indices. a) Example graph, b) associated adjacency matrix and c) node embeddings. d) The same graph where the (arbitrary) order of the indices has been changed. e) The adjacency matrix and f) node matrix are now different. Consequently, any network layer that operates on the graph should be indifferent to the ordering of the nodes.

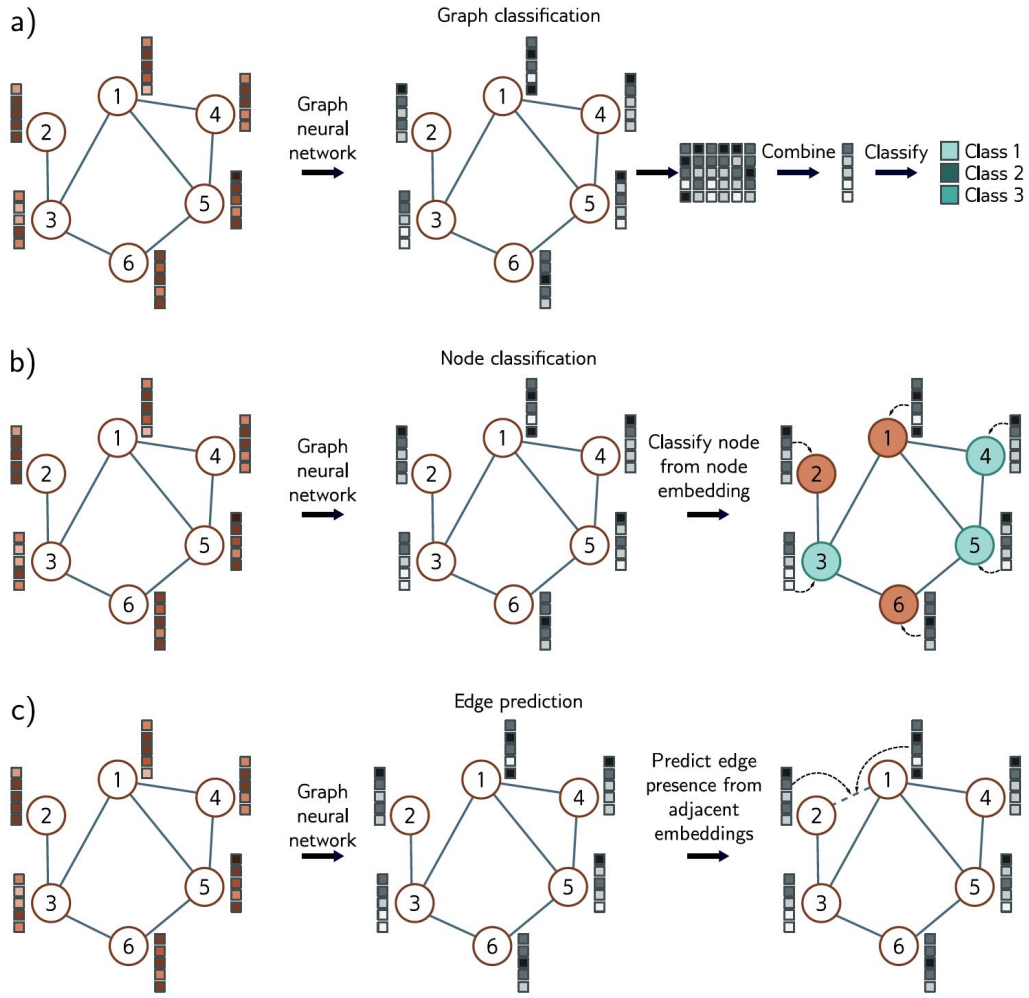

Figure 13.6 Common tasks for graphs. In each case, the input is a graph represented by its adjacency matrix and node embeddings. The graph neural network processes the node embeddings by passing them through a series of layers. The node embeddings at the last layer contain information about both the node and its context in the graph. a) Graph classification. The node embeddings are combined (e.g., by averaging) and then mapped to a fixed-size vector that is passed through a softmax function to produce class probabilities. b) Node classification. Each node embedding is used individually as the basis for classification (cyan and orange colors represent assigned node classes). c) Edge prediction. Node embeddings adjacent to the edge are combined (e.g., by taking the dot product) to compute a single number that is mapped via a sigmoid function to produce a probability that a missing edge should be present.

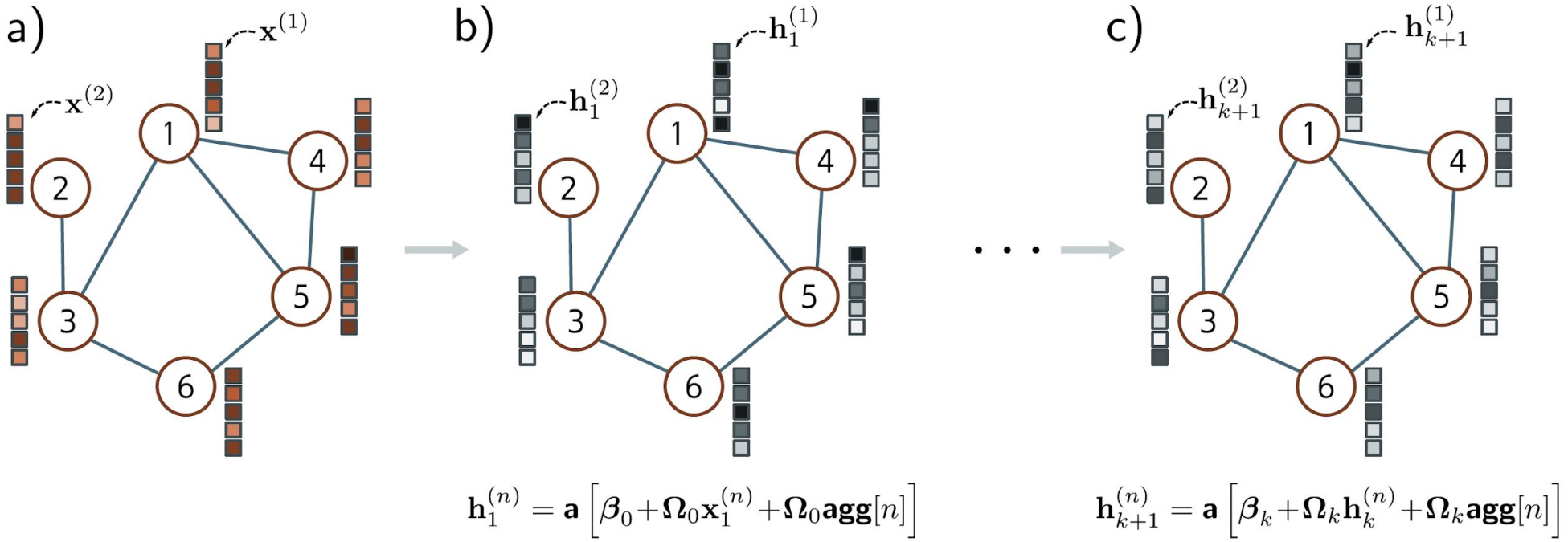

Figure 13.7 Simple Graph CNN layer. a) Input graph consists of structure (embodied in graph adjacency matrix $\mathbf{A}$, not shown) and node embeddings (stored in columns of $\mathbf{X}$). b) Each node in the first hidden layer is updated by (i) aggregating the neighboring nodes to form a single vector, (ii) applying a linear transformation $\boldsymbol{\Omega}_0$ to the aggregated nodes, (iii) applying the same linear transformation $\boldsymbol{\Omega}_0$ to the original node, (iv) adding these together with a bias $\boldsymbol{\beta}_0$, and finally (v) applying a nonlinear activation function $\mathbf{a}[\bullet]$ like a ReLU. c) This process is repeated at subsequent layers (but with different parameters for each layer) until we produce the final embeddings at the end of the network.

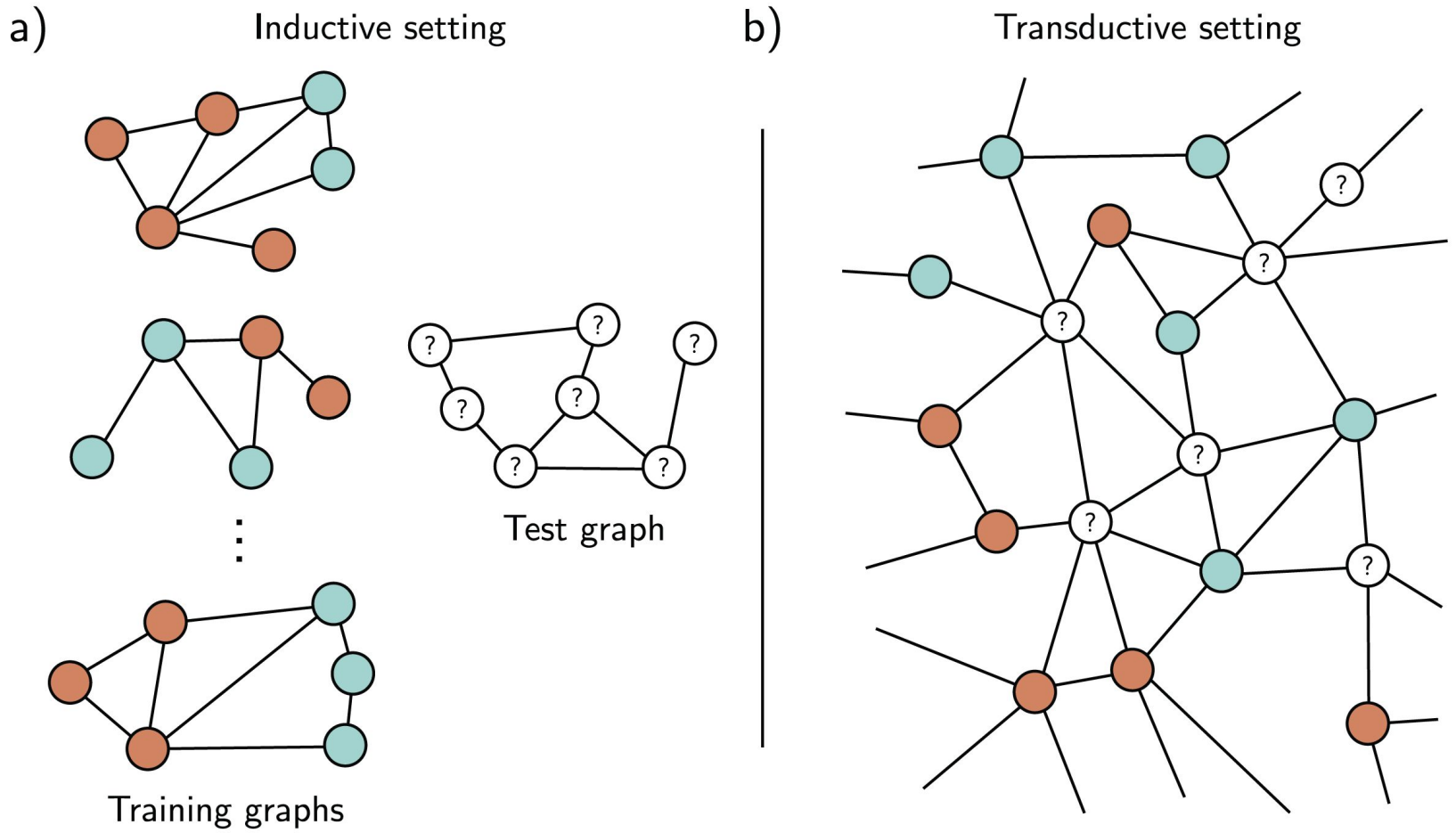

Figure 13.8 Inductive vs. transductive problems. a) Node classification task in the inductive setting. We are given a set of I training graphs, where the node labels (orange and cyan colors) are known. After training, we are given a test graph and must assign labels to each node. b) Node classification in the transductive setting. There is one large graph in which some nodes have labels (orange and cyan colors), and others are unknown. We train the model to predict the known labels correctly and then examine the predictions at the unknown nodes.

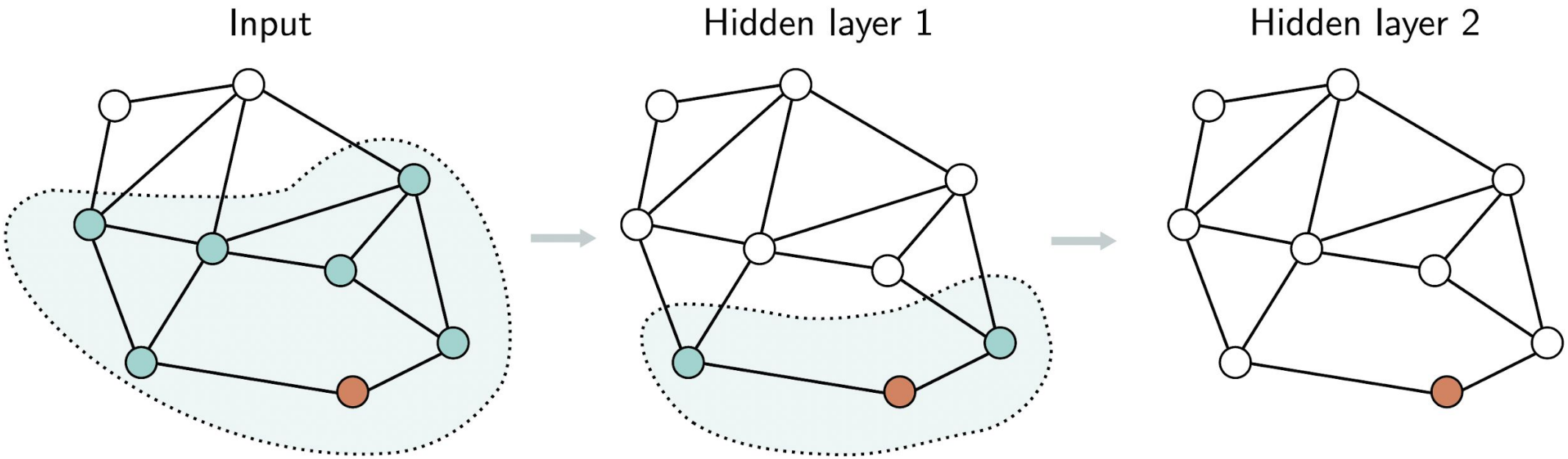

Figure 13.9 Receptive fields in graph neural networks. Consider the orange node in hidden layer two (right). This receives input from the nodes in the 1-hop neighborhood in hidden layer one (shaded region in center). These nodes in hidden layer one receive inputs from their neighbors in turn, and the orange node in layer two receives inputs from all the input nodes in the 2-hop neighborhood (shaded area on left). The region of the graph that contributes to a given node is equivalent to the notion of a receptive field in convolutional neural networks.

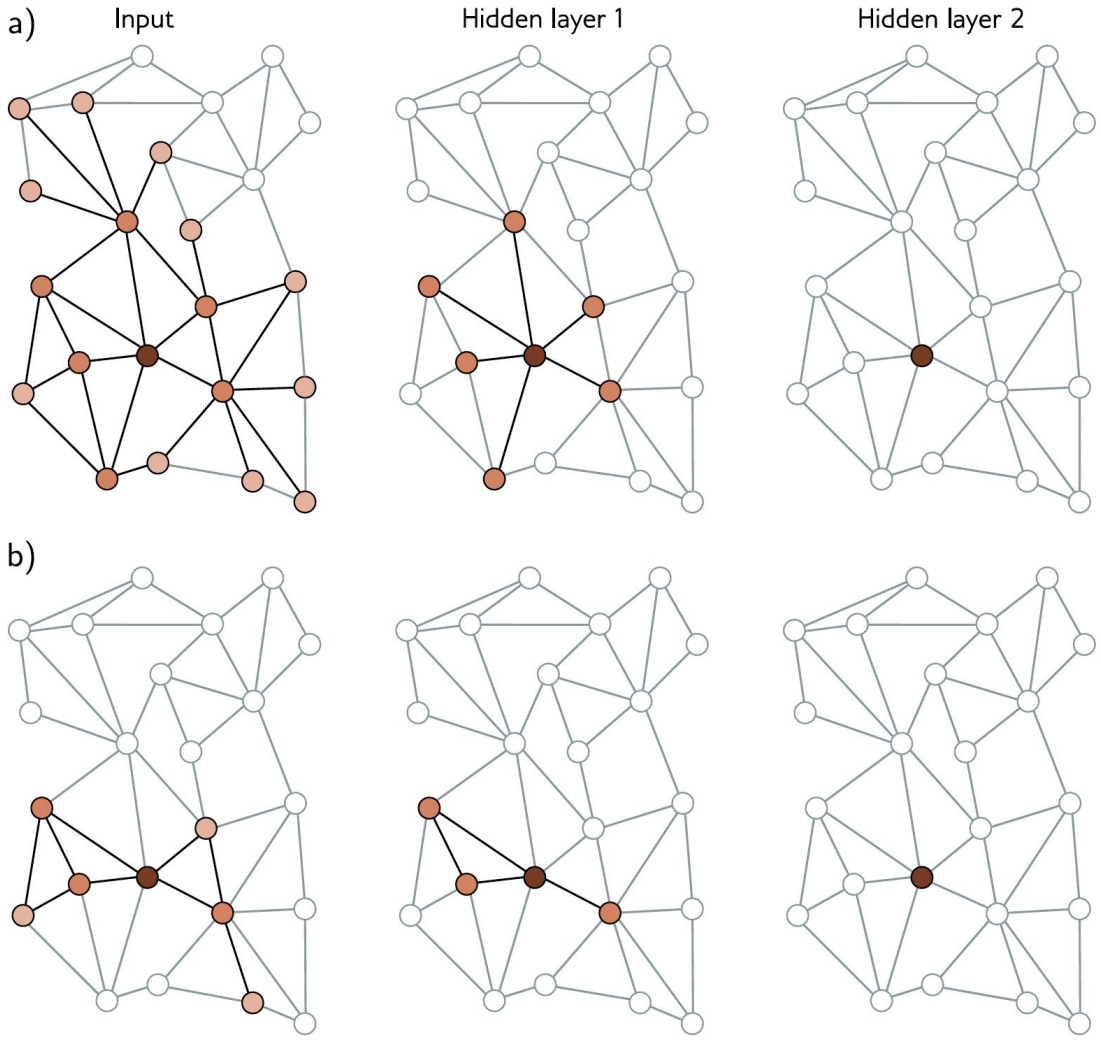

Figure 13.10 Neighborhood sampling. a) One way of forming batches on large graphs is to choose a subset of labeled nodes in the output layer (here, just one node in layer two, right) and then working back to find all of the nodes in the K-hop neighborhood (receptive field). Only this sub-graph is needed to train this batch. Unfortunately, if the graph is densely connected, this may retain a large proportion of the graph. b) One solution is neighborhood sampling. As we work back from the final layer, we select a subset of neighbors (here, three) in the layer before and a subset of the neighbors of these in the layer before that. This restricts the size of the graph for training the batch. In all panels, the brightness represents the distance from the original node.

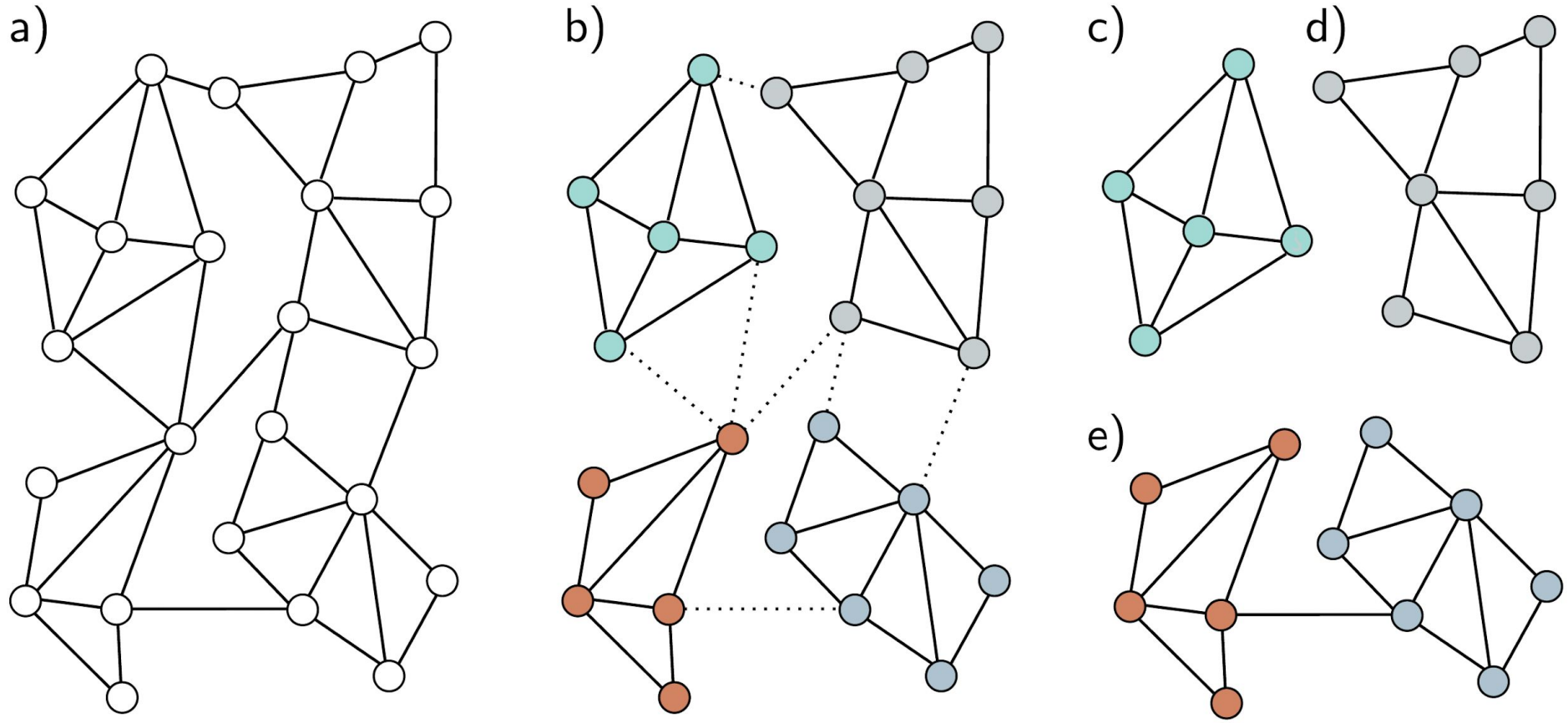

Figure 13.11 Graph partitioning. a) Input graph. b) The input graph is partitioned into smaller subgraphs using a principled method that removes the fewest edges. c-d) We can now use these subgraphs as batches to train in a transductive setting, so here, there are four possible batches. e) Alternatively, we can use combinations of the subgraphs as batches, reinstating the edges between them. If we use pairs of subgraphs, there would be six possible batches here.

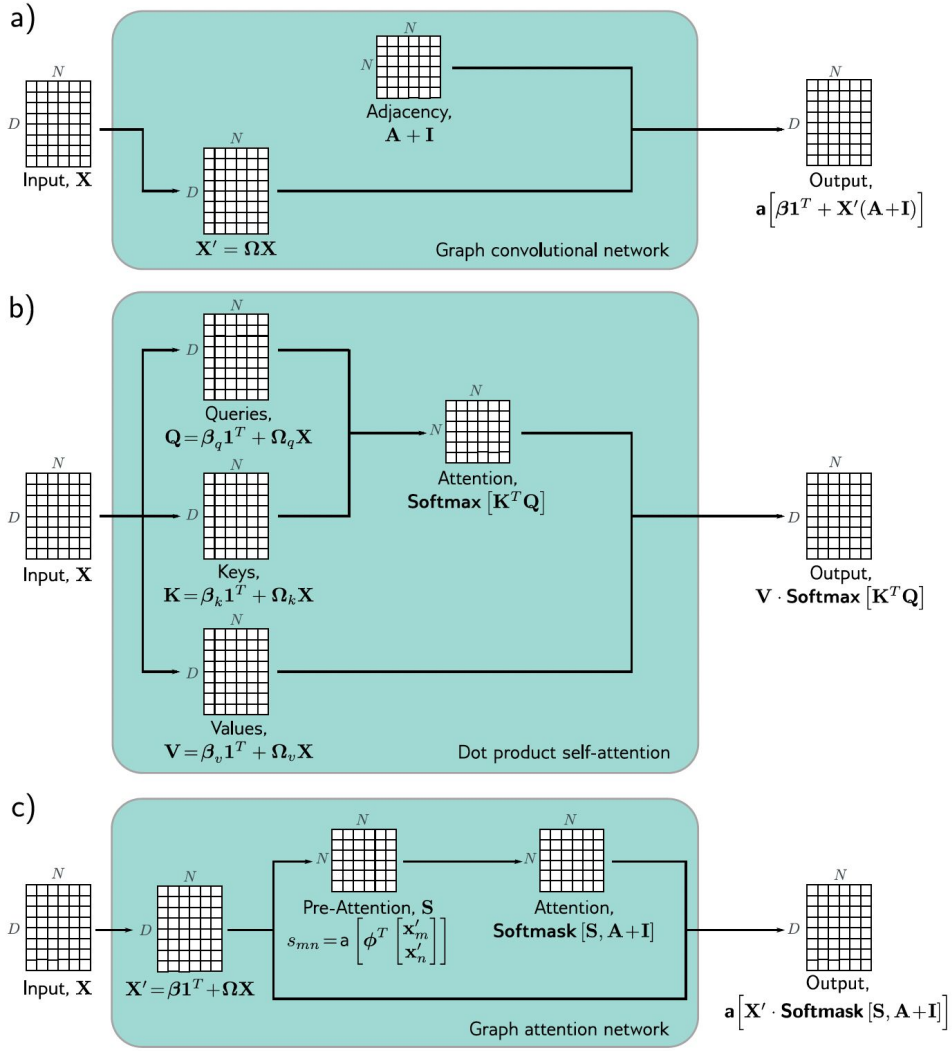

Figure 13.12 Comparison of graph convolutional network, dot product attention, and graph attention network. In each case, the mechanism maps N embeddings of size D stored in a $D \times N$ matrix $\mathbf{X}$ to an output of the same size. a) The graph convolutional network applies a linear transformation $\mathbf{X}' = \boldsymbol{\Omega}\mathbf{X}$ to the data matrix. It then computes a weighted sum of the transformed data, where the weighting is based on the adjacency matrix. A bias $\boldsymbol{\beta}$ is added, and the result is passed through an activation function. b) The outputs of the self-attention mechanism are also weighted sums of the transformed inputs, but this time the weights depend on the data itself via the attention matrix. c) The graph attention network combines both of these mechanisms; the weights are both computed from the data and based on the adjacency matrix.

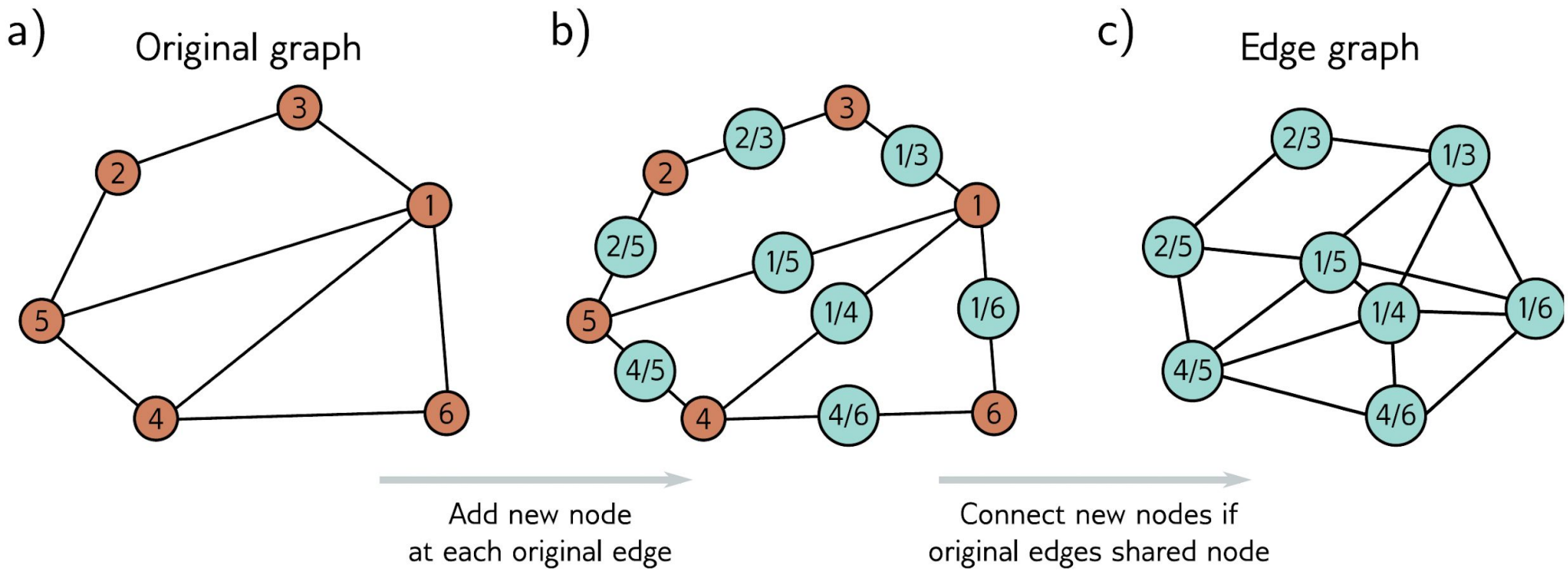

Figure 13.13 Edge graph. a) Graph with six nodes. b) To create the edge graph, we assign one node for each original edge (cyan circles), and c) connect the new nodes if the edges they represent connect to the same node in the original graph.

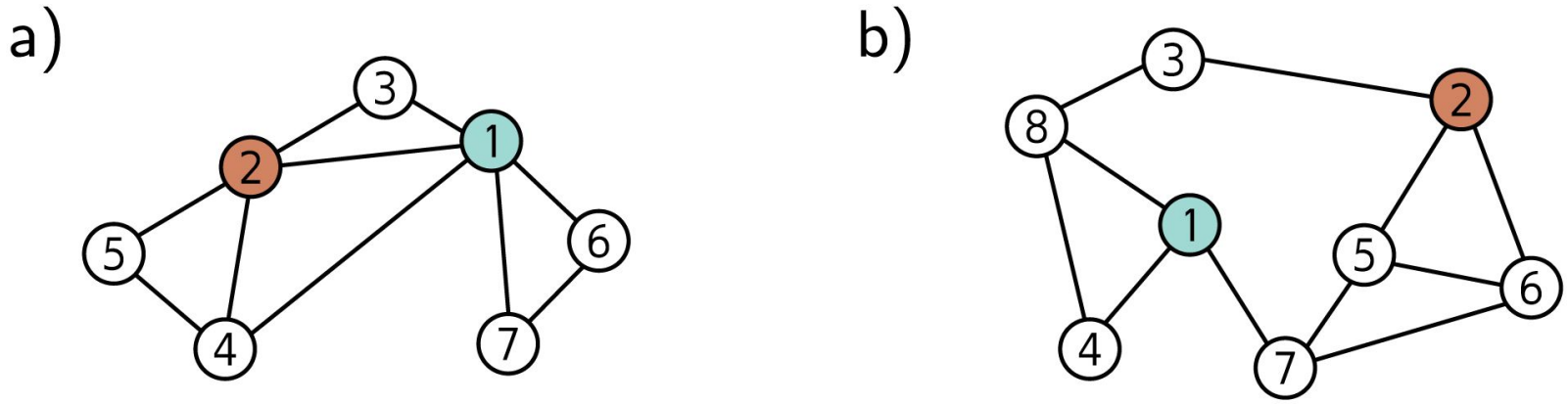

Figure 13.14 Graphs for problems 13.1, 13.3, and 13.8.

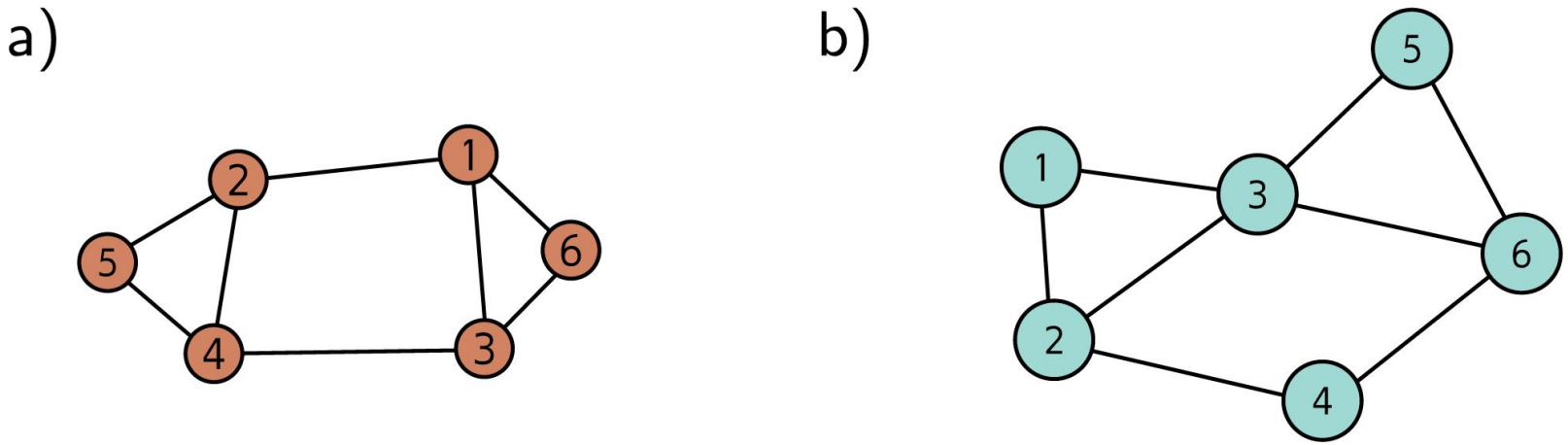

Figure 13.15 Graphs for problems 13.11–13.13.