13. Networks

Last time

- Neurons as classifiers
- Neural networks
- Stochastic gradient descent

Goals for today

- Connectivity: Networks / graphs
- Shortest path
- Random walks

Modelling connectivity

- Real-world systems often depend on **connectivity**:
    - Travel: Massachusetts road network; world airports
    - Social network: Alumni of MIT and their friendships (or not)
    - Network of interacting genes turning one another on and off
- Model as **networks** / **graphs**

Networks generalize themes of course

- Dynamics of genetic networks
- Contact network: trace people / animals using GPS as they move
- How does an epidemic spread via plane travel
- Random walk on network: effects of altered gene
- How long until two of them meet?

What is a network?

- A **network** is the pair (V, E)
- V is set of **vertices** / **nodes** $i = 1, \dots, N$
- E is set of edges joining them, e.g. edge $(3, 5)$ joins nodes 3 and 5
- Tells us how nodes are connected
- **Directed** (one-way arrows) or **undirected** (arrows go both ways)
- Edges may have **weights**, e.g. distance between nodes

What questions would we like to answer?

- Where can I move from here? (Where am I connected to?)
- Where can I move *from* to get here *from*? (Not necessarily the same)
- What is **degree** = number of neighbours
- Am I connected to a given node?
- What is **distance** from given node? $= \infty$ if not connected

How can we represent a network?

- How encode a network in the computer?
- What are possible representations?
- I.e. possible data structures

Possible solutions

- Vector of edges (i, j)
- Adjacency matrix: $A_{ij} = 1$ if i and j are connected, 0 otherwise
- Vector of neighbours of each node
- Which is better?
- Depends on how **sparse** the network / matrix is

Examples of networks:

- Completely connected
- Square / cubic grid
- **Erdos–Renyi**: N nodes, each connected to k neighbours at random

Networks in Julia

- Simple implementation:
- `Node` and `Network` types
- `add_vertex!` and `add_edge!` methods
- ≥ 2 Julia packages: `LightGraphs.jl` + `MatrixNetworks.jl`

Small-world networks

- Watts–Strogatz 1998
- Start from ring with N nodes, each connected to k neighbours
- Rewire edges at random with probability p, avoiding duplicate edges
- What is effect on path length?

Calculating shortest paths

- Several algorithms to find shortest paths
- Shortest path from single node: Dijkstra algorithm
- All pairs of shortest path lengths: Floyd–Warshall algorithm

Floyd–Warshall

- Suppose vertices are $(1, \dots, n)$
- Consider pair of vertices i, j
- Look at paths $i \to j$ with intermediate vertices in $(1, \dots, k)$
- Call p a path of minimal length / weight
- Relate to those using vertices $(1, \dots, k-1)$ only

- If k is not intermediate vertex of p then p is in set of those using only $(1, \ldots, k)$
- If k is intermediate, write p as $i \to k \to j$.
- Subpaths are shortest paths

- Call d_{ij}^{k} the distance of a shortest path from i to j with intermediates in $(1, \dots, k)$.
- $d_{ij}^{0} = w_{ij}$, weight / adjacency matrix
- $d_{ij}^{k} = \text{min}(d_{ij}^{k-1},\, d_{ik}^{k-1} + d_{kj}^{k-1})$

```
function floyd_warshall(W)
    n = size(W, 1)
    D = deepcopy(W)
    D[D .== 0] .= 1000
    # D[D .== 1] .= 0

    Ds = [deepcopy(D)]

    for k = 1:n
        new_D = zeros(size(D))
        for i in 1:n, j in 1:n
            if i == j
                new_D[i, j] = 0
                continue
            end
            new_D[i, j] = min(D[i, j], D[i, k] + D[k, j])
        end
```

Small-world property

- Distance between 2 arbitrary nodes is “small”:
- How fast does it grow with
- Cf. conversation at a party with somebody you don’t know – “six degrees of separation”

Preferential attachment:

- Barabási–Albert 1999 (also previously): **Scale-free** networks
- Model for structure of World Wide Web / internet
- Suppose node i has degree (number of neighbours) d_i
- At each step, choose a node i with probability $\propto d_i$
- Attach 1 new node there
- How choose nodes with probability in this way?

Implementation

- Choose random integer in range between 1 and $\sum_i d_i$
- Create *cumulative distribution* vector:
- $c_j := \sum_{i=1}^{j} d_i$
- `cumsum` in Julia
- Search in sorted vector c_j using bisection search
- `searchsorted` in Julia

Code

~~~
```julia
function proportional_choice(d::Vector{Int})
    c = cumsum(d)
    r = rand(1:c[end])

    return searchsorted(c, r)[1]
end
```
~~~

Random walk on a network

- If at node i, choose one neighbour uniformly and jump there
- Where does the walker spend more time?

Review

- Networks / graphs
- Data structures for implementation
- Models for social networks: small-world and scale-free