COGS109: Lecture 15

Training, associative memory

July 27, 2023

Modeling and Data Analysis

Summer Session 1, 2023

C. Alex Simpkins Jr., Ph.D.

RDPRobotics LLC | Dept. of CogSci, UCSD

Announcements

- New cape replacements logistics
- Assignments remaining A2, A3, D5, D6, D7, Q3, Q4, project
- Project checkpoint 2 EDA
- Project meetings to check in

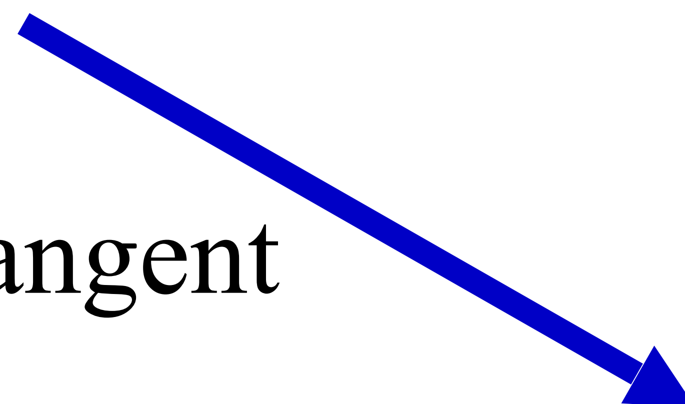
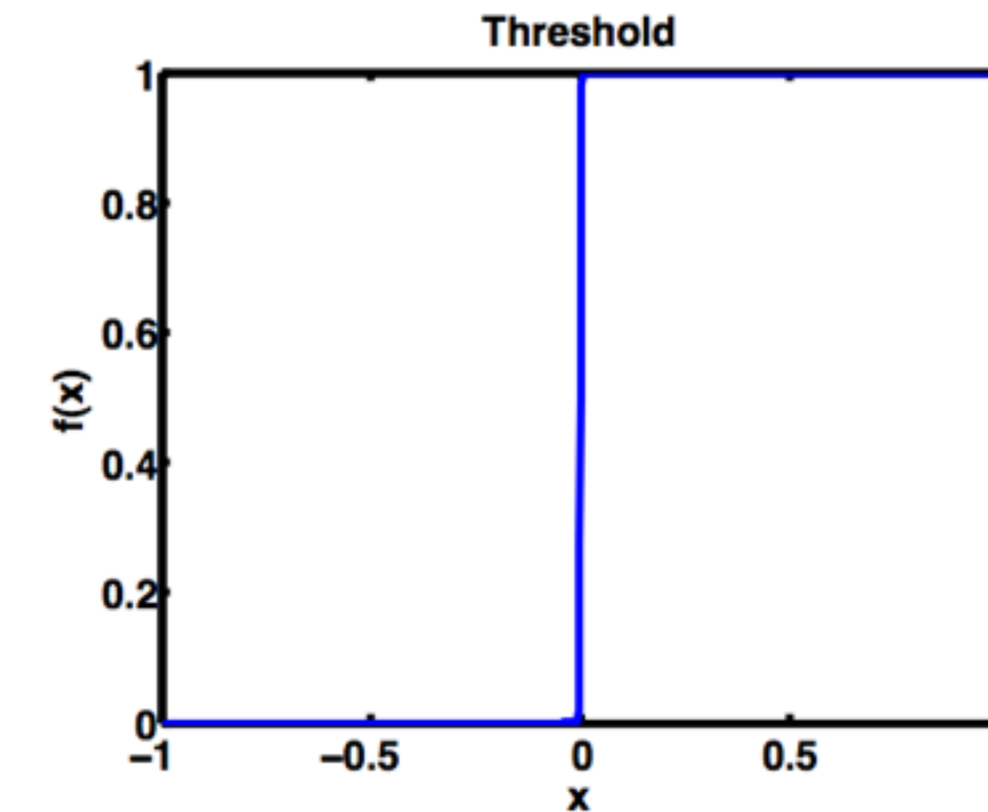
Outline for today

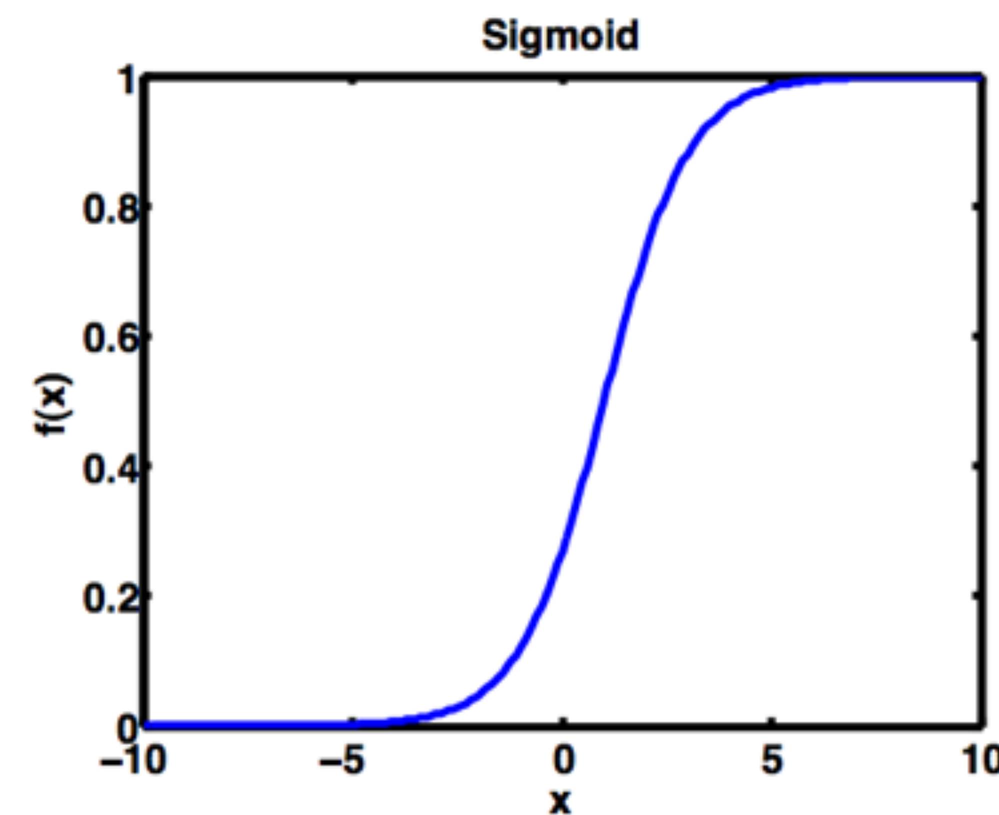
- Announcements
- Instead of a threshold, we can consider other activation functions
  - Threshold, sigmoid, linear, etc
  - Fitting arbitrary functions
- Multilayer neural networks
  - Some of the typical network topologies

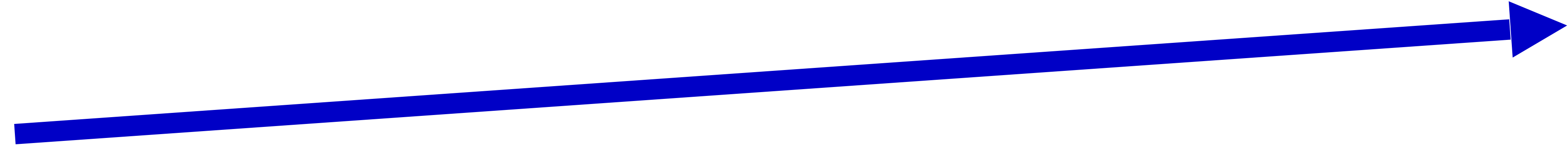
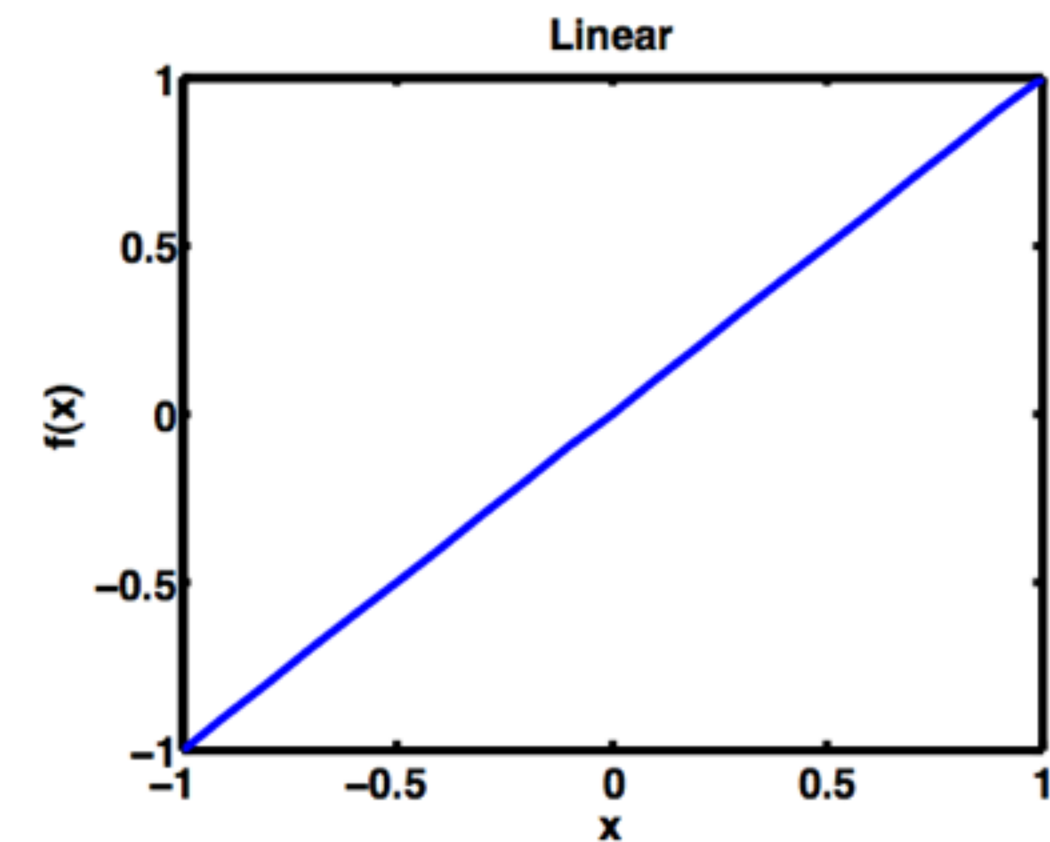
- Methods of training networks
  - **What is Supervised learning**
  - **What is Unsupervised learning**
  - **What is Reinforcement learning**
- Matlab neural network toolbox demos
- Potential issues with training networks
  - **Overfitting and generalization**
- Methods of dealing with these issues

Other activation function concepts

- Threshold
- Sigmoid
- Gaussian
- Hyperbolic tangent
- Sine
- Unit sum
- Square root
- Logistic
- Softmax
- Linear
- Many others



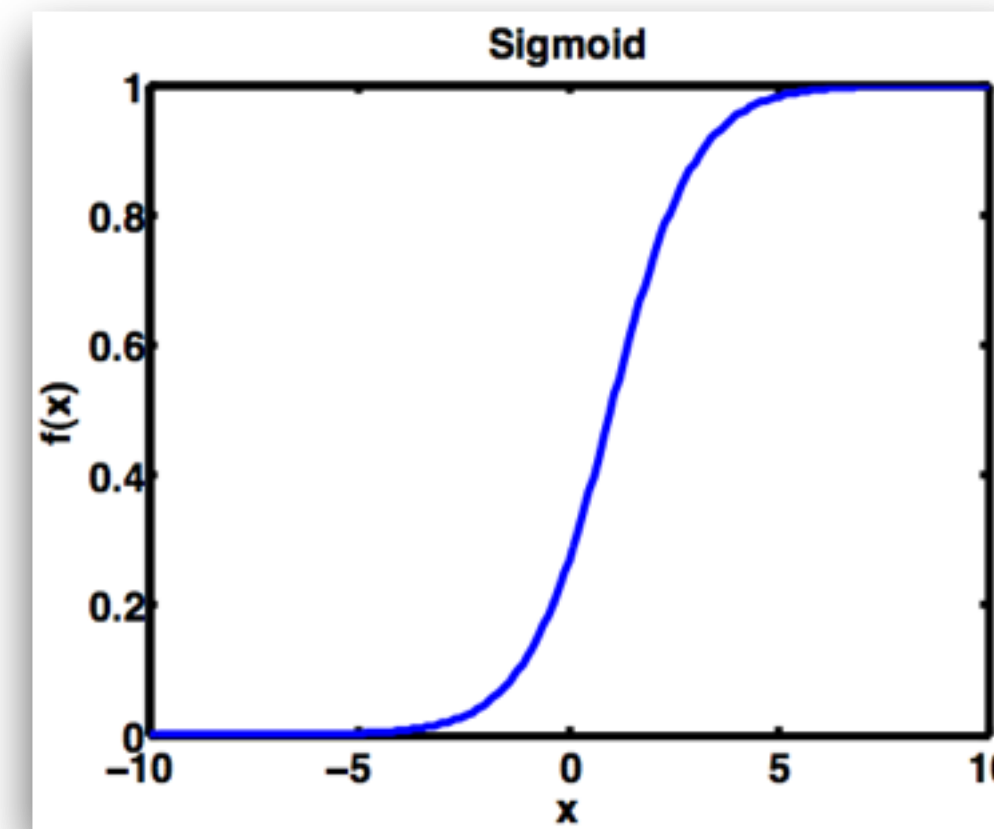
So you can have any shape activation function

- Not just threshold
- Allows you to create real-valued outputs

$$y = f\left(\sum_{i=0}^{n} w_i x_i + b\right)$$

Sigmoid

- Equation
  - **Mentioned last time**
  - **Matlab - SIGMF()**
  - **Vary parameters b and c to control the steepness of the transition from 0-1**
  - **Saturates to 0 as x->-inf, and 1 as x->+inf**
- Networks of neurons with real-valued inputs and sigmoid activation functions can be used to approximate mathematical functions
  - **Any continuous real-valued function can be approximated to arbitrary accuracy with a feedforward network of at least one hidden layer**
  - **Matlab demo -> nnd11fa**

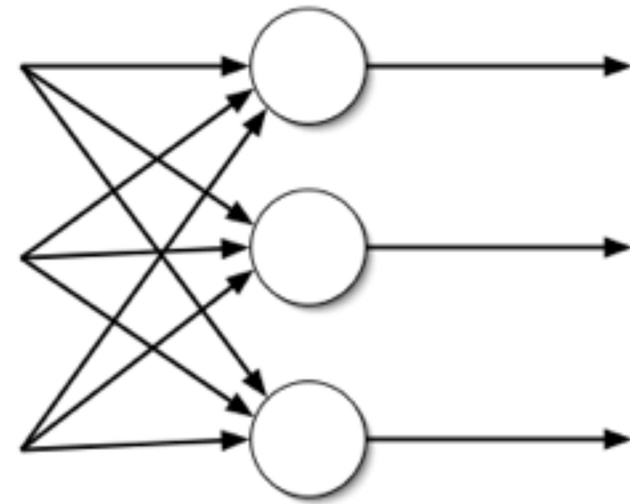


$$y = \frac{1}{1 + e^{-(bx-c)}}$$

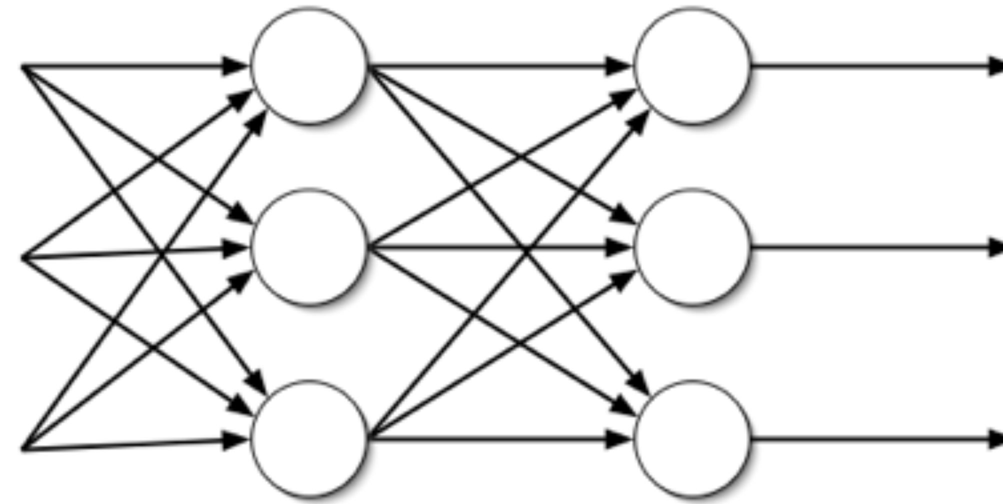
```
def sig(x):
 return 1/(1 + np.exp(-x))
```

Some typical network topologies

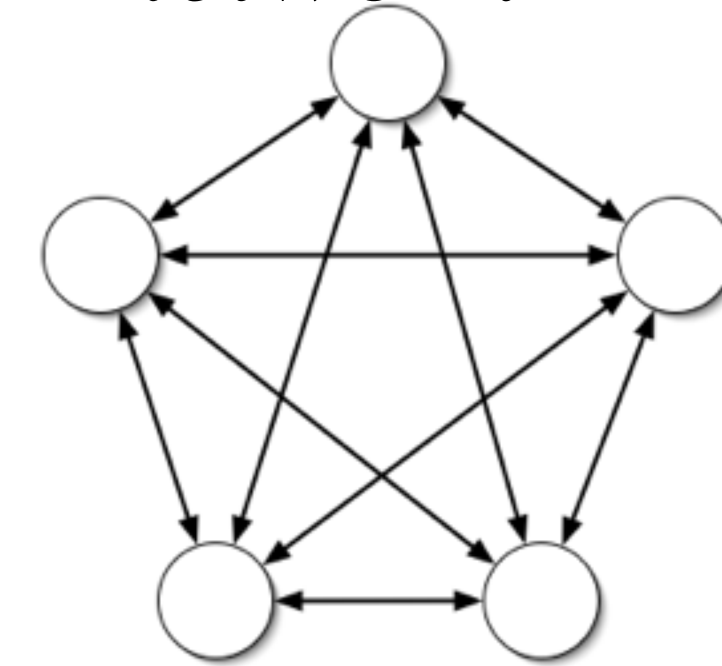
Single layer perceptron



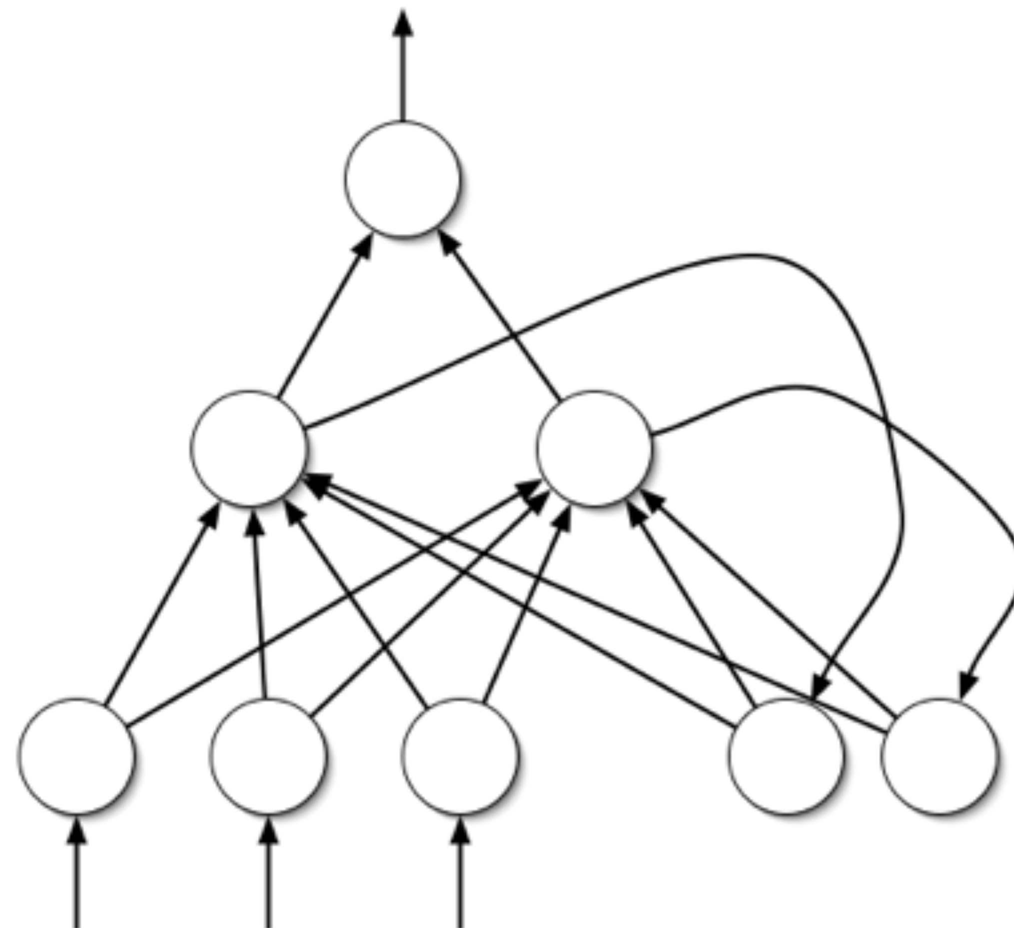
Multi-layer perceptron



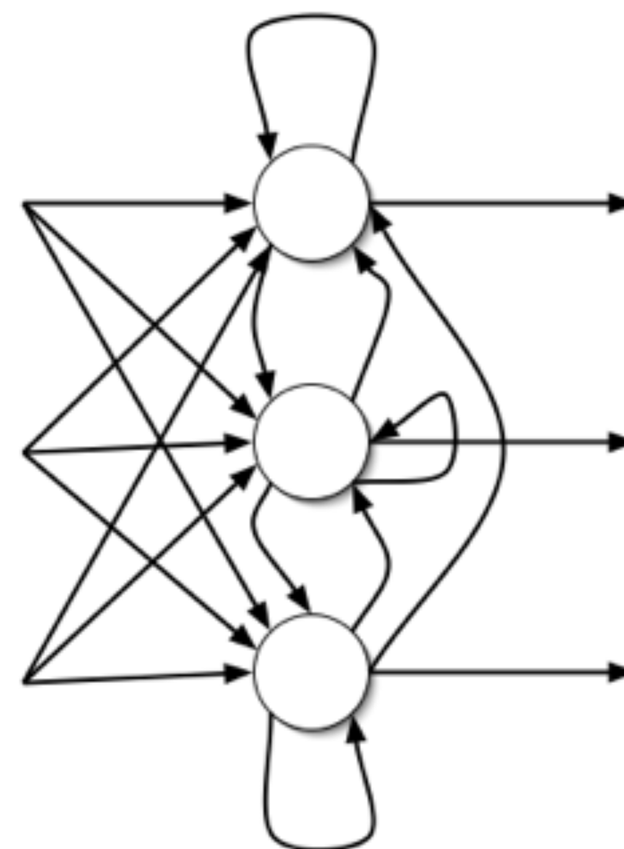
Hopfield network



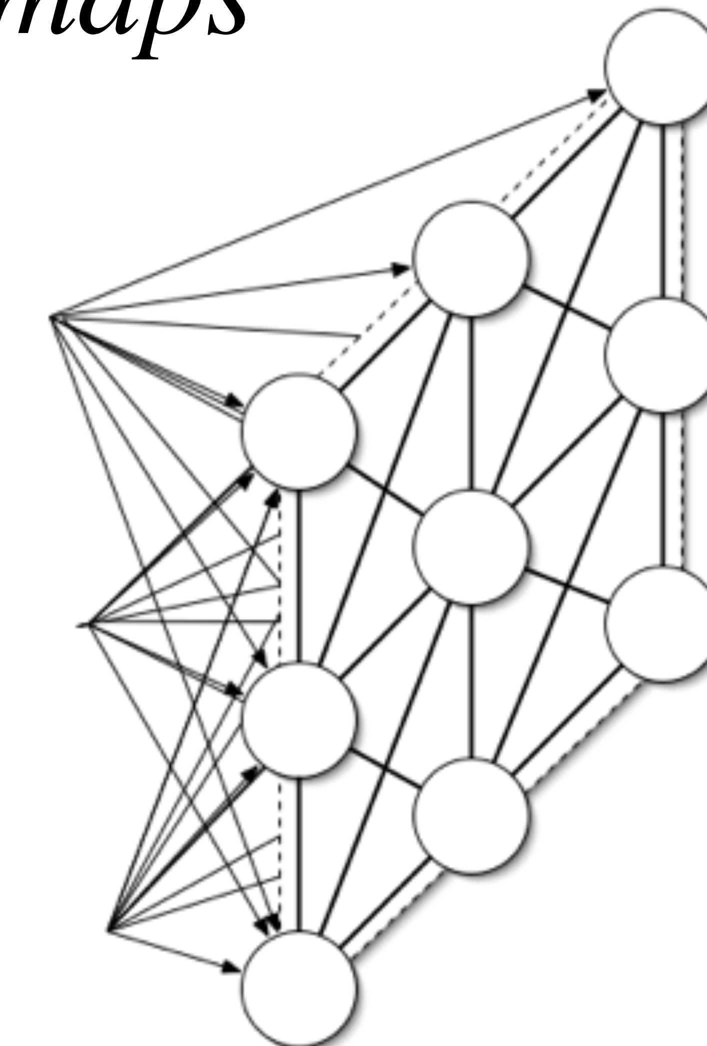
Elman recurrent network



Competitive networks



Self-organizing maps



So now we have these fancy networks,
how can we get them to ‘learn?’

- Generally boils down to three learning strategies
  - **Supervised learning**
  - **Unsupervised learning**
  - **Reinforcement learning**
- Many methods with variants, but basically all fall under these above categories

- *Method of learning whereby an error value is generated from the actual response of the network and the desired response. Following that, the weights are then modified such that the error is gradually reduced*
- **Training set** - A set of known input/output pairs is presented to the network in order to appropriately adjust the weights to produce the desired output given a certain input
- We already saw one example in the perceptron learning algorithm
- We will discuss backpropagation today

- There is still an input/output relationship but no feedback is provided indicating whether network's associations are correct or not
- The network must discover by itself similarities in the patterns of the data
  - **Self-organizing networks - networks that possess the ability to to infer patterns from input-only data**

Reinforcement learning

- Input/output data and a teaching signal
  - **The teaching signal is not a measure of the error, rather an indication of the result as ‘right’ or ‘wrong’ direction**

Neural Network Demos in matlab

- In matlab (you need the Neural Network Toolbox)
  - **nnd2n1 One-input neuron demonstration.**
  - **nnd2n2 Two-input neuron demonstration.**
  - **nnd4db Decision boundaries demonstration.**
  - **nnd4pr Perceptron rule demonstration.**
  - **nnd9sdq Steepest descent for quadratic function demonstration.**
  - **nnd11nf Network function demonstration.**
  - **nnd11bc Backpropagation calculation demonstration**
  - **nnd11fa Function approximation demonstration.**
  - **nnd11gn Generalization demonstration.**

Back propagation algorithms

- General algorithm
  - **Present inputs**
  - **Propagate network responses forward**
  - **Compute the error between output and desired output**
  - **Back-propagate deltas**
  - **Update weights**
  - **Repeat for next pattern**
- Matlab demo **- nnd11bc**
- **Coded in python:** https://machinelearninggeek.com/backpropagation-neural-network-using-python/
- **Using scikit-learn in python**
  - https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Specifically

1. Initialize weights randomly
2. Present an input vector pattern to the network
3. Evaluate the outputs of the network by propagating signals forwards
4. For all output neurons, calculate

$$y = f\left(\sum_{i=0}^{n} w_i x_i + b\right) = \left(1 + exp(-(\sum_{i=0}^{n} w_i x_i + b))\right)^{-1}$$

   1. **d_j is desired output of neuron j and y_j is current output**

$$\delta_j = (y_j - d_j)$$

5. For all other neurons compute delta

$$\delta_j = \sum_k w_{jk} g'(x) \delta_k$$

   1. **Where delta_k is the delta_j of succeeding layer, and**

$$g'(x) = y_k(1 - y_k)$$

6. Update weights according to

$$w_{ij}(t+1) = w_j(t) - \eta y_i y_j (1 - y_j) \delta_j$$

7. Goto 2 until iterationmax or minimal error

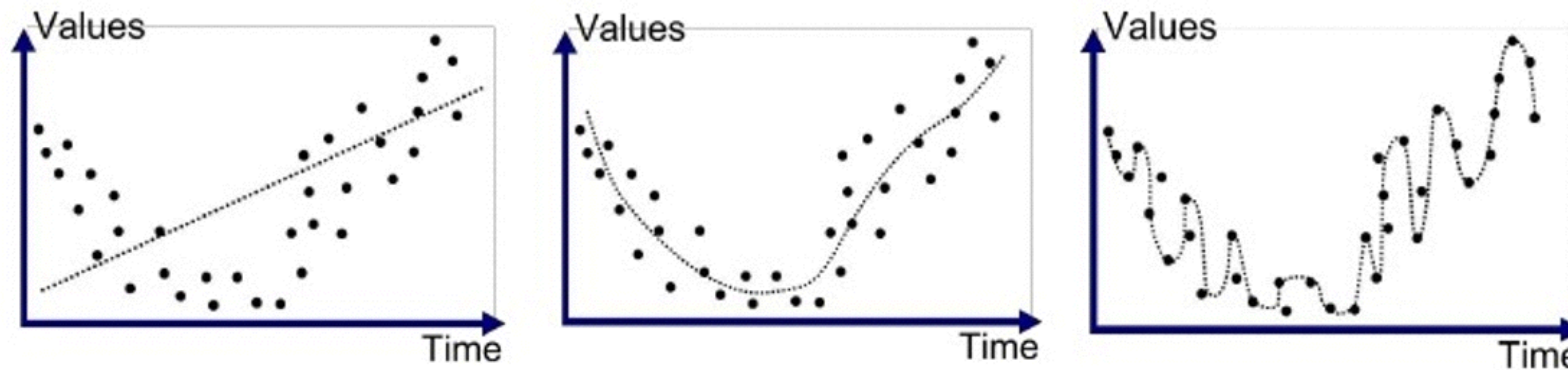
- https://scikit-learn.org/stable/modules/neural_networks_supervised.html

- Simple function fit
- Classification

- Over-fitting
- Generalization
- We want to reduce over-fitting and increase generalization of our fits

What is overfitting again?

- Happens in machine learning where we have many parameters and limited data, the algorithm begins to fit the noise
- Not truly part of the system, varies from one sample to the next



https://www.google.com/url?sa=i&url=https%3A%2F%2Fmedium.com%2Fgreyatom%2Fwhat-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76&psig=AOvVaw2RnpUIFc9mMXwi3SWPhQRR&ust=1690988870076000&source=images&cd=vfe&opi=89978449&ved=0CA0QjRxqFwoTCKDin8_eu4ADFQAAAAAdAAAAABAD

Techniques to Prevent Overfitting

- Regularization
  - **Reduction of hidden units**
    - Only fit simpler functions
  - **Weight decay**
- Early stopping
  - **Using validation sets**
- Bayesian regularization
  - **(see the MacKay Book)**
- Others
  - Random dropouts
  - etc

- *Note: Remember that overfitting is a problem when fitting many parameters to small amounts of data*
  - **Infinite data would be then no problem**
- Simplify the function you are fitting by reducing the number of network hidden layers - similar to using a lower degree polynomial to fit data
  - **Limits the capability of your network**
- But ahead of time we may not know the complexity of the function we want to fit, so how do we deal with this?
  - **Intuition - visualization of the data allows the programmer insight**
  - **Others**

Technique 2: Regularization to prevent overfitting

- ***Regularization*** - adding a penalty to the usual error function to encourage smoothness

$$E_{new} = E + \nu * \omega$$

- Here ν is the regularization parameter and ω is the smoothness penalty

$$\omega = \frac{1}{2}\sum_i w_i^2$$

- Weight decay sets
  - **Note that when you then take the partial derivative of E_{new} with respect to a weight the update rule will now include a term that is -w_i.**
  - **This will encourage the weights to decay to zero (hence the name)**
  - **Also one can use simply the absolute value - more robust to outliers L_1 vs. L_2 regularization**

Regularization types

$$\omega = \sum_i |w_i|$$

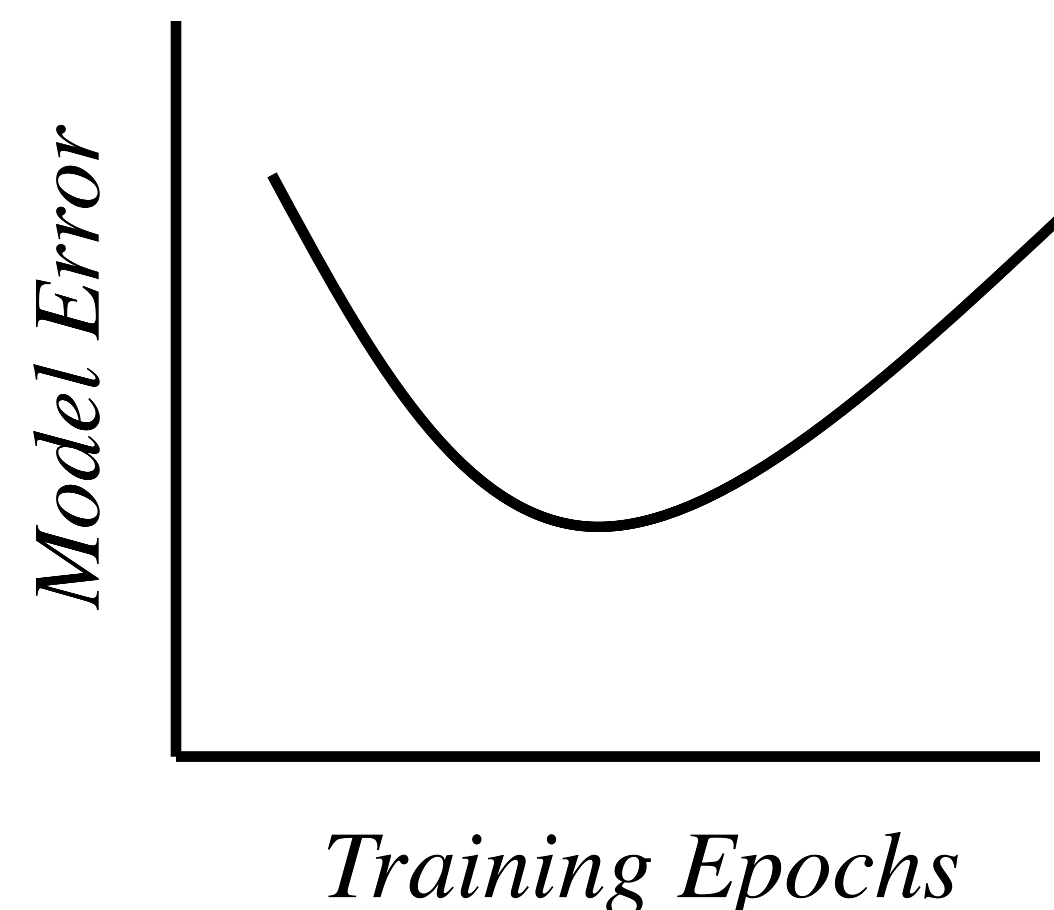
- **L_1 regularization**
  - penalizes sum of absolute value of weights
  - Model becomes simpler, more interpretable (i.e. std. vs. variance)
  - Robust to outliers

$$\omega = \frac{1}{2} \sum_i w_i^2$$

- **L_2 regularization**
  - penalizes sum of square of weights
  - Model becomes complex, able to handle more complex patterns
  - Not robust to outliers

Technique 3: Early stopping to prevent overfitting

- Start the weights very small
  - **Then the neural network starts by behaving fairly linearly**
  - **The weights gradually increase to handle nonlinearities**
- Split the data into a **validation set** and a **training set**
  - **Use the training set to adjust the weights**
  - **Use the validation set to compute model error**
  - **As the fit improves the error will decrease, when the error starts to increase again, you are fitting the noise in the training set**



Technique 4: Bayesian regularization to prevent overfitting

- The Bayesian neural network formalism of David MacKay and Radford Neal, considers neural networks not as single networks but as distributions over weights (and biases)
- The output of a trained network is thus not the result of applying one set of weights but an average over the outputs from the distribution.
- This can be computationally expensive but MacKay and Neal have developed approximations and the approach leads to automatic regularization that is very effective.
  - MacKay, Neural Computation, Vol. 4, No. 3, 1992, pp. 415 to 447
  - Foresee and Hagan, Proceedings of the International Joint Conference on Neural Networks, June, 1997

- Improvements on gradient descent
  - **Gradient descent with momentum**
  - ***Conjugate gradient***
  - **Variable learning rate**
  - **For nonquadratic functions, minimization (ie Nelder Mead, golden section line search, Brent's method, etc - See numerical methods book)**
    - Demos:
      - **nnd12sd1**
      - **nnd12sd2**
      - **nnd12mo**
      - **nnd12vl**
      - **nnd12ls**
      - **nnd12cg**

Unsupervised learning for associative memory

- Hebbian learning (Hebb 1949)
- The weights of neurons whos activities are positively correlated are increased:

$$\frac{dw_{ij}}{dt} \sim Correlation(x_i, x_j)$$

- So when stimulus m is present, the activity of neuron m increases
- Neuron n is associated with another stimulus n
- If these two stimuli co-occur in the environment, the Hebbian learning rule will increase the weights w_{nm} and w_{mn}
  - **Now when stimulus n appears later alone, the positive weight from n->m will cause neuron m to be also activated**

Associative memory with Hopfield networks

- Associative memory sample
  - **(Yellow)--(banana smell)**
- What is a binary Hopfield network?
  - **Weights are constrained to be** $w_{ij} = w_{ji}$
    - Symmetric
    - Bidirectional
    - No self connections (w_ii = 0)
  - **Activity rule**

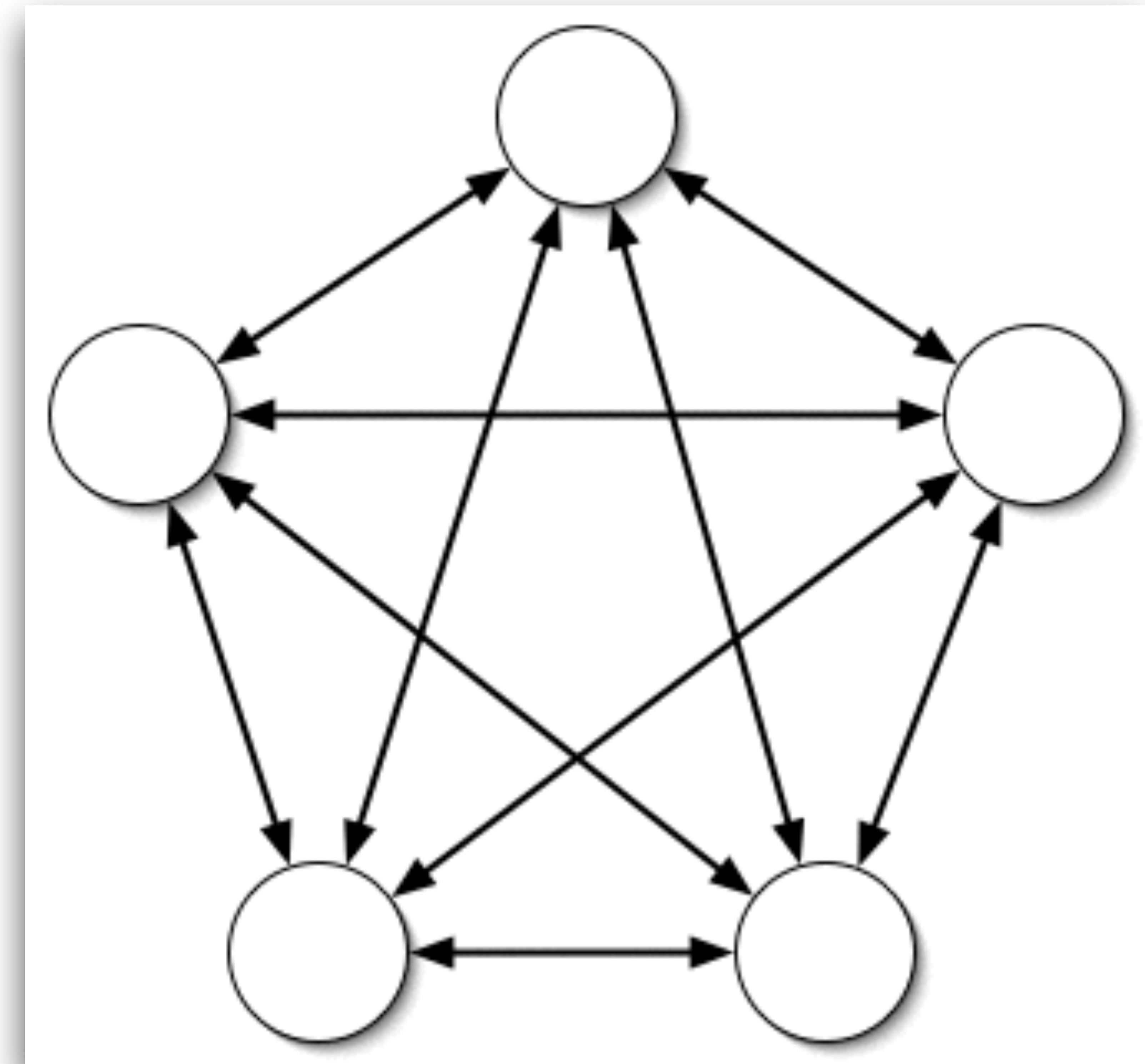
$$x(a) = \Theta(a) \equiv \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

    - We need to specify the order of updates as either
      - **Synchronous**

$$a_k = \sum_j w_{kj} x_j$$

$$x_k = \Theta(a_k)$$

      - **Asynchronous -** each neuron sequentially (either fixed or random order) computes its activation then updates its output state and weights



Associative memory with Hopfield networks

- Associative memory sample
  - **(Yellow)--(banana smell)**
- What is a binary Hopfield network?
  - **Weights are constrained to be** $w_{ij} = w_{ji}$
    - Symmetric
    - Bidirectional
    - No self connections (w_ii = 0)
  - **Activity rule**

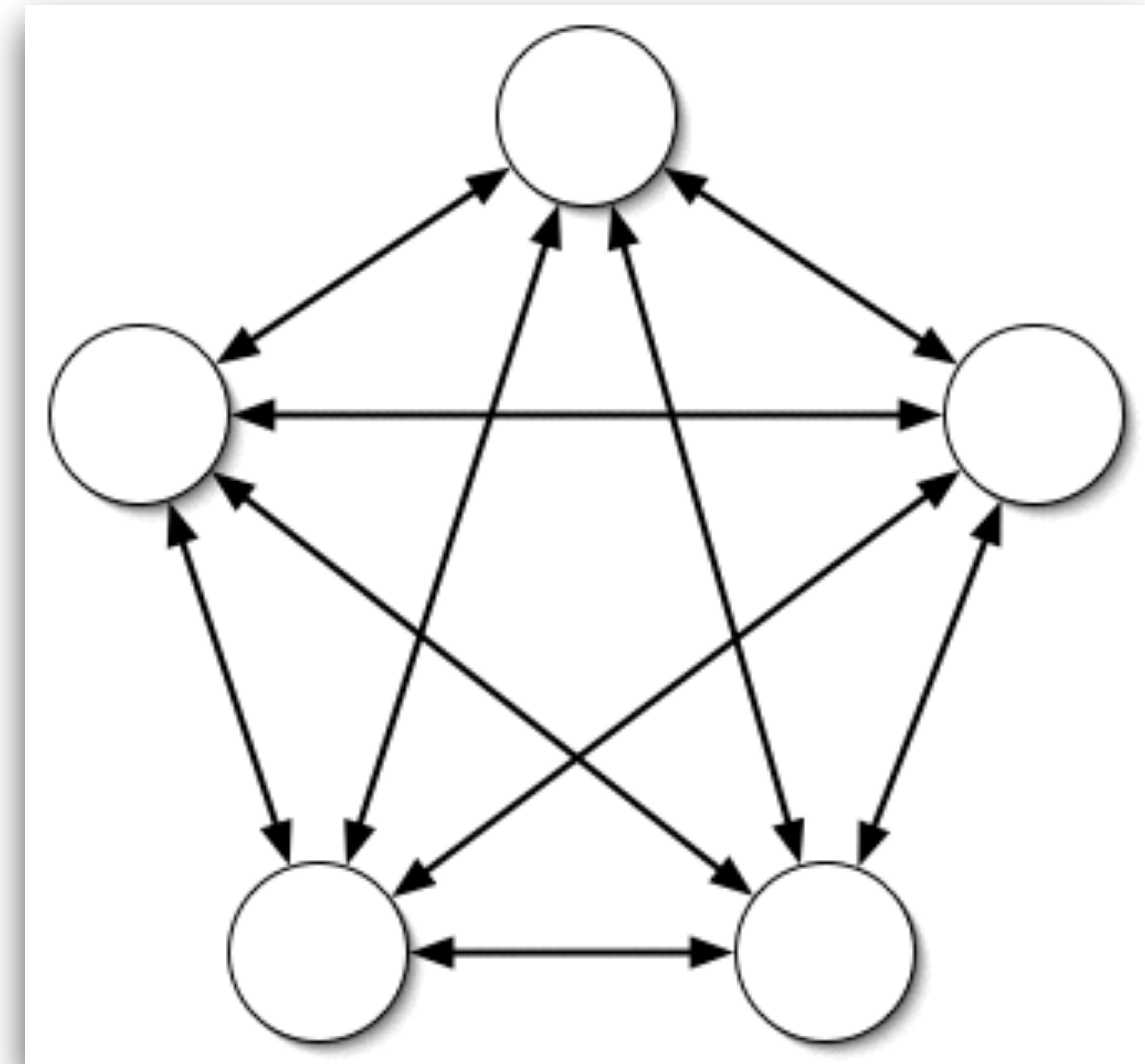
$$x(a) = \Theta(a) \equiv \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

- We need to specify the order of updates as either
  - **Synchronous**
    - *Neurons compute activations*
    - *Then update their states*

$$a_k = \sum_j w_{kj} x_j$$

$$x_k = \Theta(a_k)$$

  - **Asynchronous -** each neuron sequentially (either fixed or random order) computes its activation then updates its output state and weights



Binary Network learning rule

- Learning rule
  - **The problem - make a set of memories $\{\mathbf{x}^{(n)}\}$ stable states of the network's activity rule**
    - Each memory is a binary pattern $x_i \in \{-1, 1\}$
    - Setting the weights is done according to Hebb's rule (sum of outer products):

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)}$$

    - We may set eta to prevent a particular weight from growing with N:

$$\eta = 1/N$$

Associative memory example

Desired memories:

moscow------russia
lima----------peru
london-----england
tokyo--------japan
edinburgh-scotland
ottawa------canada
oslo--------norway
stockholm---sweden
paris-------france

- Pattern completion

moscow---:::::::: $\Longrightarrow$ moscow------russia
::::::::::--canada $\Longrightarrow$ ottawa------canada

- Error correction

otowa-------canada $\Longrightarrow$ ottawa------canada
egindurrh-sxotland $\Longrightarrow$ edinburgh-scotland

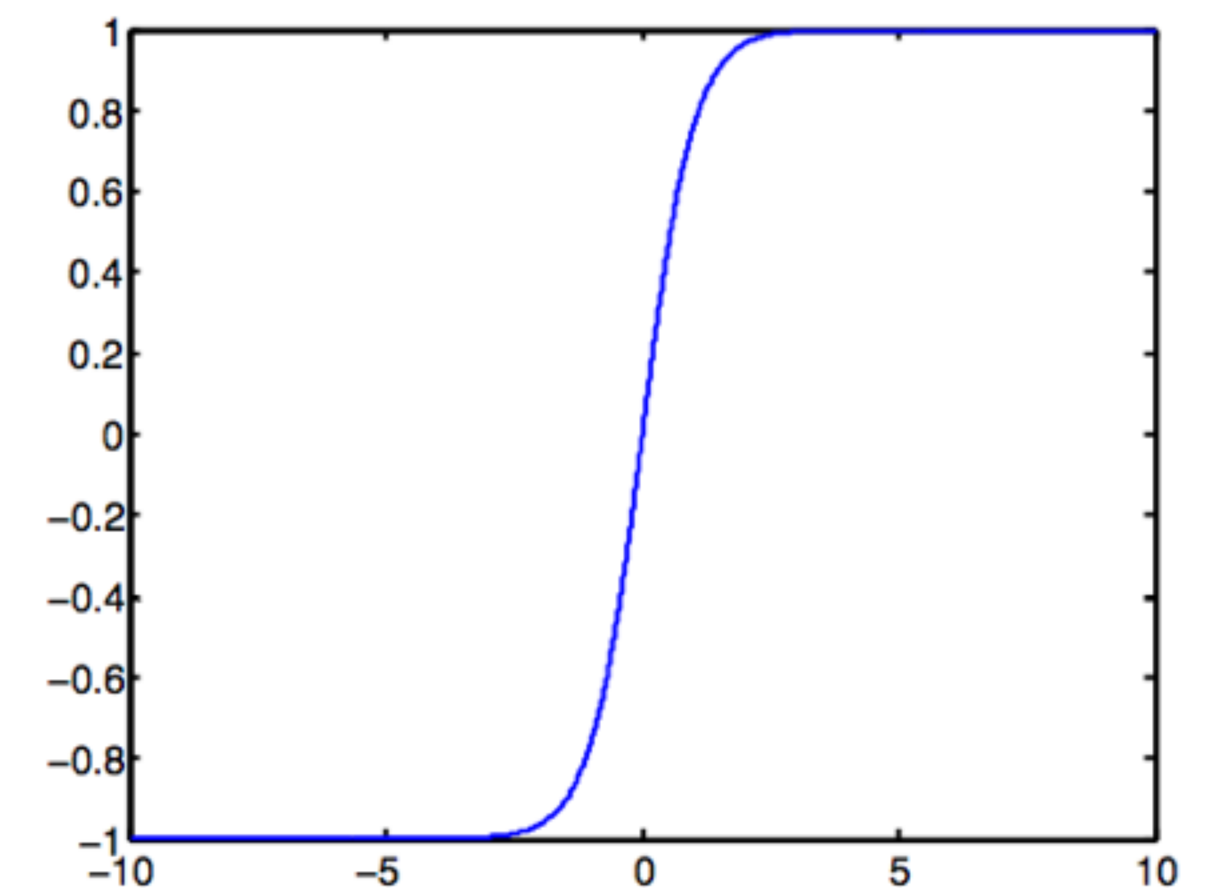
Continuous form of the Hopfield network

- Similar rules, but instead of binary states, we have continuous states from (-1,1)

$$a_i = \sum_j w_{ij} x_j$$

$$x_i = \tanh(a_i)$$

Plot of y=Tanh(x)



- Eta becomes more important

- Alternatively, we may fix η and introduce a gain $\beta \in (0, \infty)$ into the activation function

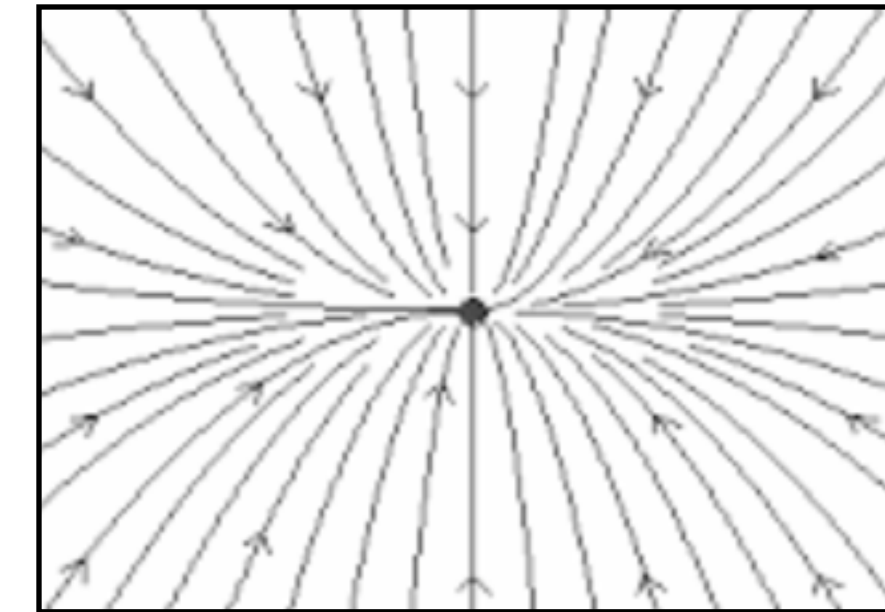
$$x_i = \tanh(\beta a_i)$$

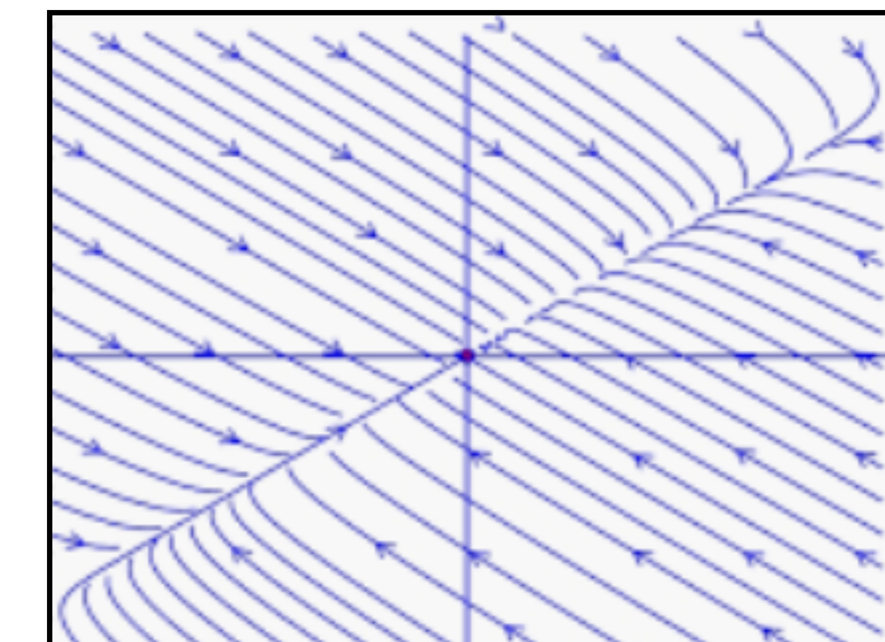
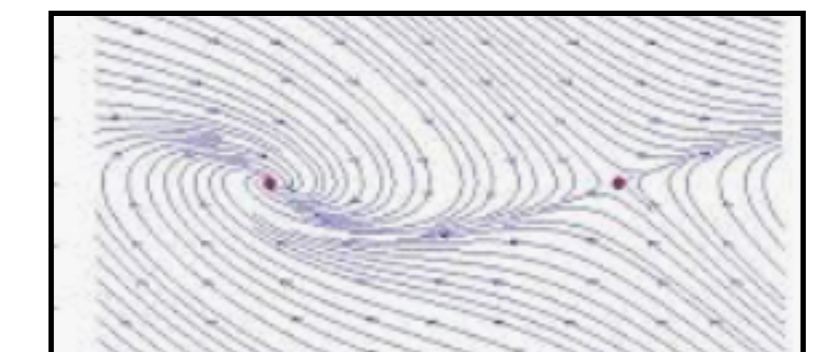
Convergence of the Hopfield network

- The hope is that the Hopfield networks we have defined will perform associative memory recall
- We hope that the activity rule of a Hopfield network can take a partial memory or a corrupted memory, then perform pattern completion or error correction to restore the original memory.
- But how can we know if this is going to happen?

Stability in nonlinear dynamics

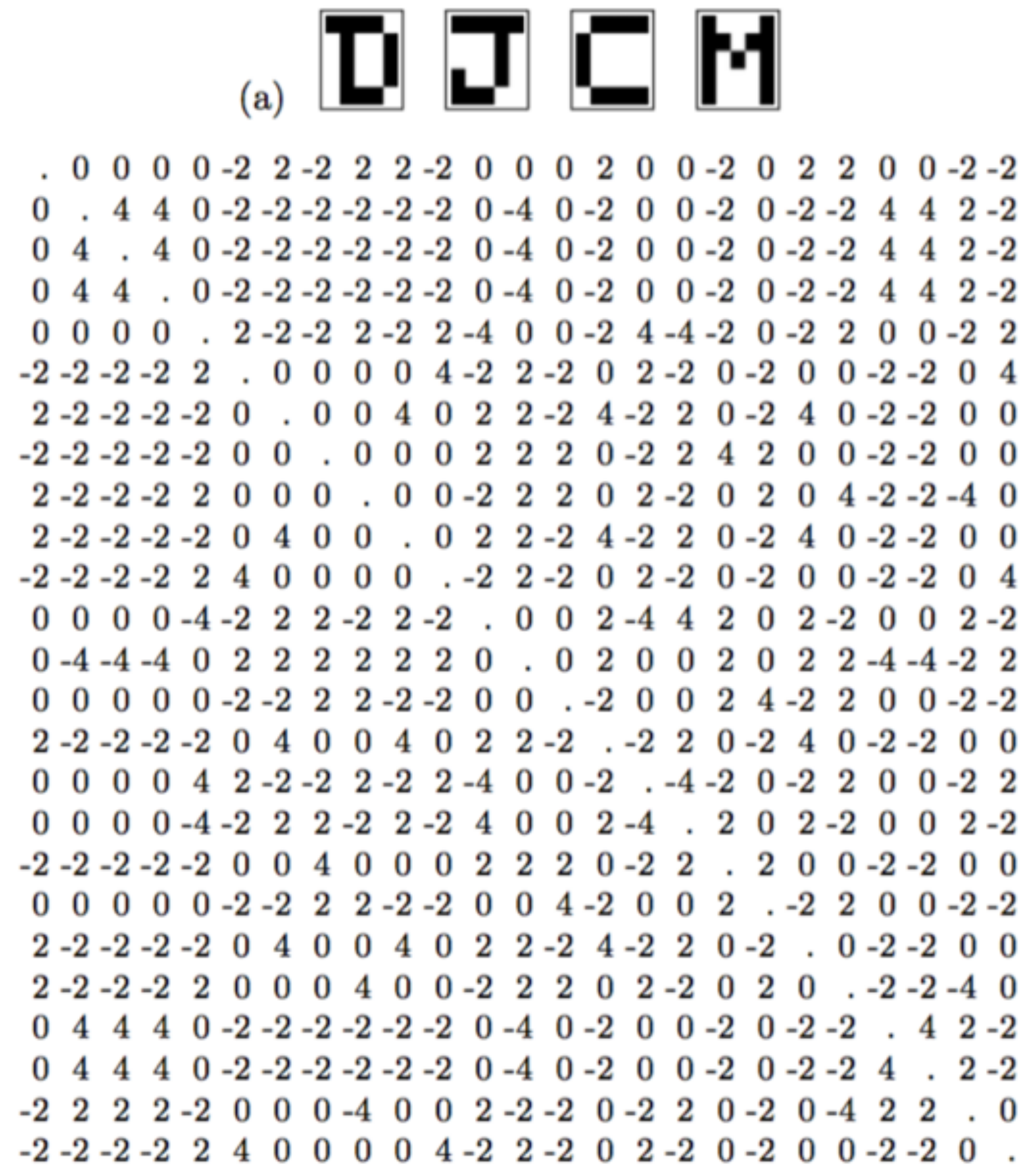
- Lyapunov functions
  - **If you can show that a lypapunov function exists for an ANN, then it's dynamics converge rather than diverge**
  - **Look up lyapunov functions for more info, there is not time to cover them here**

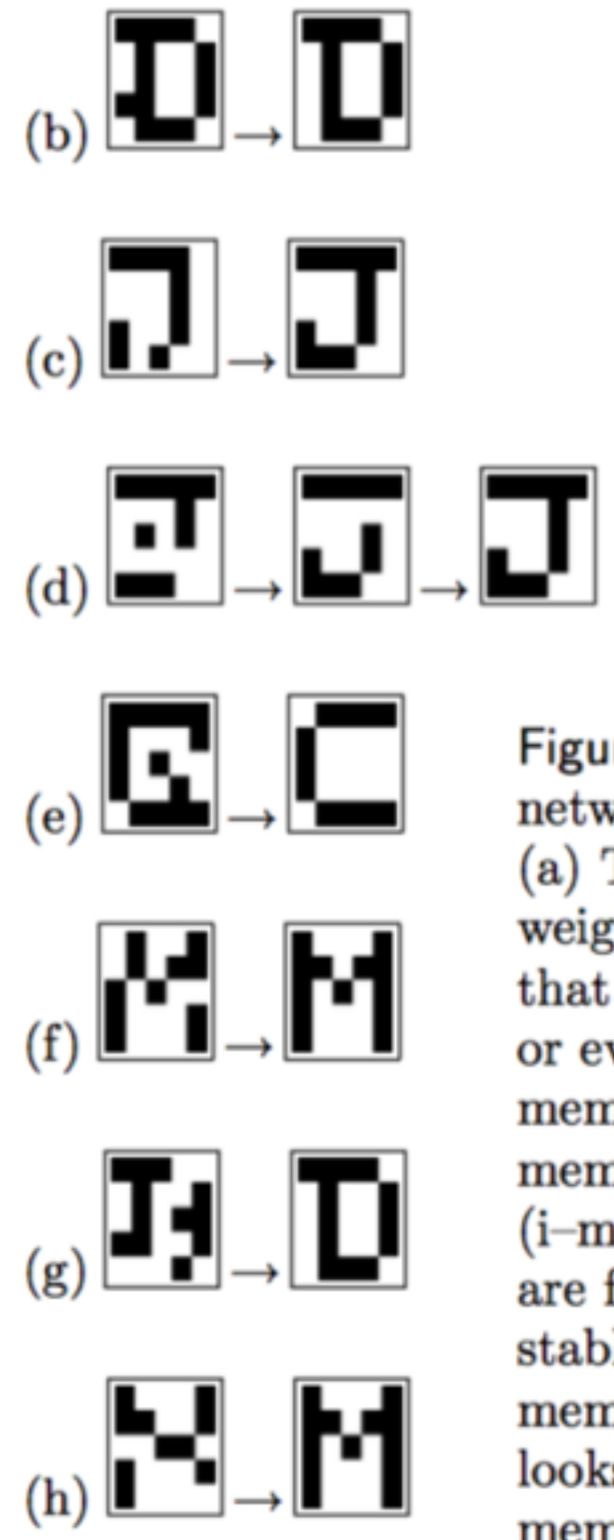


- Hopfield network’s activity rules if implemented asynchronously have a Lyapunov function that is convex
- So the dynamics will ALWAYS converge to a stable fixed point
- Depends on the fact that HN’s connections are symmetric and updates are asynchronously made
- Mackay p.508 for the proof

Introducing a 1-bit error is corrected in 1 iteration





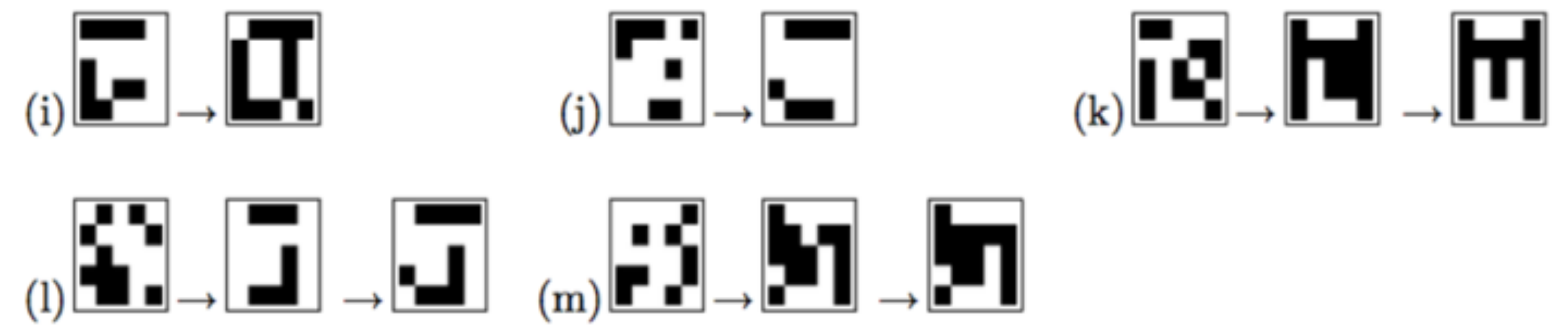
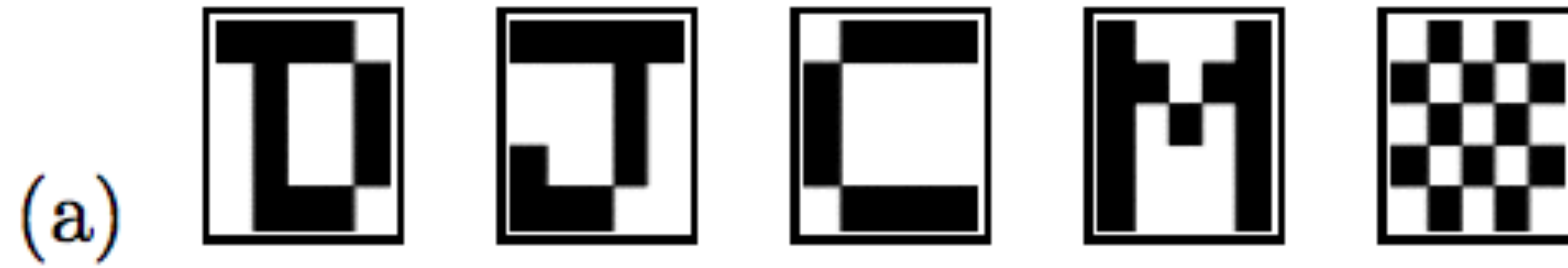
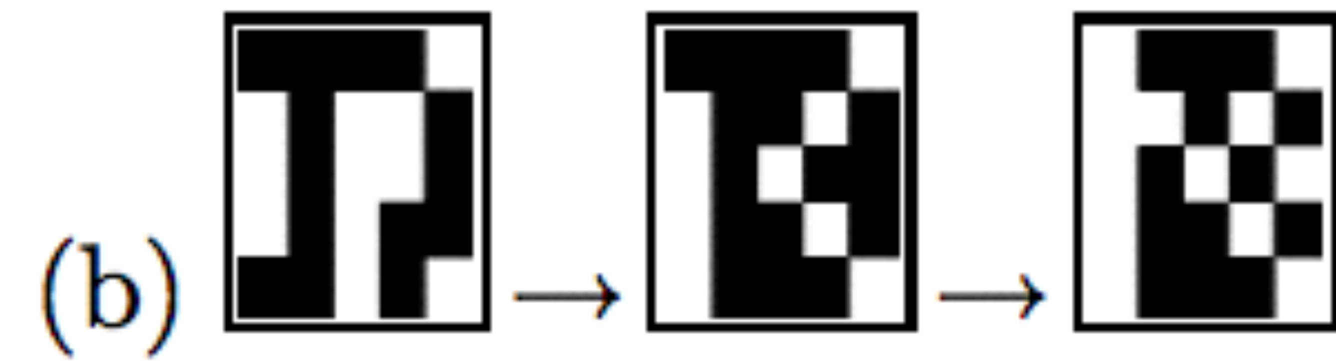


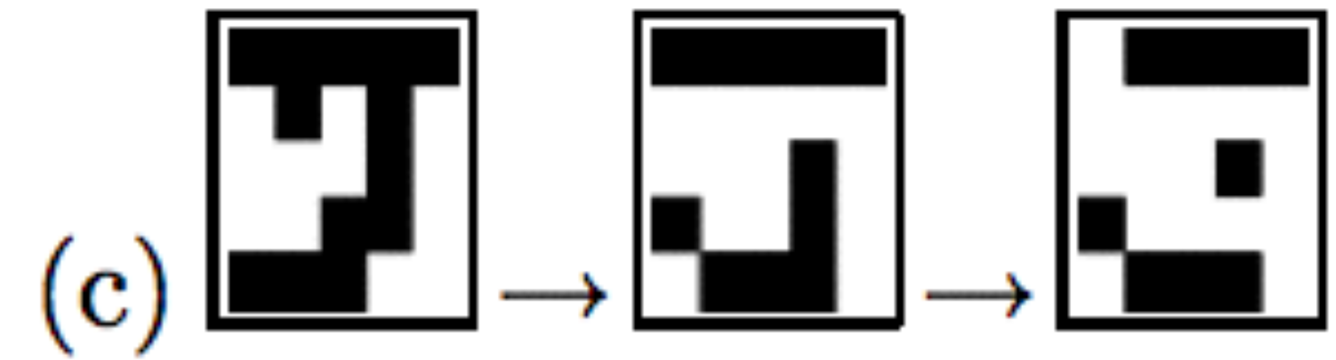
Figure 42.3. Binary Hopfield network storing four memories. (a) The four memories, and the weight matrix. (b–h) Initial states that differ by one, two, three, four, or even five bits from a desired memory are restored to that memory in one or two iterations. (i–m) Some initial conditions that are far from the memories lead to stable states other than the four memories; in (i), the stable state looks like a mixture of two memories, ‘D’ and ‘J’; stable state (j) is like a mixture of ‘J’ and ‘C’; in (k), we find a corrupted version of the ‘M’ memory (two bits distant); in (l) a corrupted version of ‘J’ (four bits distant) and in (m), a state which looks spurious until we recognize that it is the inverse of the stable state (l).

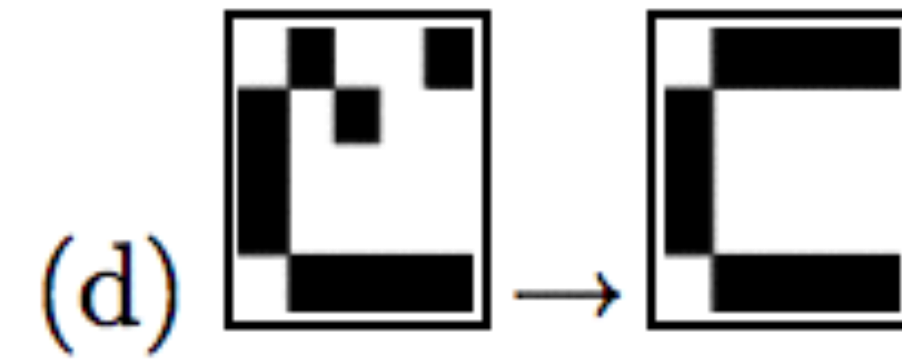
Brain damage (p. 511 in MacKay) - delete 26 weights, still converges

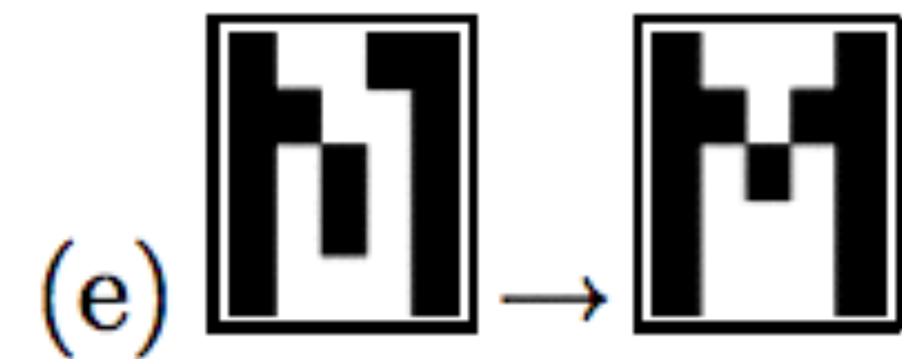


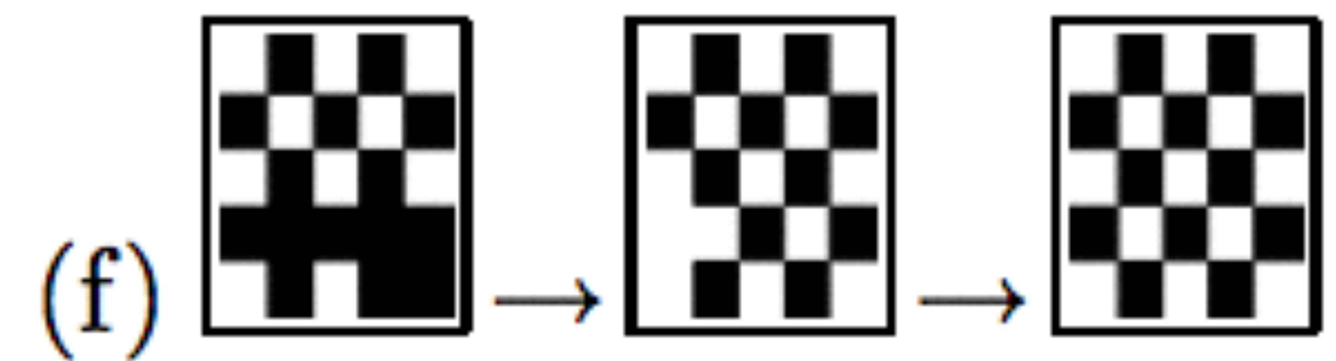

```
 .-1  1-1  1  x  x-3  3  x  x-1  1-1  x-1  1-3  x  1  3-1  1  x-1
-1  .  3  5-1-1-3-1-3-1-3  1  x  1-3  1-1-1-1-1-3  5  3  3-3
 1  3  .  3  1-3-1  x-1-3-1-1  x-1-1-1  1-3  1-3-1  3  5  1-1
-1  5  3  .-1-1-3-1-3-1-3  1-5  1-3  1-1-1-1-1-3  5  x  3-3
 1-1  1-1  .  1-1-3  x  x  3-5  1-1-1  3  x-3  1-3  3-1  1-3  3
 x-1-3-1  1  .-1  1-1  1  3-1  1-1-1  3-3  1  x  1  x-1-3  1  3
 x-3-1-3-1-1  .-1  1  3  1  1  3-3  5-3  3-1-1  x  1-3-1-1  1
-3-1  x-1-3  1-1  .-1  1-1  3  1  x-1-1  1  5  1  1-1  x-3  1-1
 3-3-1-3  x-1  1-1  .-1  1-3  3  1  1  1-1-1  3-1  5-3-1  x  1
 x-1-3-1  x  1  3  1-1  .-1  3  1-1  3-1  x  1-3  5-1-1-3  1-1
 x-3-1-3  3  3  1-1  1-1  .-3  3-3  1  1-1-1-1-1  1-3-1-1  5
-1  1-1  1-5-1  1  3-3  3-3  .-1  1  1-3  3  x-1  3-3  1-1  3-3
 1  x  x-5  1  1  3  1  3  1  3-1  .-1  3-1  1  1  1  1  3-5-3-3  3
-1  1-1  1-1-1-3  x  1-1-3  1-1  .  x  1-1  3  3-1  1  1-1-1-3
 x-3-1-3-1-1  5-1  1  3  1  1  3  x  .  x  3-1-1  3  1-3-1-1  1
-1  1-1  1  3  3-3-1  1-1  1-3-1  1  x  .-5-1-1-1  1  1-1-1  1
 1-1  1-1  x-3  3  1-1  x-1  3  1-1  3-5  .  1  1  1-1-1  1  1-1
-3-1-3-1-3  1-1  5-1  1-1  x  1  3-1-1  1  .  1  1-1-1-3  1-1
 x-1  1-1  1  x-1  1  3-3-1-1  1  3-1-1  1  1  .-3  3-1  1-3-1
 1-1-3-1-3  1  x  1-1  5-1  3  1-1  3-1  1  1-3  .  x-1-3  1-1
 3-3-1-3  3  x  1-1  5-1  1-3  3  1  1  1-1-1  3  x  .-3-1-5  1
-1  5  3  5-1-1-3  x-3-1-3  1-5  1-3  1-1-1-1-1-3  .  3  x-3
 1  3  5  x  1-3-1-3-1-3-1-1-3-1-1-1  1-3  1-3-1  3  .  1-1
 x  3  1  3-3  1-1  1  x  1-1  3-3-1-1-1  1  1-3  1-5  x  1  .-1
-1-3-1-3  3  3  1-1  1-1  5-3  3-3  1  1-1-1-1-1  1-3-1-1  .
```

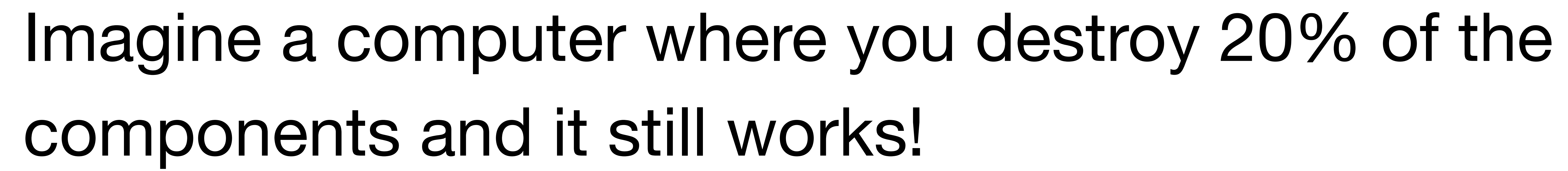











Imagine a computer where you destroy 20% of the components and it still works!

- Stability of memories is an issue to be considered
- For failure mode analysis (where hopfield networks fail to correctly restore memories), see MacKay Chapter 42