
COGS 181, Fall 2017

Neural Networks and Deep Learning

Lecture 6: Perceptron

Midterm 1

We will have midterm 1 on next Tuesday (10/24/2017)

Time: 12:30-1:50

Location: CSB 001

You can bring one page “cheat sheet”. No use of computers/smart-phones during the exam.

Bring your pen.

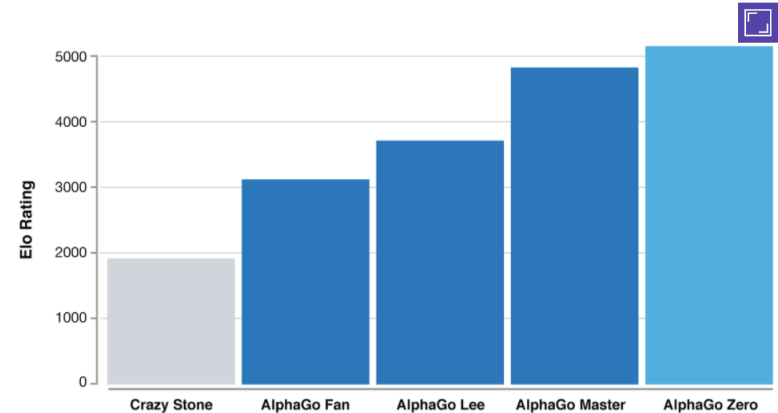
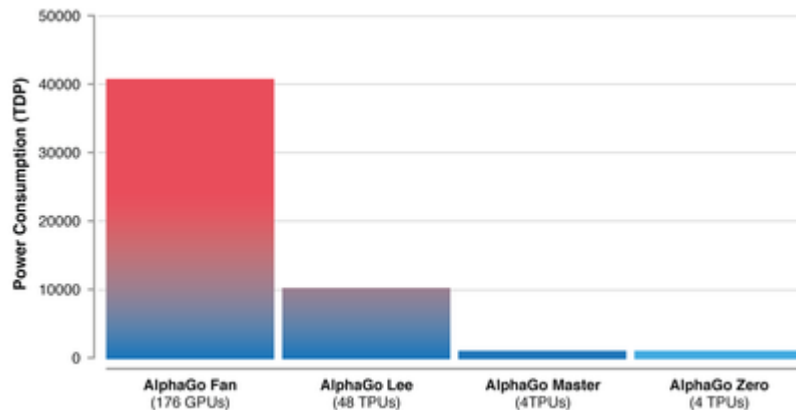
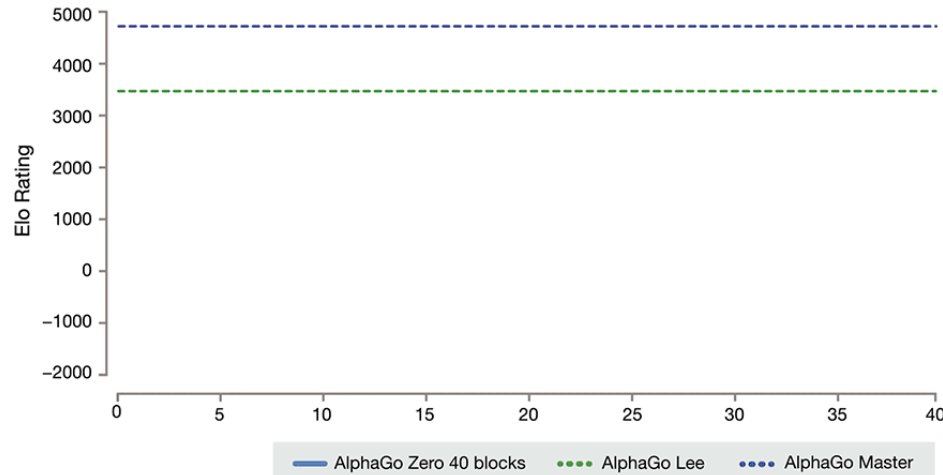
Bring your calculator, if you want.

What's new

<https://deepmind.com/blog/alphago-zero-learning-scratch/>

"Mastering the game of Go without human knowledge",

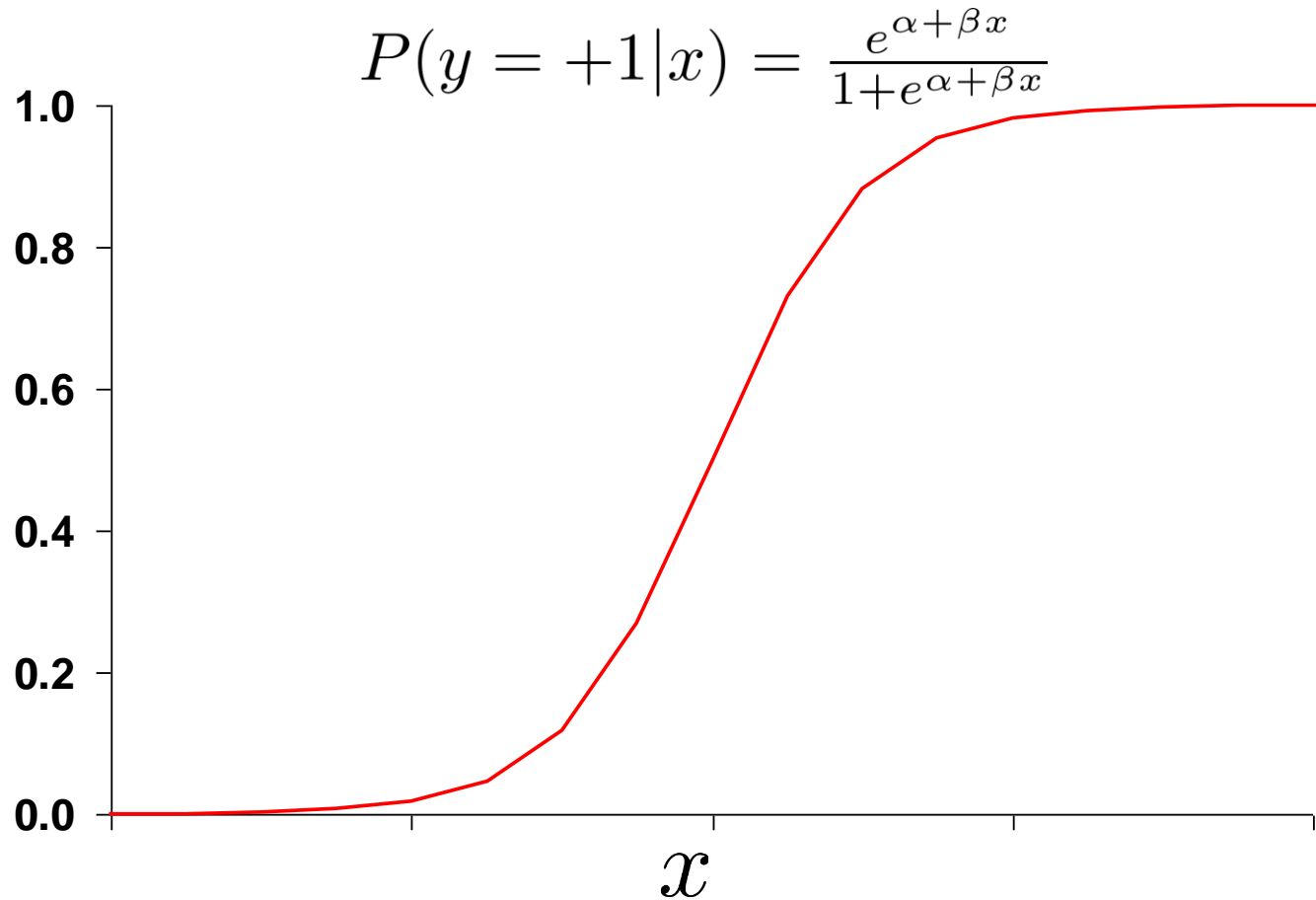
David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis, Nature 354–359, 2017.



Elo ratings - a measure of the relative skill levels of players in competitive games such as Go - show how AlphaGo has become progressively stronger during its development

The logistic function

Probability of
being positive



The logistic function

$$p(y = 1|x) = \frac{e^{\alpha + \beta x}}{1 + e^{\alpha + \beta x}} = \frac{1}{1 + e^{-(\alpha + \beta x)}}$$

$$\ln\left(\frac{p(y=1|x)}{1-p(y|x)}\right) = \alpha + \beta x \quad \text{logit of } p(y = 1|x)$$

Advantages of the logit

- Simple transformation of $P(y|x)$
- Linear relationship with x
- Can be continuous (Logit between $-\infty$ to $+\infty$)
- Known binomial distribution (P between 0 and 1)
- Directly related to the notion of odds of disease

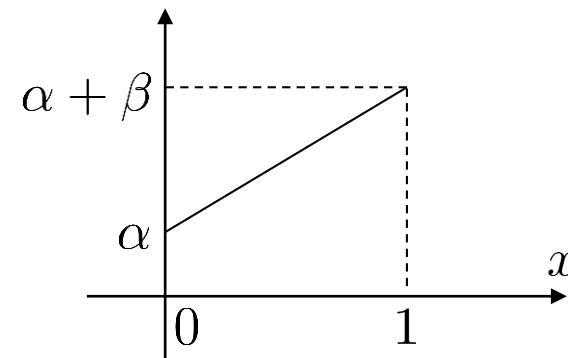
$$\ln\left(\frac{p}{1-p}\right) = \alpha + \beta x \quad \frac{p}{1-p} = e^{\alpha + \beta x}$$

The logistic function

$$\text{Logit of } P(y = +1|x): \ln\left(\frac{P(y=+1|x)}{1-P(y=+1|x)}\right) = \alpha + \beta x$$

Advantages of the logit:

1. Simple transformation of $p(y|x)$
2. Linear relationship with x
3. Can be continuous (Logit between $-\infty$ to $+\infty$)
4. Known binomial distribution (P between 0 and 1)



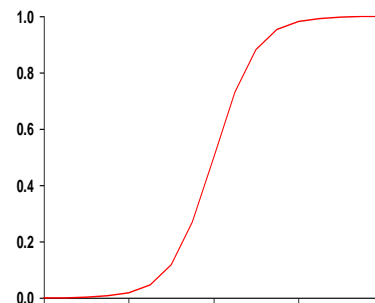
The logistic function

Logit of $P(y = +1|x)$: $\ln\left(\frac{P(y=+1|x)}{1-P(y=+1|x)}\right) = \alpha + \beta x$

$$P(y = +1|x) = \frac{e^{\alpha + \beta x}}{1 + e^{\alpha + \beta x}} = \frac{1}{1 + e^{-(\alpha + \beta x)}}$$

$$P(y = -1|x) = \frac{1}{1 + e^{\alpha + \beta x}}$$

$$P(y = +1|x) + P(y = -1|x) = 1$$



An agnostic notation: $P(y|x) = \frac{1}{1 + e^{-y(\alpha + \beta x)}}$

Training a logistic regression classifier

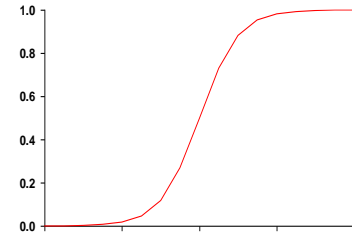
$$S_{training} = \{(x_i, y_i), i = 1..n\} \quad \begin{array}{l} x_i \in R, i = 1..n \\ y_i \in \{-1, +1\}, i = 1..n \end{array}$$

$$S_{training} = \{(-1.1, -1), (3.2, +1), (2.5, -1), (5.0, +1), (4.3, +1)\}$$

Train a classifier $f(x)$:

If $f(x)$ is a logistic regression classifier:

$$f(x) = \begin{cases} +1 & \text{if } \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}} \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$

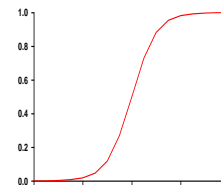


Training a logistic regression classifier

$$S_{training} = \{(-1.1, -1), (3.2, +1), (2.5, -1), (5.0, +1), (4.3, +1)\} \quad \begin{array}{l} x_i \in R, i = 1..n \\ y_i \in \{-1, +1\}, i = 1..n \end{array}$$

Train a logistic regression classifier $f(x)$:

$$P(y = +1|x) = \frac{1}{1+e^{-(\alpha+\beta x)}} \quad f(x) = \begin{cases} +1 & \text{if } \frac{1}{1+e^{-(\alpha+\beta x)}} \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$



$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n [P(y_i = +1|x_i)]^{0.5+y_i/2} \times [P(y_i = -1|x_i)]^{0.5-y_i/2}$$

$$P(y_i|x_i) = \frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} - \sum_{i=1}^n \ln\left(\frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}\right) = \arg \min_{(\alpha, \beta)} \sum_{i=1}^n \ln(1+e^{-y_i(\alpha+\beta x_i)})$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} [\ln(1+e^{(\alpha-1.1\beta)}) + \ln(1+e^{-(\alpha+3.2\beta)}) +$$

$$\ln(1+e^{(\alpha+2.5\beta)}) + \ln(1+e^{-(\alpha+5.0\beta)}) + \ln(1+e^{-(\alpha+4.3\beta)})]$$

Training a logistic regression classifier

We want to maximize the probability for a given training set. Suppose we have two points $\{(x_1, y_1 = +1), (x_2, y_2 = -1)\}$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^2 [P(y_i | x_i)]$$

where

$$P(y_i | x_i) = [P(y_i = +1 | x_i)]^{0.5+y_i/2} \times [P(y_i = -1 | x_i)]^{0.5-y_i/2}$$

In the example given above:

$$P(y_1 | x_1) = [P(y_1 = +1 | x_1)]^{0.5+1/2} \times [P(y_1 = -1 | x_1)]^{0.5-1/2} = [P(y_1 = +1 | x_1)] \times 1, \text{ since } y_1 = +1$$

$$P(y_2 | x_2) = [P(y_2 = +1 | x_2)]^{0.5-1/2} \times [P(y_2 = -1 | x_2)]^{0.5+1/2} = 1 \times [P(y_2 = -1 | x_2)], \text{ since } y_2 = -1$$

Training a logistic regression classifier

We want to maximize the probability for a given training set. Suppose we have two points $\{(x_1, y_1 = +1), (x_2, y_2 = -1)\}$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^2 [P(y_i | x_i)]$$

where

$$P(y_i | x_i) = [P(y_i = +1 | x_i)]^{0.5+y_i/2} \times [P(y_i = -1 | x_i)]^{0.5-y_i/2}$$

We then have the form:

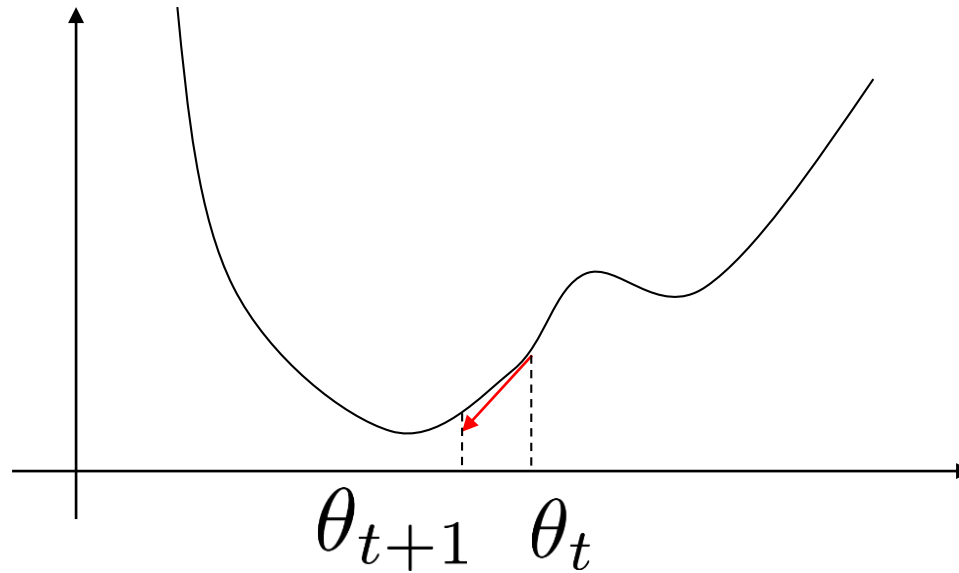
$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n [P(y_i = +1 | x_i)]^{0.5+y_i/2} \times [P(y_i = -1 | x_i)]^{0.5-y_i/2},$$

which can be simplified to:

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}$$

if we have: $P(y|x) = \frac{1}{1+e^{-y(\alpha+\beta x)}}$

Gradient descent (ascent)



Gradient Descent Direction

- (a) Pick a direction $\nabla g(x_t)$
- (b) Pick a step size λ_t
- (c) $\theta_{t+1} = \theta_t - \lambda_t \times \nabla g(\theta_t)$ such that function decreases
- (d) Repeat

Training a logistic regression classifier

$$P(y_i|x_i) = \frac{1}{1+e^{-y_i(\alpha+\beta x_i)}}$$

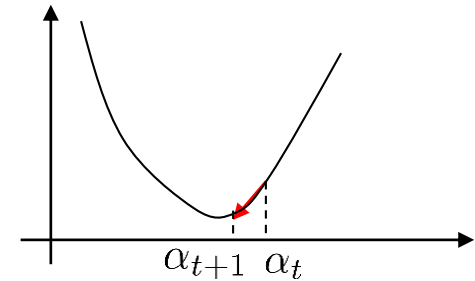
$$x_i \in R, i = 1..n$$

$$y_i \in \{-1, +1\}, i = 1..n$$

Train a logistic regression classifier $f(x)$:

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} g(\alpha, \beta)$$

$$g(\alpha, \beta) = \sum_{i=1}^n \ln(1 + e^{-y_i(\alpha+\beta x_i)})$$



$$\nabla_{\alpha} g(\alpha, \beta) = \sum_i \frac{-y_i e^{-y_i(\alpha+\beta x_i)}}{1+e^{-y_i(\alpha+\beta x_i)}} = \sum_i -y_i(1 - P(y_i|x_i))$$

$$\nabla_{\beta} g(\alpha, \beta) = \sum_i \frac{-y_i x_i e^{-y_i(\alpha+\beta x_i)}}{1+e^{-y_i(\alpha+\beta x_i)}} = \sum_i -y_i x_i(1 - P(y_i|x_i))$$

$$\alpha_{t+1} = \alpha_t - \lambda_t \times \nabla_{\alpha} g(\alpha_t, \beta_t)$$

$$\beta_{t+1} = \beta_t - \lambda_t \times \nabla_{\beta} g(\alpha_t, \beta_t)$$

Sometimes we also use $\{0, 1\}$ for y

$$S_{training} = \{(-1.1, 0), (3.2, 1), (2.5, 0), (5.0, 1), (4.3, 1)\}$$

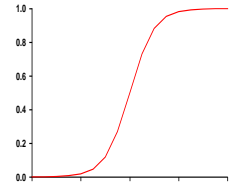
$$x_i \in R, i = 1..n$$

$$y_i \in \{0, 1\}, i = 1..n$$

Train a logistic regression classifier $f(x)$:

$$P(y = 1|x) = \frac{1}{1+e^{-(\alpha+\beta x)}} \quad f(x) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(\alpha+\beta x)}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$P(y = 0|x) = \frac{e^{-(\alpha+\beta x)}}{1+e^{-(\alpha+\beta x)}} = \frac{1}{1+e^{(\alpha+\beta x)}}$$



$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n [P(y_i = 1|x_i)]^{y_i} \times [P(y_i = 0|x_i)]^{1-y_i}$$

$$P(y_i|x_i) = \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

Sometimes we also use $\{0, 1\}$ for y

$$S_{training} = \{(-1.1, 0), (3.2, 1), (2.5, 0), (5.0, 1), (4.3, 1)\}$$

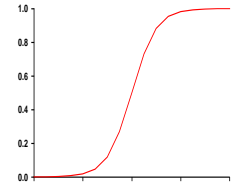
$$x_i \in R, i = 1..n$$

$$y_i \in \{0, 1\}, i = 1..n$$

Train a logistic regression classifier $f(x)$:

$$P(y = 1|x) = \frac{1}{1+e^{-(\alpha+\beta x)}} \quad f(x) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(\alpha+\beta x)}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$P(y = 0|x) = \frac{e^{-(\alpha+\beta x)}}{1+e^{-(\alpha+\beta x)}} = \frac{1}{1+e^{(\alpha+\beta x)}}$$



$$P(y_i|x_i) = \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \max_{(\alpha, \beta)} \prod_{i=1}^n \frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} - \sum_{i=1}^n \ln\left(\frac{1}{1+e^{-(2y_i-1)(\alpha+\beta x_i)}}\right) = \arg \min_{(\alpha, \beta)} \sum_{i=1}^n \ln(1+e^{-(2y_i-1)(\alpha+\beta x_i)})$$

$$(\alpha, \beta)^* = \arg \min_{(\alpha, \beta)} [\ln(1+e^{(\alpha-1.1\beta)}) + \ln(1+e^{-(\alpha+3.2\beta)}) +$$

$$\ln(1+e^{(\alpha+2.5\beta)}) + \ln(1+e^{-(\alpha+5.0\beta)}) + \ln(1+e^{-(\alpha+4.3\beta)})]$$

Multivariate input

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}$$

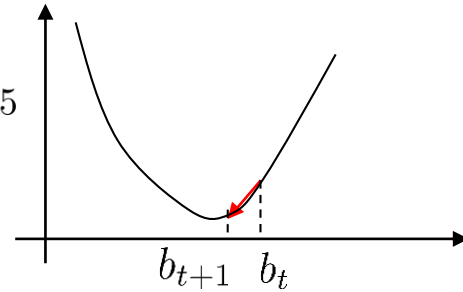
$$\mathbf{x}_i \in R^m, i = 1..n$$

$$y_i \in \{-1, +1\}, i = 1..n$$

Train a logistic regression classifier $f(\mathbf{x})$:

$$(b, \mathbf{w})^* = \arg \min_{(b, \mathbf{w})} g(b, \mathbf{w}) \quad f(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(b+\mathbf{w}^T\mathbf{x})}} \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$

$$g(b, \mathbf{w}) = \sum_{i=1}^n \ln(1 + e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)})$$



$$\nabla_b g(b, \mathbf{w}) = \sum_i \frac{-y_i e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -y_i(1 - P(y_i|\mathbf{x}_i))$$

$$\nabla_{\mathbf{w}} g(b, \mathbf{w}) = \sum_i \frac{-y_i \mathbf{x}_i e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -y_i \mathbf{x}_i(1 - P(y_i|\mathbf{x}_i))$$

$$b_{t+1} = b_t - \lambda_t \times \nabla_b g(b_t, \mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \times \nabla_{\mathbf{w}} g(b_t, \mathbf{w}_t)$$

Multivariate input and $y \in \{0, 1\}$

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}}$$

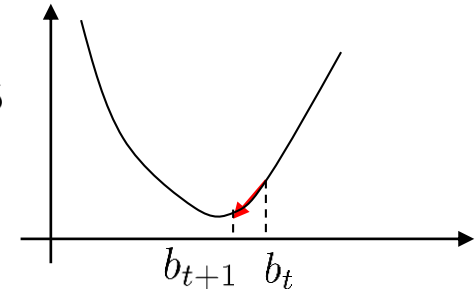
$$\mathbf{x}_i \in R^m, i = 1..n$$

$$y_i \in \{0, 1\}, i = 1..n$$

Train a logistic regression classifier $f(\mathbf{x})$:

$$(b, \mathbf{w})^* = \arg \min_{(b, \mathbf{w})} g(b, \mathbf{w}) \quad f(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(b+\mathbf{w}^T\mathbf{x})}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$g(b, \mathbf{w}) = \sum_{i=1}^n \ln(1 + e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)})$$



$$\nabla_b g(b, \mathbf{w}) = \sum_i \frac{-(2y_i-1)e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -(2y_i-1)(1-P(y_i|\mathbf{x}_i))$$

$$\nabla_{\mathbf{w}} g(b, \mathbf{w}) = \sum_i \frac{-(2y_i-1)\mathbf{x}_i e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}}{1+e^{-(2y_i-1)(b+\mathbf{w}^T\mathbf{x}_i)}} = \sum_i -(2y_i-1)\mathbf{x}_i(1-P(y_i|\mathbf{x}_i))$$

$$b_{t+1} = b_t - \lambda_t \times \nabla_b g(b_t, \mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda_t \times \nabla_{\mathbf{w}} g(b_t, \mathbf{w}_t)$$

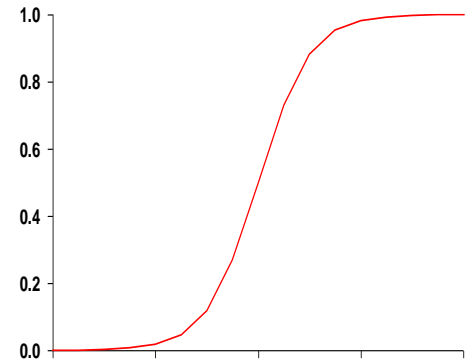
Logistic regression classifier

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i(b+\mathbf{w}^T\mathbf{x}_i)}}$$

$$\mathbf{x} \in R^m$$

$$y \in \{-1, +1\}$$

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(b+\mathbf{w}^T\mathbf{x})}} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$



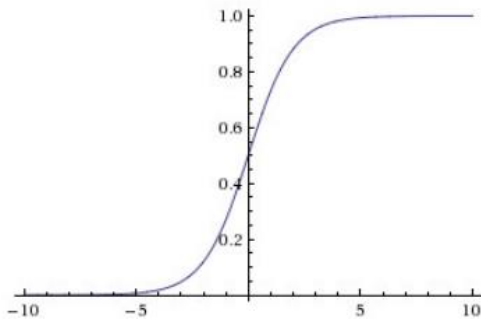
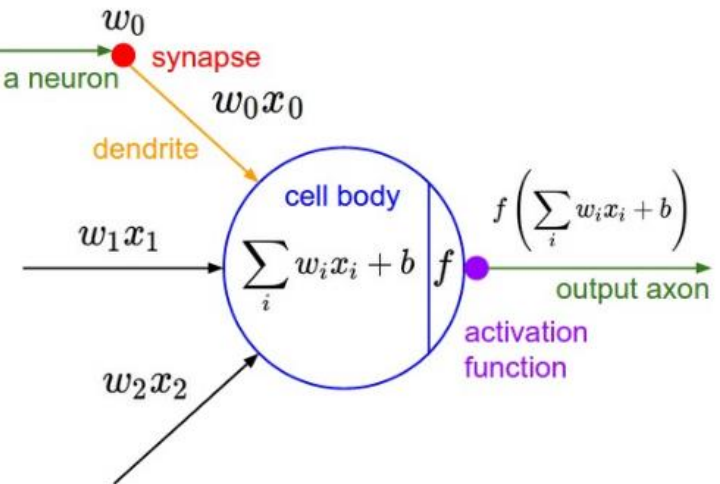
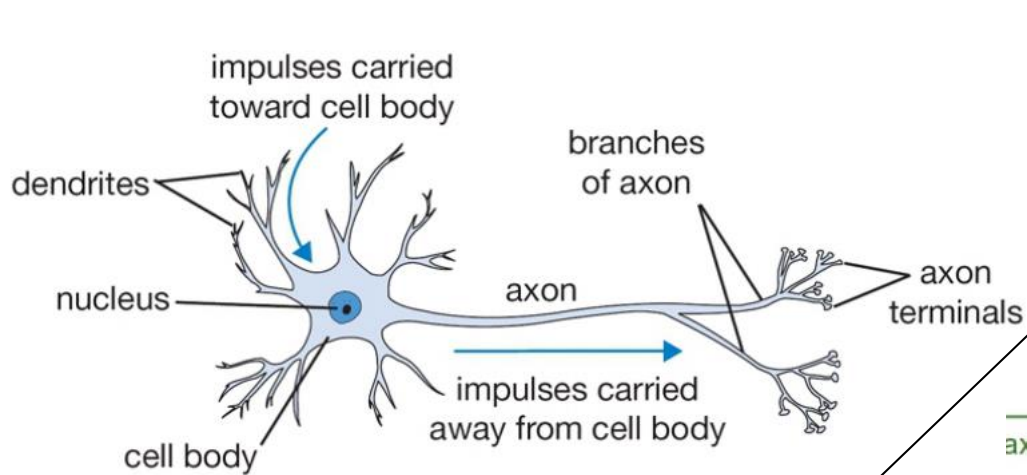
Pros:

1. It is well-normalized.
2. Easy to turn into probability.
3. Easy to implement.

Cons:

1. Indirect loss function.
2. Dependent on good feature set.
3. Weak on feature selection.

Perceptron



Sigmoid function
$$f(x) = \frac{1}{1+e^{-x}}$$

Perceptron

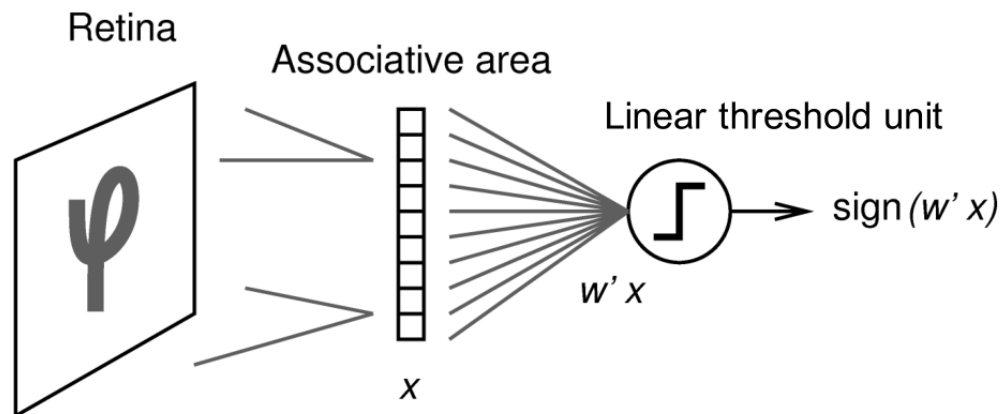
Let's look at a very simple form.

Slides modified from Jake Olson's lecture.

Perceptron



Frank Rosenblatt

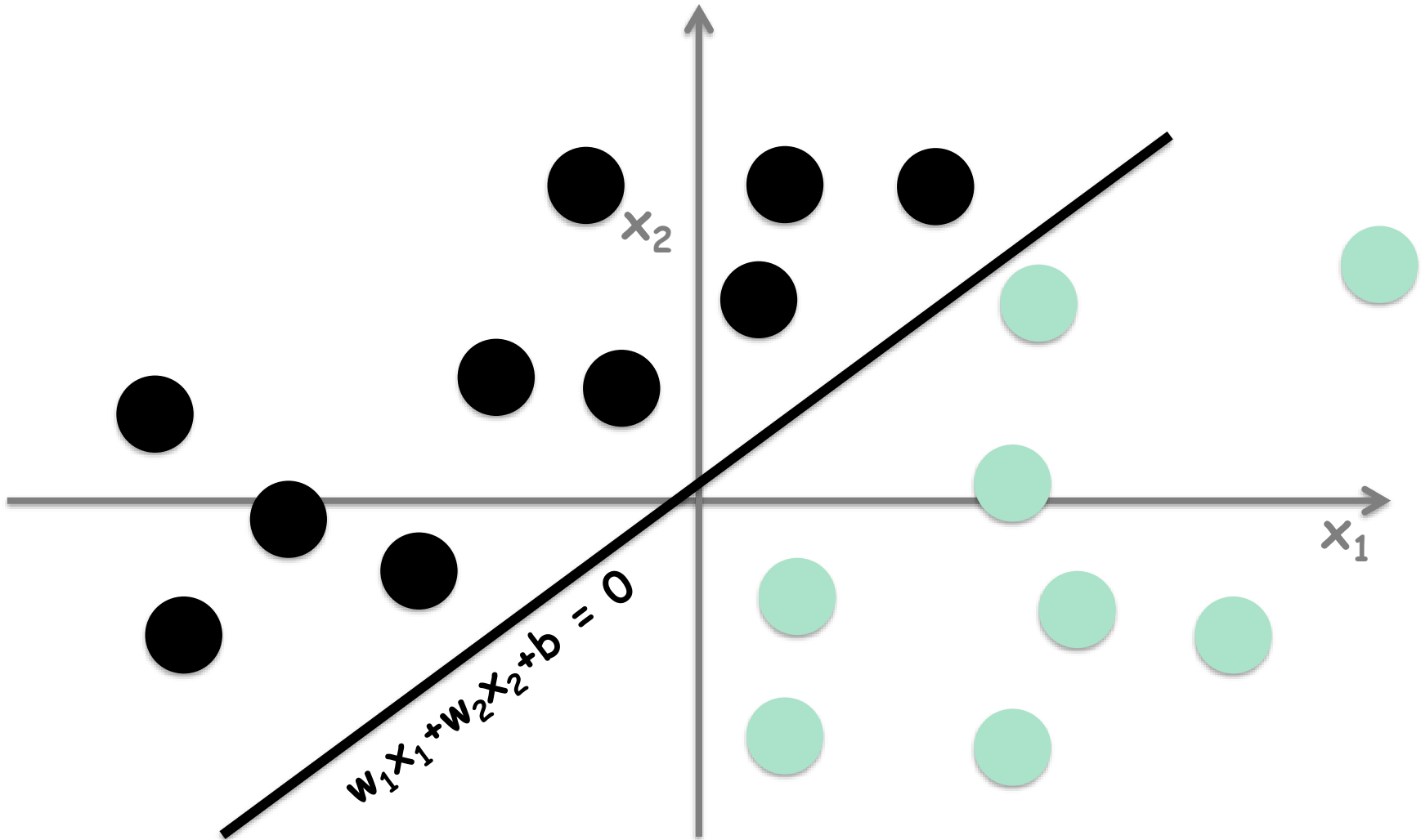


Supervised learning of the weights w using the Perceptron algorithm.

Linear classifier

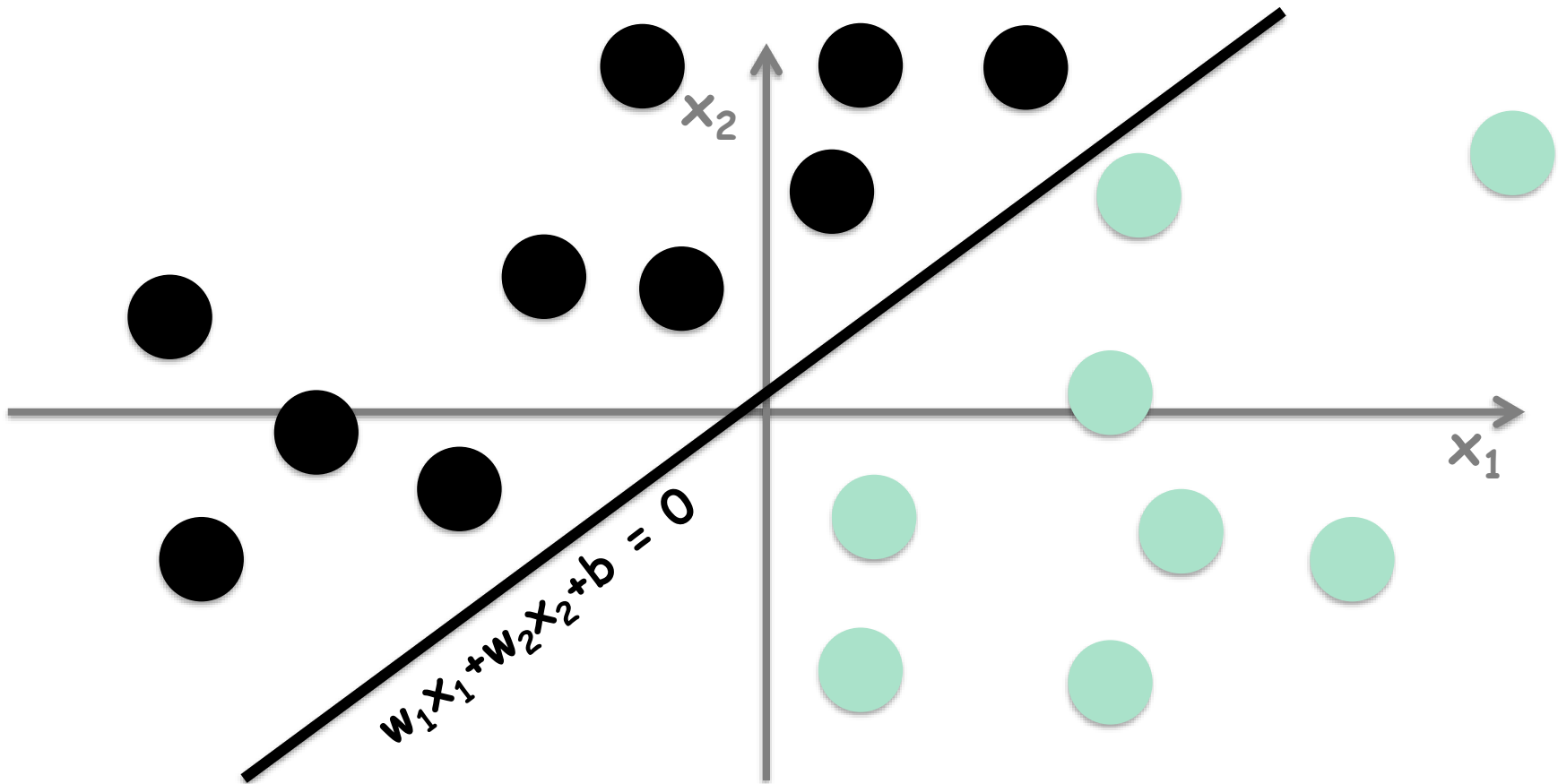
Find: $\arg \min_{\mathbf{w}} \sum_{i=1}^n \mathbf{1}(y_i \neq \text{sign}(w_1 x_{i1} + w_2 x_{i2} + b))$

Linear decision boundary: $w_1 x_1 + w_2 x_2 + b = 0$



Linear classifier

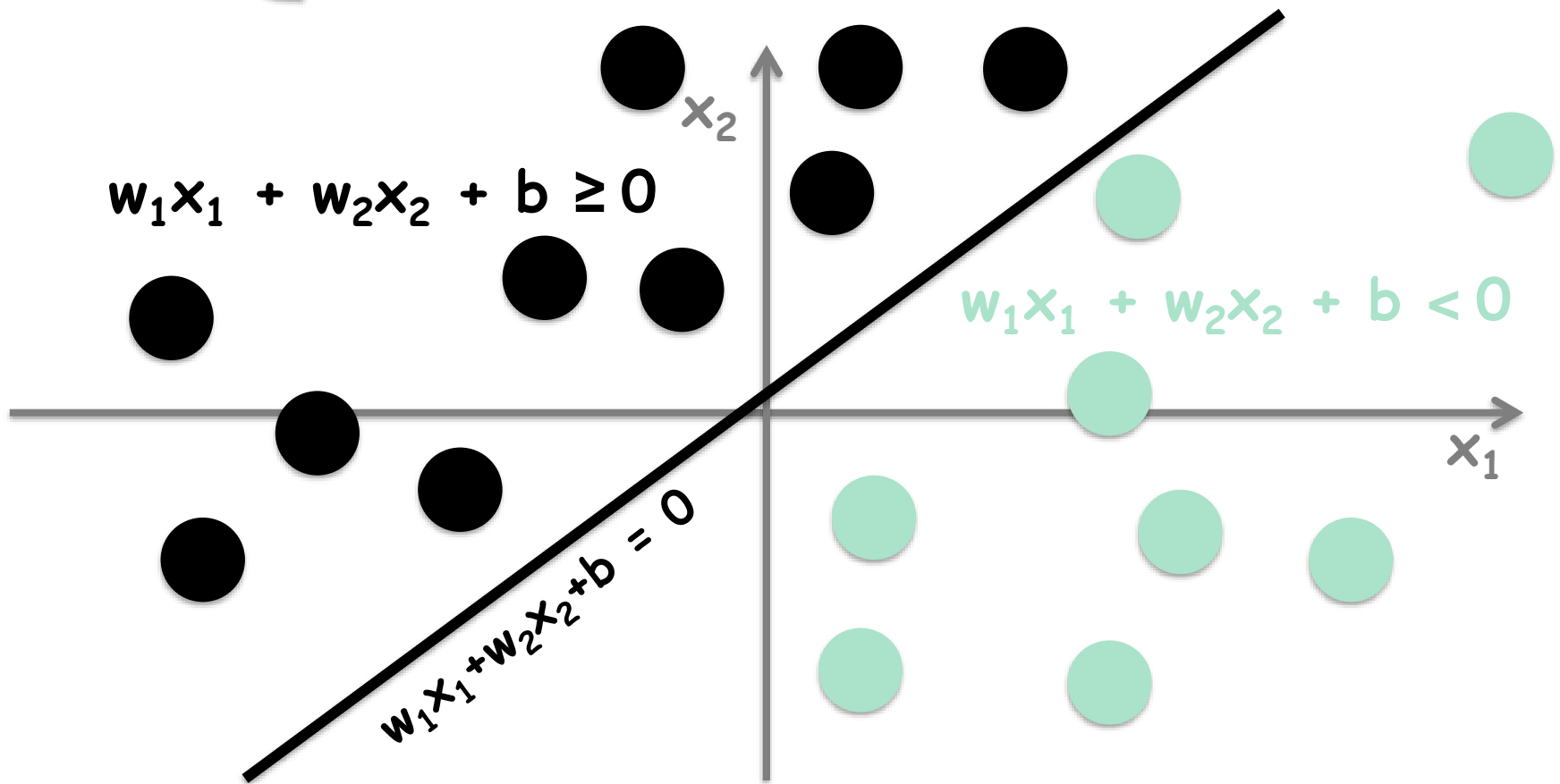
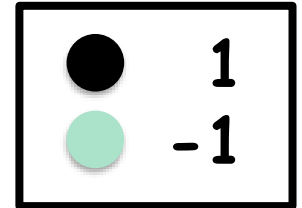
Linear decision boundary: $w_1x_1 + w_2x_2 + b = 0$



Linear classifier

Linear decision boundary and a classification

$$f(x) = \begin{cases} 1 & w_1x_1 + w_2x_2 + b \geq 0 \\ -1 & w_1x_1 + w_2x_2 + b < 0 \end{cases}$$



Perceptron Learning Algorithm

- Initialize the weights (however you choose)
 - $w_1x_1 + w_2x_2 + b$ (initialize w_1 , w_2 , and b)
- Step 1: Choose a data point.
- Step 2: Compute the model output for the datapoint.
- Step 3: Compare model output to the target output.
 - If correct classification, go to Step 5!
 - If not, go to Step 4.
- Step 4: Update weights using perceptron learning rule. Start over on Step 1 with the first data point.

Or

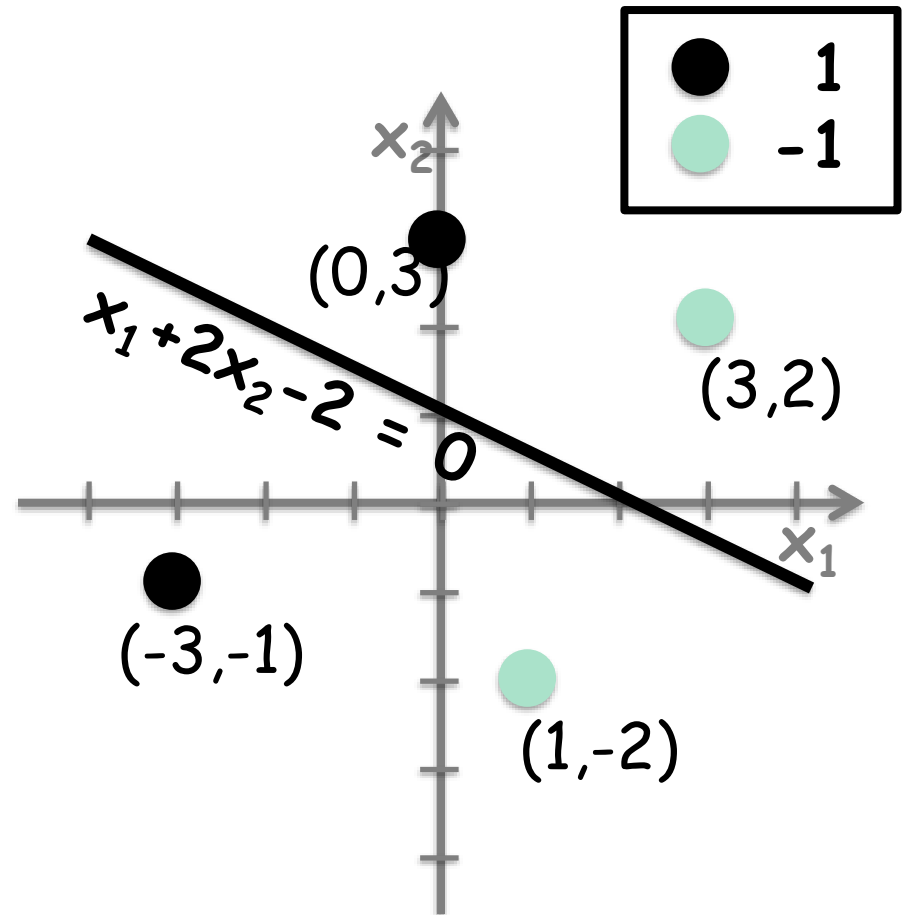
$$W_{new} = W_{old} + (target_i - output_i)x_i$$
$$b_{new} = b_{old} + (target_i - output_i)$$

$$W_{new} = W_{old} + \lambda(target_i - output_i)x_i$$
$$b_{new} = b_{old} + \lambda(target_i - output_i)$$

- Step 5: Go to the next data point. If you have gone through them all, you have found the solution!

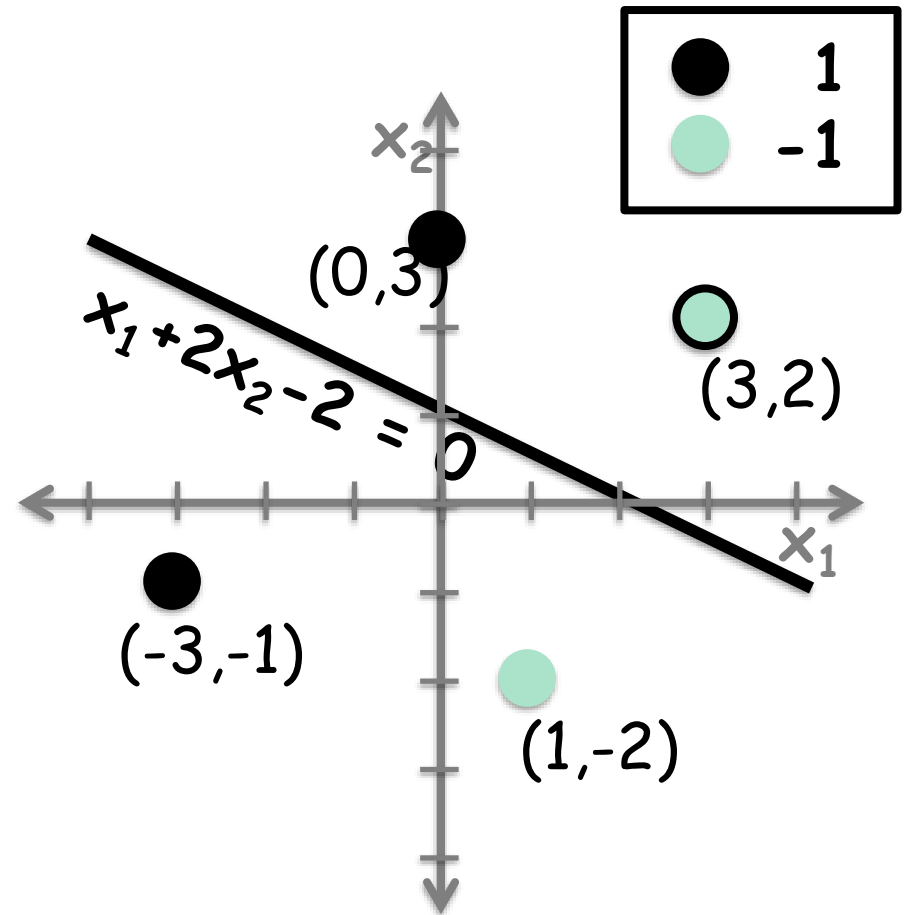
Initialize the weights

- Choose randomly – but start the weights small!
- We'll choose:
 - $w_1 = 1$
 - $w_2 = 2$
 - $b = -2$
- Generally:
 - $w_1x_1 + w_2x_2 + b = 0$
- Applying our values:
 - $x_1 + 2x_2 - 2 = 0$



Step 1: Choose a point

- Choose randomly (or sequentially), but remember which you choose!



Step 2: Compute the model output

Inputs

$$\mathbf{x}_1 = 3$$

$$\mathbf{x}_2 = 2$$

$$\text{output} = 1$$

Use the inputs and the linear boundary equation to find the “net activity.”

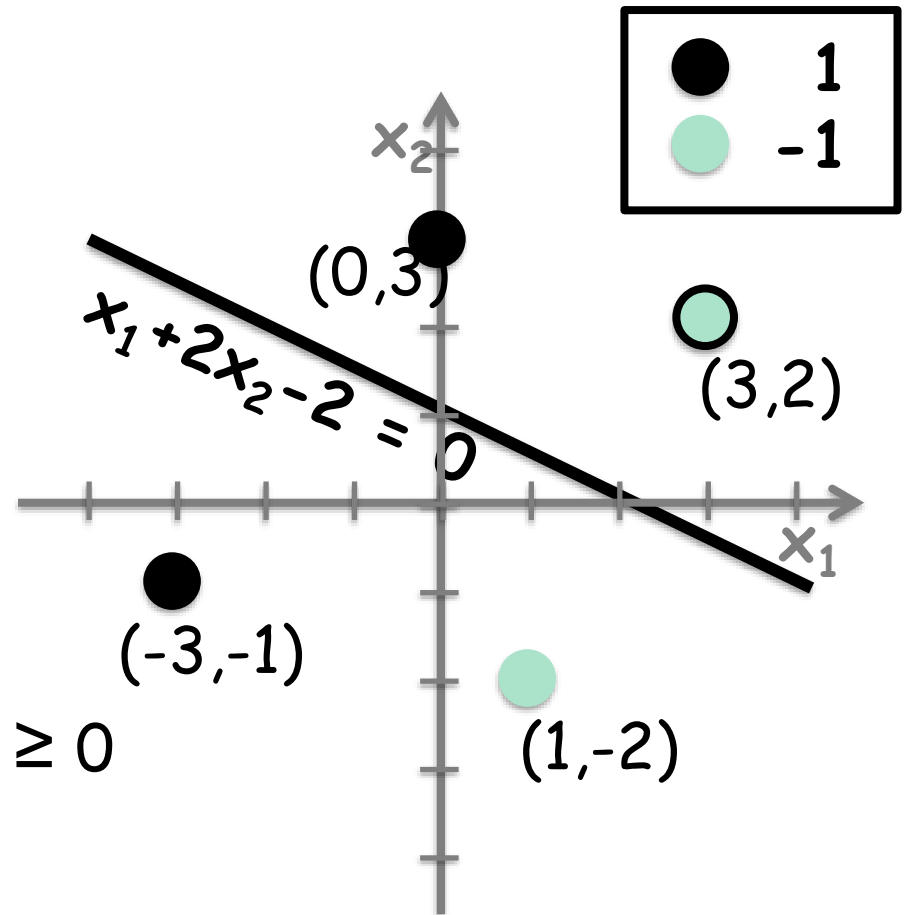
$$\text{net} = \mathbf{x}_1 + 2\mathbf{x}_2 - 2$$

$$\text{net} = (3) + 2(2) - 2$$

$$\text{net} = 5$$

Find the model output using the classification (step) *activation function*.

$$\text{output} = f(\text{net}) = \begin{cases} 1 & \text{net} \geq 0 \\ -1 & \text{net} < 0 \end{cases}$$



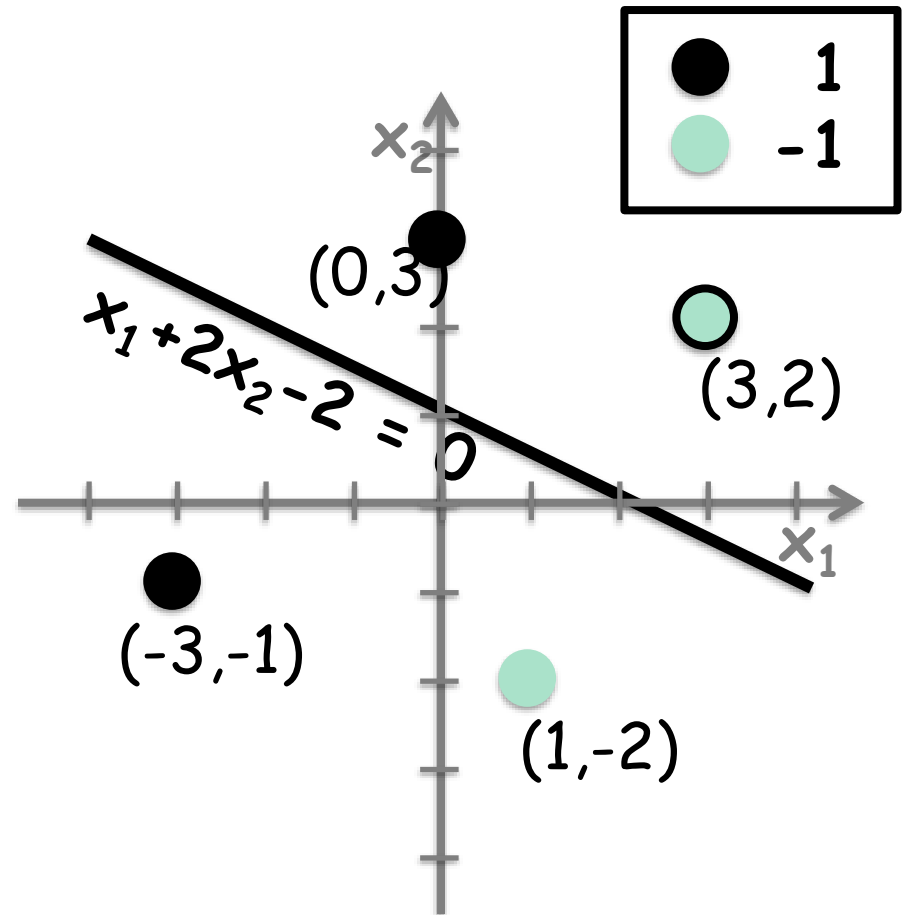
Step 3: Compare the model output and target values

Does the model output equal the target value?

If yes, do another point until you have done them all!

If no, update the weights.

output = 1
target = -1
failure :(



Step 4: Update weights using perceptron learning rule.

- We need to improve! How?
 - Move the line to a better spot!

- How?

$$w_{new} = w_{old} + (target_i - output_i) x_i$$

$$b_{new} = b_{old} + (target_i - output_i)$$

For this example:

$$w_1 = 1 + (-1 - 1) * 3 = -5$$

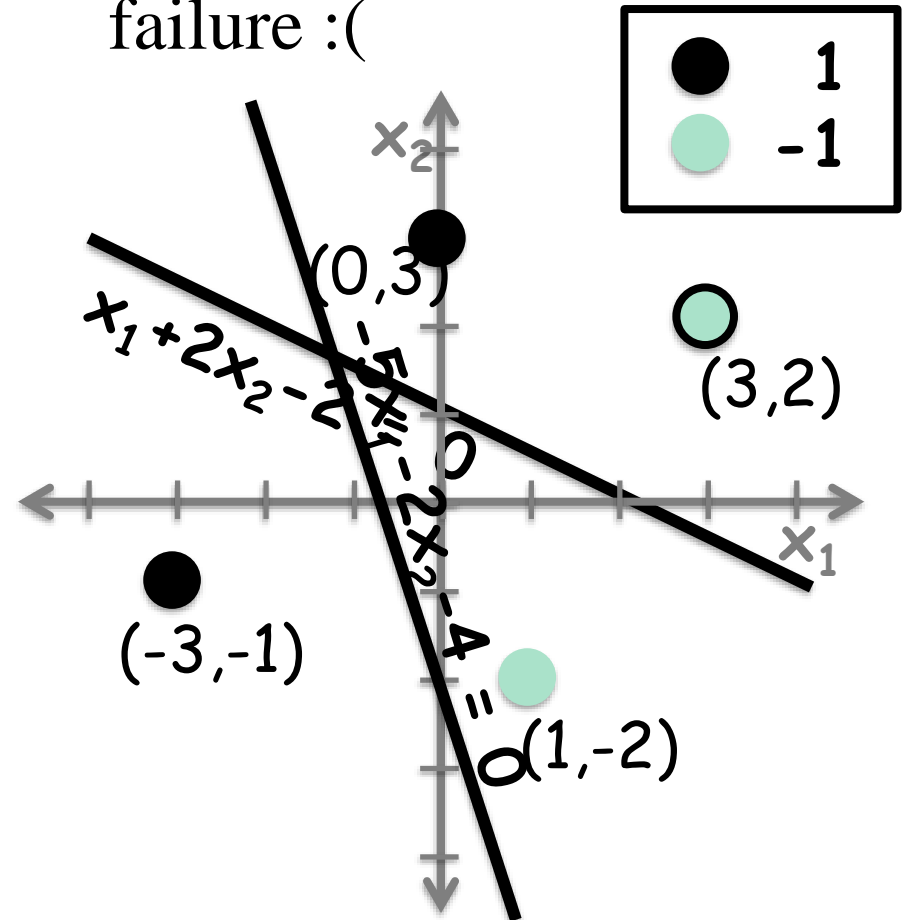
$$w_2 = 2 + (-1 - 1) * 2 = -2$$

$$b = -2 + (-1 - 1) = -4$$

output = 1

target = -1

failure :(



Perceptron Learning Algorithm

- Initialize the weights (however you choose)
 - $w_1x_1 + w_2x_2 + b$ (initialize w_1 , w_2 , and b)
- Step 1: Choose a data point.
- Step 2: Compute the model output for the datapoint.
- Step 3: Compare model output to the target output.
 - If correct classification, go to Step 5!
 - If not, go to Step 4.
- Step 4: Update weights using perceptron learning rule. Start over on Step 1 with the first data point.

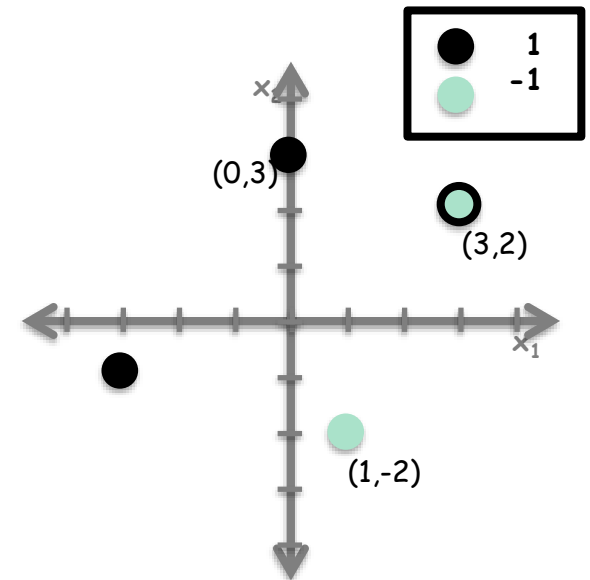
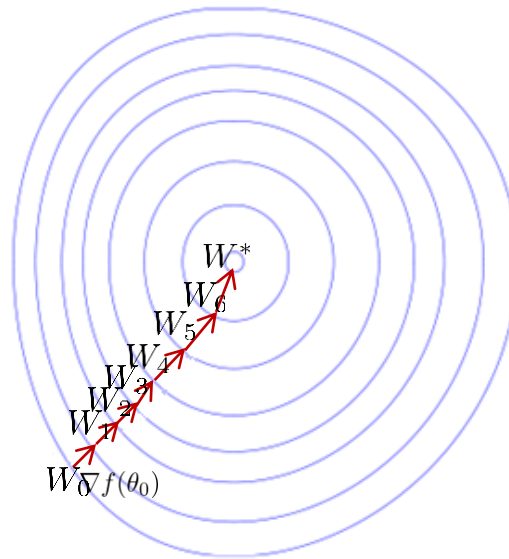
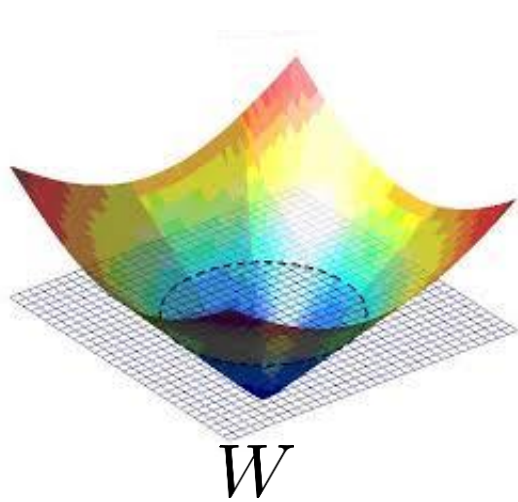
Or

$$W_{new} = W_{old} + (target_i - output_i)x_i$$
$$b_{new} = b_{old} + (target_i - output_i)$$

$$W_{new} = W_{old} + \lambda(target_i - output_i)x_i$$
$$b_{new} = b_{old} + \lambda(target_i - output_i)$$

- Step 5: Go to the next data point. If you have gone through them all, you have found the solution!

Perceptron training is a special gradient descent algorithm



$$W_{t+1} \leftarrow W_t - \nabla f(W_t)$$

$$f(W_t) = (f(\mathbf{x}_i) - y_i)\mathbf{x}_i$$

The XOR problem that kills the Perceptron algorithm

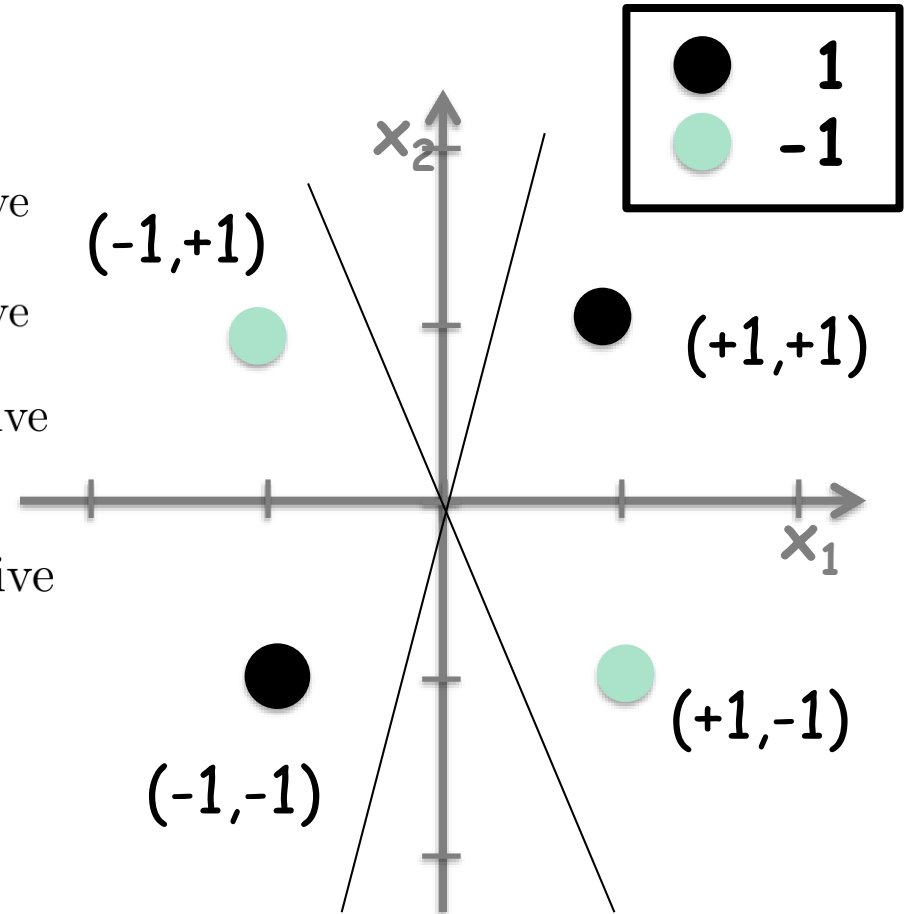
The XOR operator: \oplus

Positive (+1) \oplus Negative (-1) = Negative

Positive (+1) \oplus Positive (+1) = Positive

Negative (-1) \oplus Positive (+1) = Negative

Negative (-1) \oplus Negative (-1) = Positive



AI after Minsky's book *Perceptron*



Artificial neural networks: a brief history

Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain".

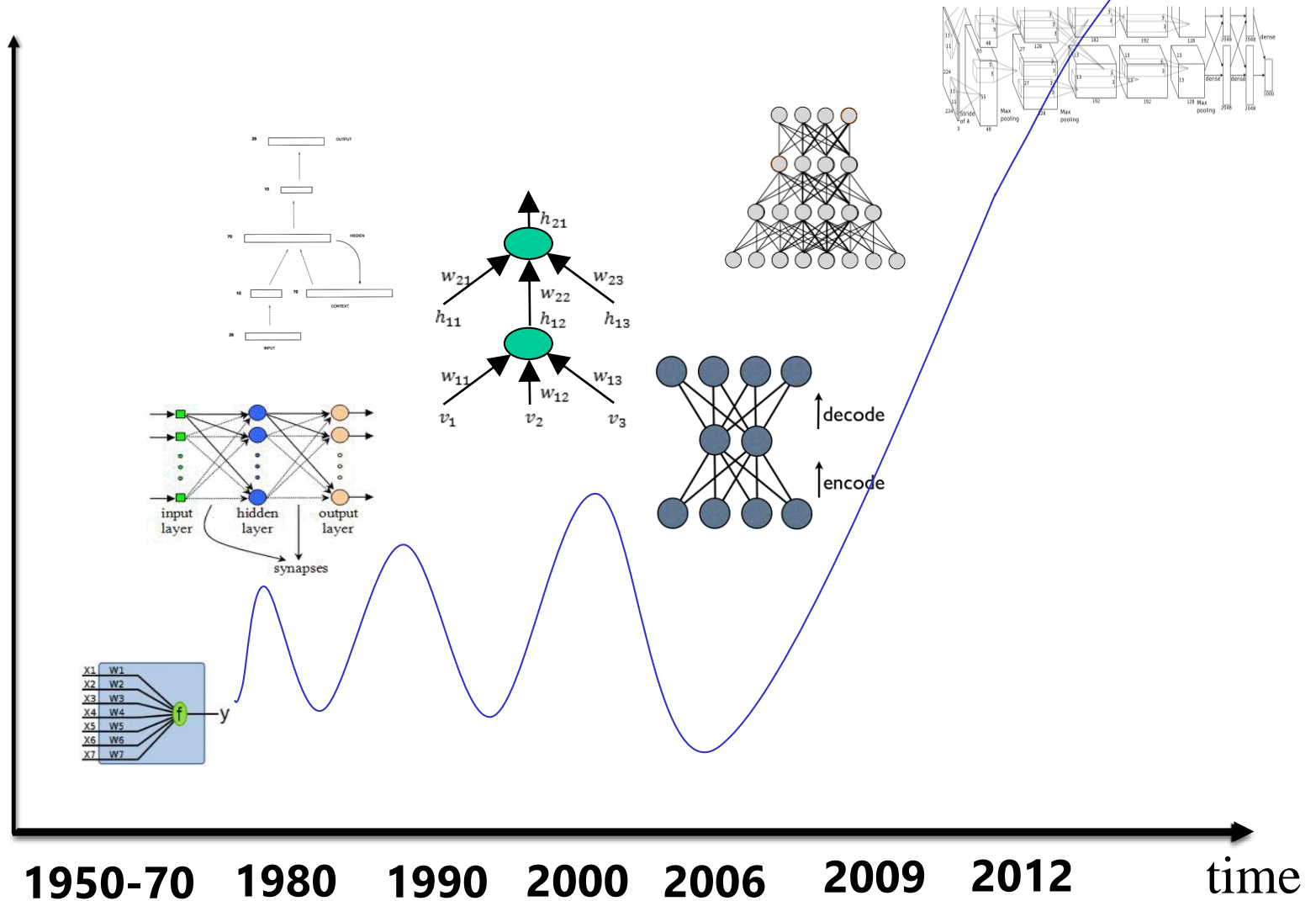
Hopfield J. (1982), "Neural networks and physical systems with emergent collective computational abilities", PNAS.

Rumelhart D., Hinton G. E., Williams R. J. (1986), "Learning internal representations by error-propagation".

Elman, J.L. (1990). "Finding Structure in Time".

Hinton, G. E.; Osindero, S.; Teh, Y. (2006). "A fast learning algorithm for deep belief nets".

Krizhevsky, A.; Sutskever, I.; Hinton, G. (2012) "ImageNet Classification with Deep Convolutional Neural Networks"



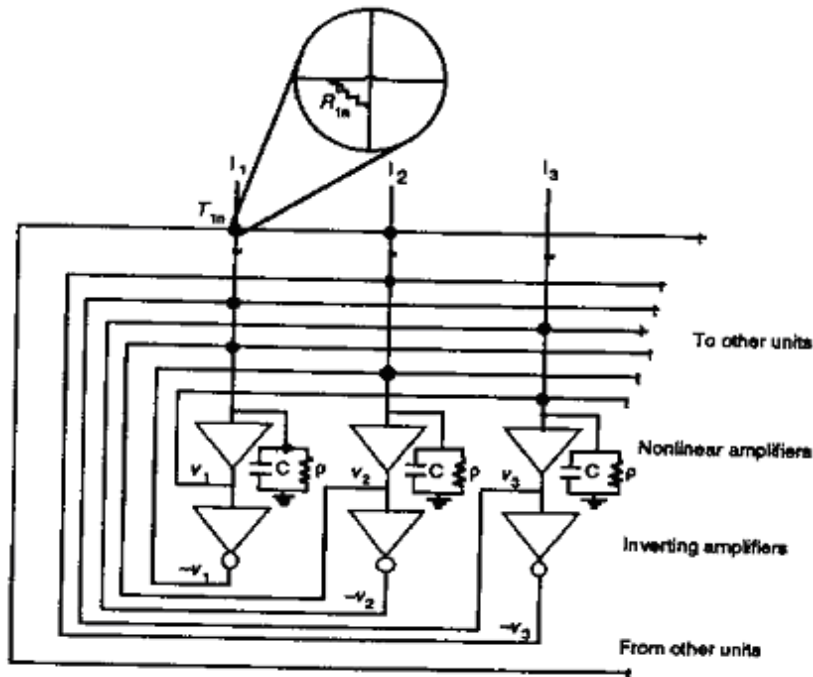
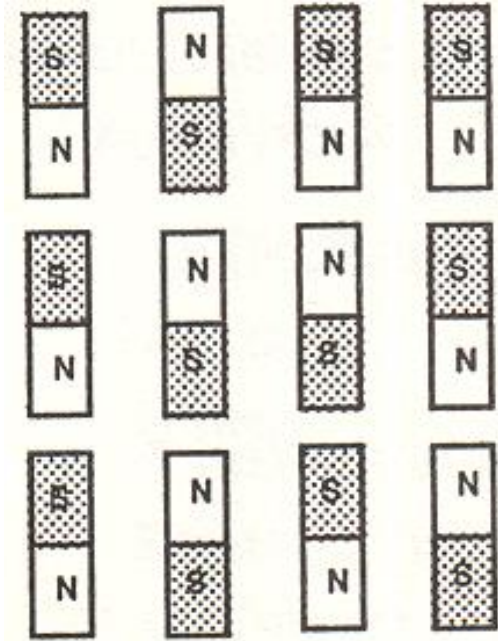
Hopfield Neural Networks

- Fundamentals of Hopfield Net
- Analog Implementation
- Associate Retrieval
- Solving Optimization Problem



Hopfield Neural Networks

The physicist Hopfield showed that models of physical systems could be used to solve computational problems



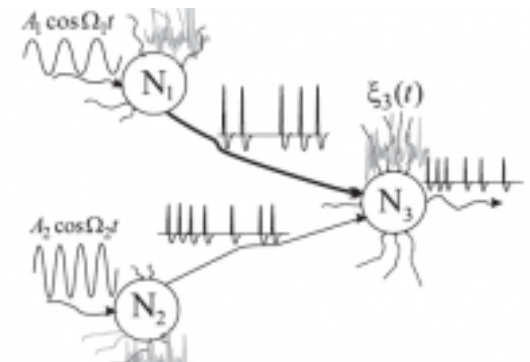
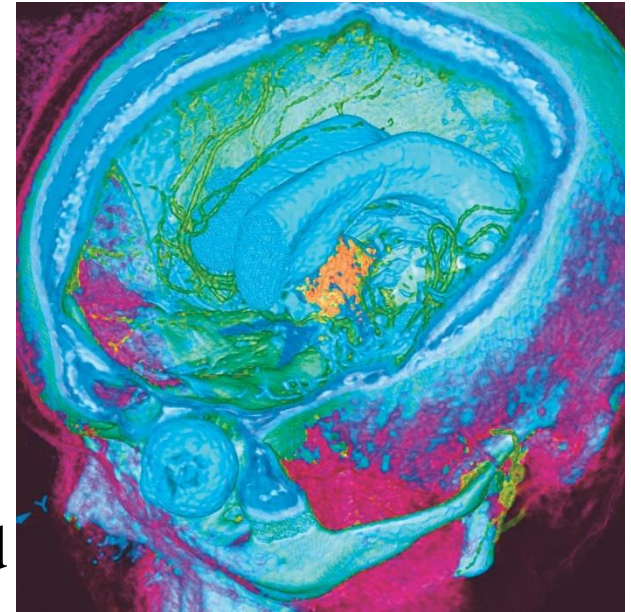
Such systems could be implemented in hardware by combining standard components such as capacitors and resistors.

The concept

Like all computers, a brain is a dynamical system that carries out its computations by the change of its 'state' with time.

Simple models of the dynamics of neural circuits are described that have collective dynamical properties. These can be exploited in recognizing sensory patterns.

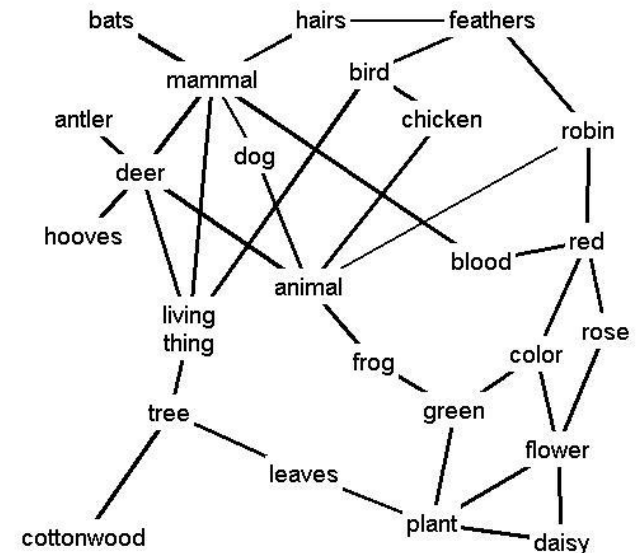
Using these collective properties in processing information is effective in that it exploits the spontaneous properties of nerve cells and circuits to produce robust computation.



J. Hopfield's quest

While the brain is totally unlike modern computers, much of what it does can be described as computation.

Associative memory, logic and inference, recognizing an odor or a chess position, parsing the world into objects, and generating appropriate sequences of locomotor muscle commands are all describable as computation.

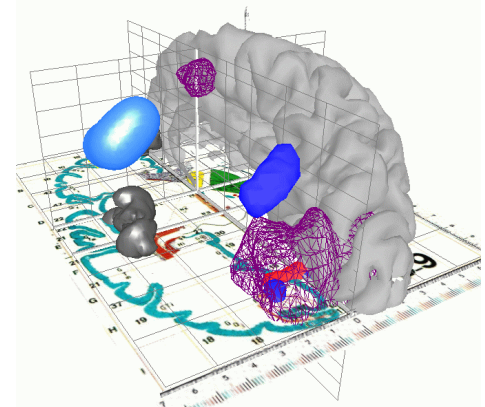
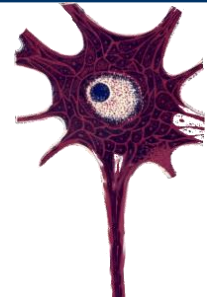


His research focuses on understanding how the neural circuits of the brain produce such powerful and complex computations.

Action potential computation

For much of neurobiology, information is represented by the paradigm of “*firing rates*”,

i.e. information is represented by the rate of generation of action potential spikes, and the exact timing of these spikes is unimportant.

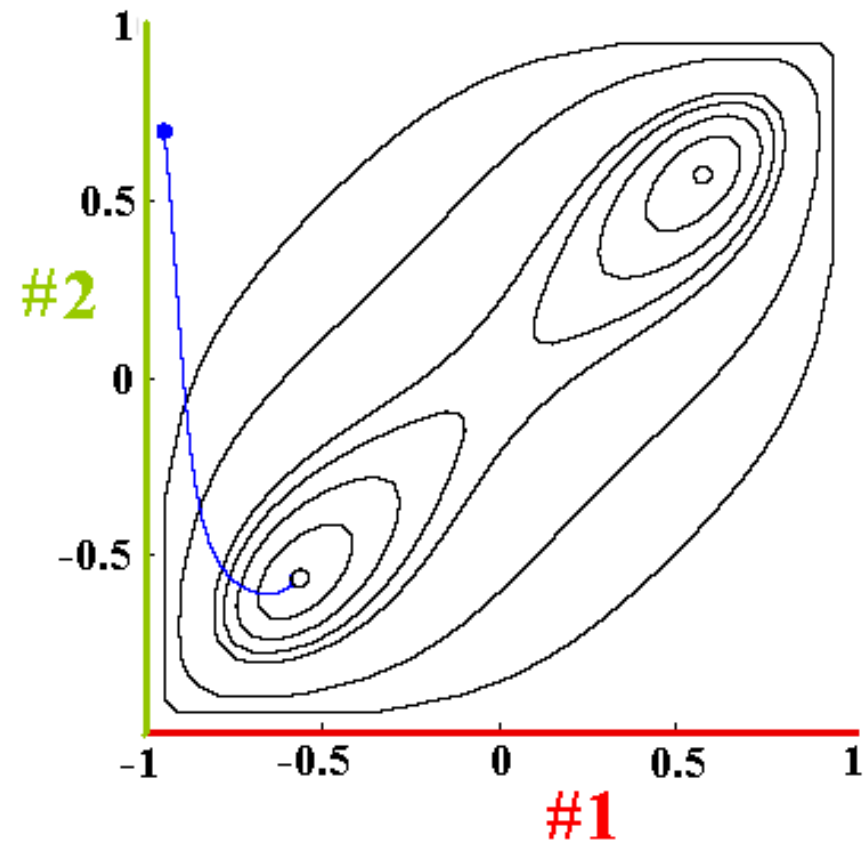
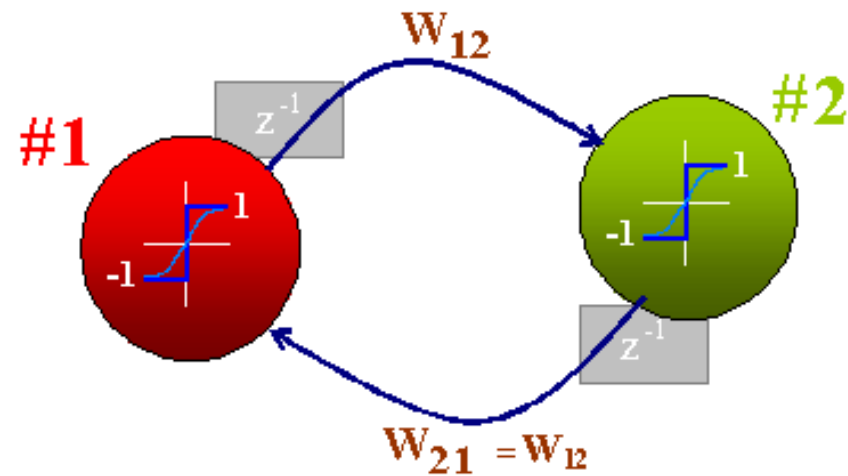


Hopfield Neural Networks

- Fundamentals of Hopfield Net
- Analog Implementation
- Associate Retrieval
- Solving Optimization Problem



An illustration: A 2-neurons Hopfield network of continuous states characterized by 2 stable states

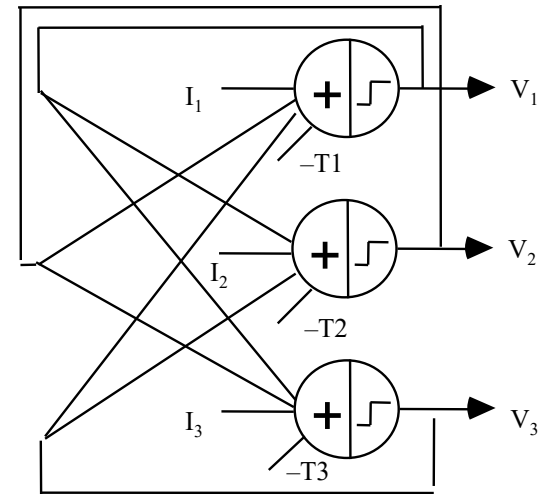


Fundamentals of Hopfield Net

- Proposed by J.J. Hopfield. A fully Connected, feed-back, fixed weight network.
- Each neuron accepts its input from the outputs of all other neurons and the its own input:

Net function: $u_i = \sum_{i \neq j} w_{ij} v_j + I_i$

Output: $v_i = \begin{cases} 1 & \text{if } u_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$



Discrete Time Formulation

Define:

$$V = [v_1, v_2, \dots, v_n]^T, \mathbf{I} = [I_1, I_2, \dots, I_n]^T$$

$$\mathbf{T} = [T_1, T_2, \dots, T_n]^T,$$

$$W = \begin{bmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1n} \\ w_{21} & 0 & w_{23} & \cdots & w_{2n} \\ w_{31} & w_{32} & 0 & \cdots & w_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \cdots & 0 \end{bmatrix}$$

$$\text{Then } V(t+1) = \text{sign}\{WV(t) + \mathbf{I}(t) - \mathbf{T}(t)\}$$

One Example

Updating rule: $V(t+1) = \text{sign}\{WV(t) + \mathbf{I}(t) - \mathbf{T}(t)\}$

Let: $V(0) = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \quad W = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \quad \mathbf{I} = \mathbf{T} = \mathbf{0}$

Then: $V(1) = \text{sign}(WV(0)) = \text{sign}\left(\begin{bmatrix} 1 \\ 1 \\ 3 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$

$$V(2) = \text{sign}(WV(1)) = \text{sign}\left(\begin{bmatrix} 3 \\ 3 \\ 3 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

One Example

Updating rule: $V(t+1) = \text{sign}\{WV(t) + \mathbf{I}(t) - \mathbf{T}(t)\}$

Let: $V(3) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$ $W = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$ $\mathbf{I} = \mathbf{T} = \mathbf{0}$

Then: $V(t+1) = \text{sign}(WV(t)) = \text{sign}\left(\begin{bmatrix} 3 \\ 3 \\ 3 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = V(t)$

The network converged!

Note that multiple solutions might exist.

One Example

Updating rule: $V(t+1) = \text{sign}\{WV(t) + \mathbf{I}(t) - \mathbf{T}(t)\}$

Let: $V(0) = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \quad W = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \quad \mathbf{I} = \mathbf{T} = \mathbf{0}$

Note that multiple solutions might exist: there are two in this case.

State 1:

$$\text{sign}\left(\begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

State 2:

$$\text{sign}\left(\begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

Example 2

Step_1. Design a network with memorized patterns (vectors) $[1, -1, 1]$ & $[-1, 1, -1]$

- Compute the outer products, sum, normalize, and set diagonals to 0:

$$(1, -1, 1)^T * (1, -1, 1) + (-1, 1, -1)^T * (-1, 1, -1) =$$

$\begin{bmatrix} 2 & -2 & 2 \\ -2 & 2 & -2 \\ 2 & -2 & 2 \end{bmatrix} \xrightarrow{\frac{1}{3}} \begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$

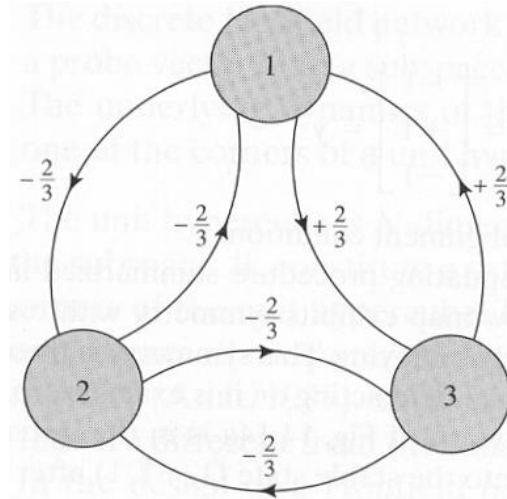
Force Diagonal to 0

$\frac{1}{3} = 1/(\text{number of neurons})$

$$\mathbf{W} = \frac{1}{3} \begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix} [+1, -1, +1] + \frac{1}{3} \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix} [-1, +1, -1] - \frac{2}{3} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Step_2. Initialization

$$\mathbf{W} = \frac{1}{3} \begin{bmatrix} 0 & -2 & +2 \\ -2 & 0 & -2 \\ +2 & -2 & 0 \end{bmatrix}$$

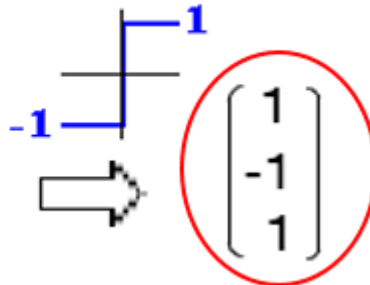


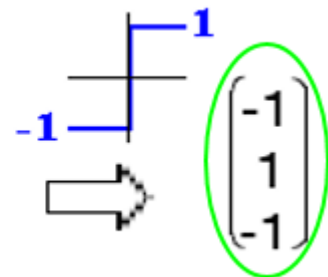
There are 8 different states that can be reached by the net and therefore can be used as its initial state

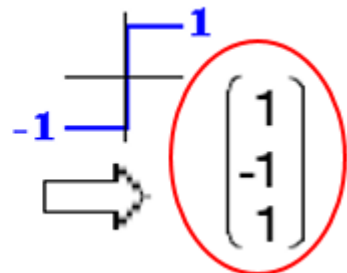
$$\begin{array}{l} \text{\#1: } \mathbf{V}_1 \\ \text{\#2: } \mathbf{V}_2 \\ \text{\#3: } \mathbf{V}_3 \end{array} \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}$$

Step_3. Iterate till convergence

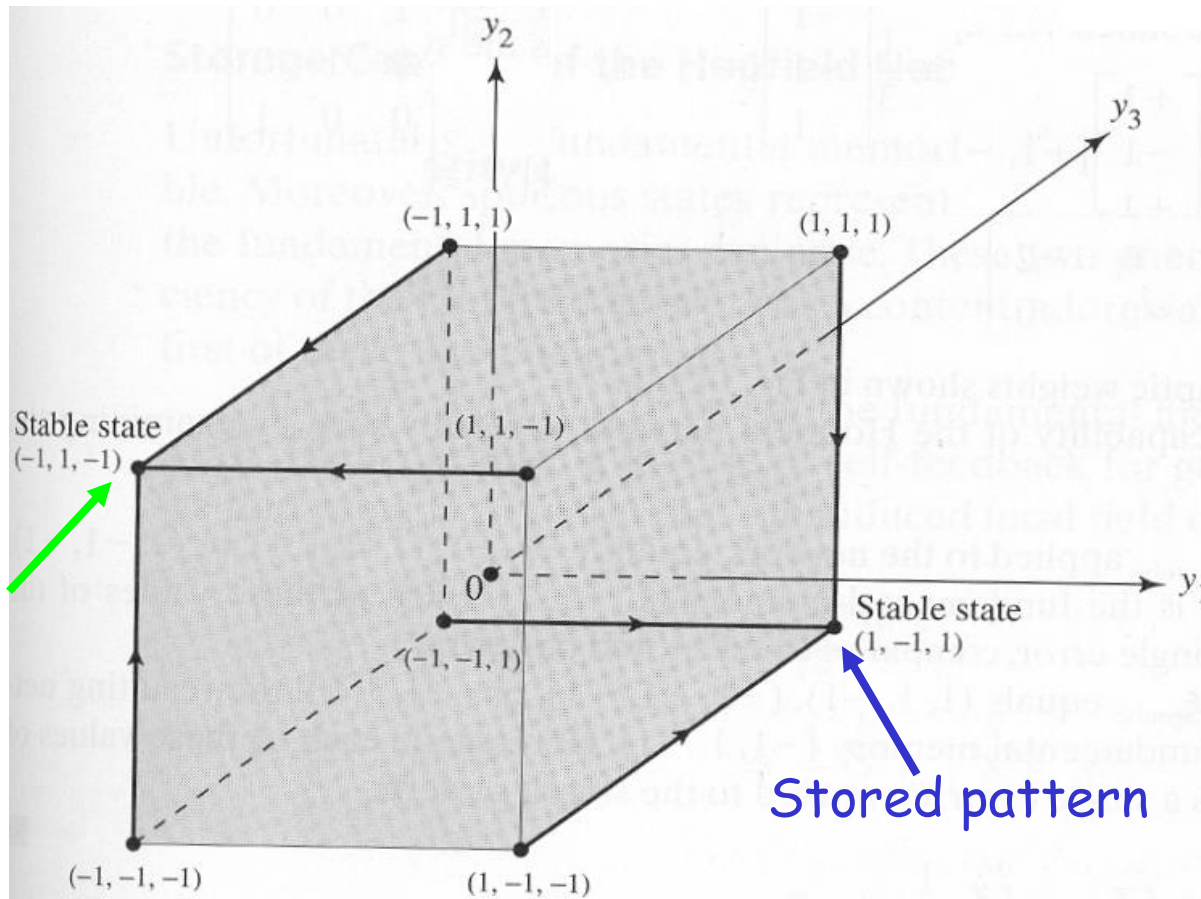
3 different examples of the net's flow

$$\frac{1}{3} \begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} 0 \\ -4 \\ 0 \end{pmatrix}$$


$$\frac{1}{3} \begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} 0 \\ 4 \\ -4 \end{pmatrix}$$


$$\frac{1}{3} \begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix} = \frac{1}{3} \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix}$$


Step_3. Iterate till convergence



Schematic diagram of all the dynamical trajectories that correspond to the designed net.

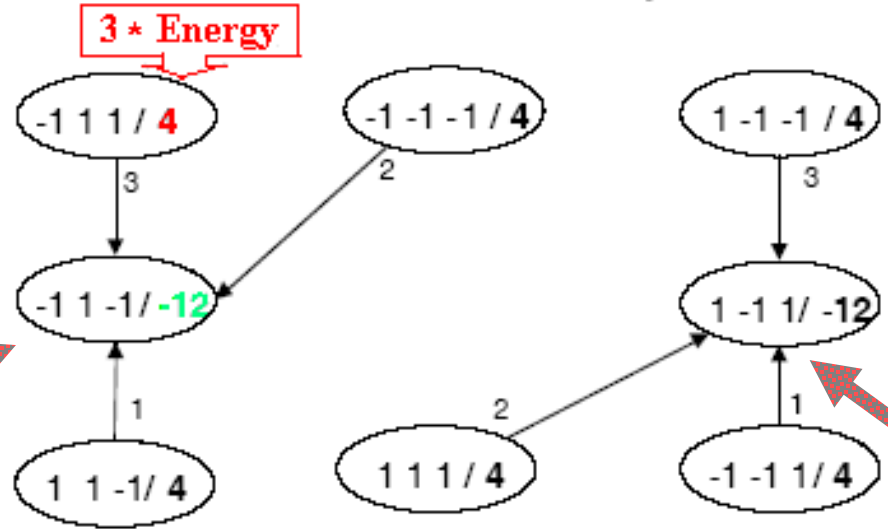
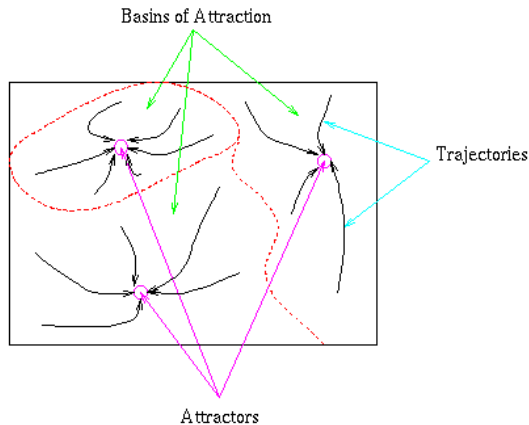
States of lowest energy correspond to attractors of Hopfield-net dynamics

$$W = \frac{1}{3} \begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$$

$$E([v_1, v_2, \dots, v_n]^T)$$

$$= -\sum_i \sum_j w_{ij} v_i v_j$$

Attractor-state



$$E_1 = \begin{pmatrix} -1 & 1 & 1 \end{pmatrix} \frac{1}{3} \begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{3} * 4$$

$$E_2 = \begin{pmatrix} -1 & 1 & -1 \end{pmatrix} \frac{1}{3} \begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} = \frac{1}{3} * -12$$

Updating in optimization

Synchronous update: All neurons are updated together. Suitable for digital implementation

$$\text{Updating rule: } V(t + 1) = \text{sign}\{WV(t) + \mathbf{I}(t) - \mathbf{T}(t)\}$$

Asynchronous update: Some neurons are updated faster than others. Not all neurons are updated simultaneously. Most natural for analog implementation.

$$\text{Updating rule: } V^o(t + 1) = \text{sign}\{WV(t) + \mathbf{I}(t) - \mathbf{T}(t)\}$$

Lyapunov function for Stability

Consider a scalar function $E(V)$ satisfying:

(i) $E(V^*) = 0$

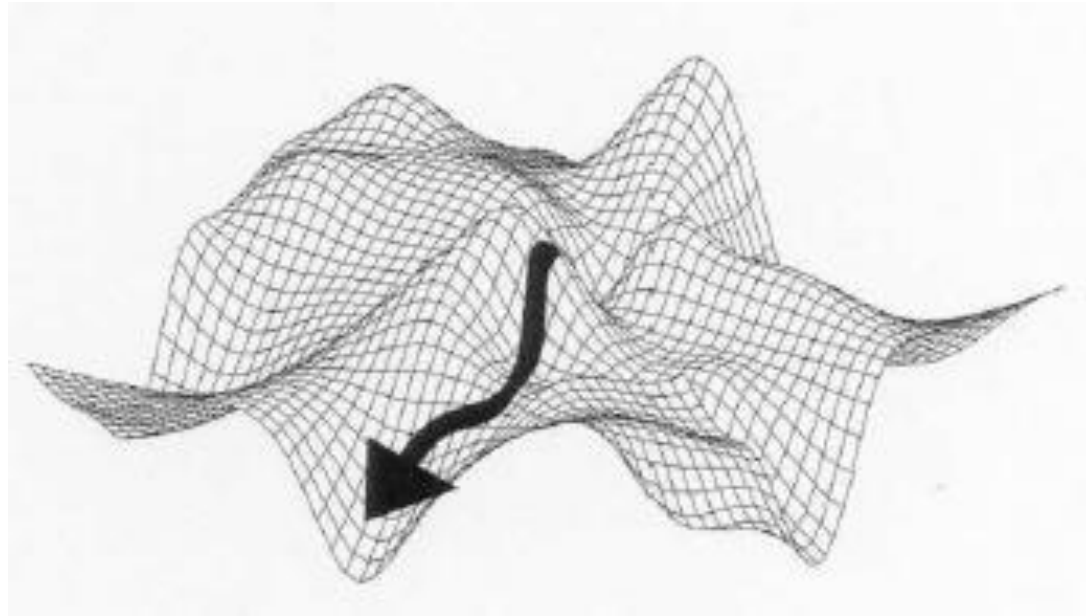
(ii) $E(V) > 0$ for $V \neq V^*$

(iii) $dE/dV = 0$ at $V = V^*$, and $dE/dV < 0$ for $V \neq V^*$

If such an $E(V)$ can be found, it is called a Lyapunov function, and the system is asymptotically stable (i.e. $V \rightarrow V^*$ as $t \rightarrow \infty$).

Hopfield Net Energy Function

The corresponding dynamical system evolves toward states of lower Energy



Hopfield Nets

- A Hopfield net is composed of binary threshold units with recurrent connections between them. Recurrent networks of non-linear units are generally very hard to analyze. They can behave in many different ways:
 - Settle to a stable state
 - Oscillate
 - Follow chaotic trajectories that cannot be predicted far into the future.
- But Hopfield realized that if the connections are symmetric, there is a global energy function
 - Each “configuration” of the network has an energy.
 - The binary threshold decision rule causes the network to settle to an energy minimum.

Running Hopfield Neural Networks

1. Define your problem and build a graph.
2. Define the weight matrix.
3. Define other constraints to build your objective function.
4. Run an optimization algorithm: the simplest could be something similar to the perceptron algorithm.

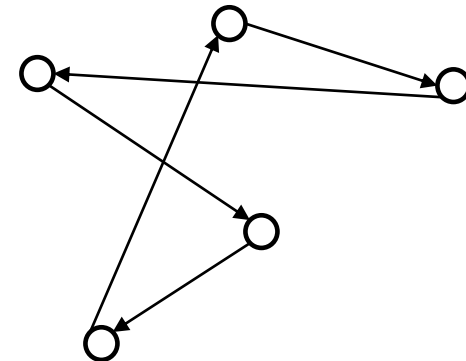
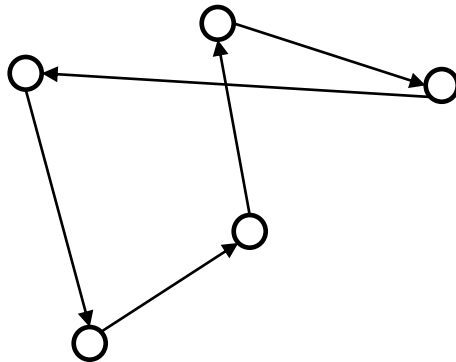
A classical example: ‘The Travelling Salesman Problem’

“Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?”

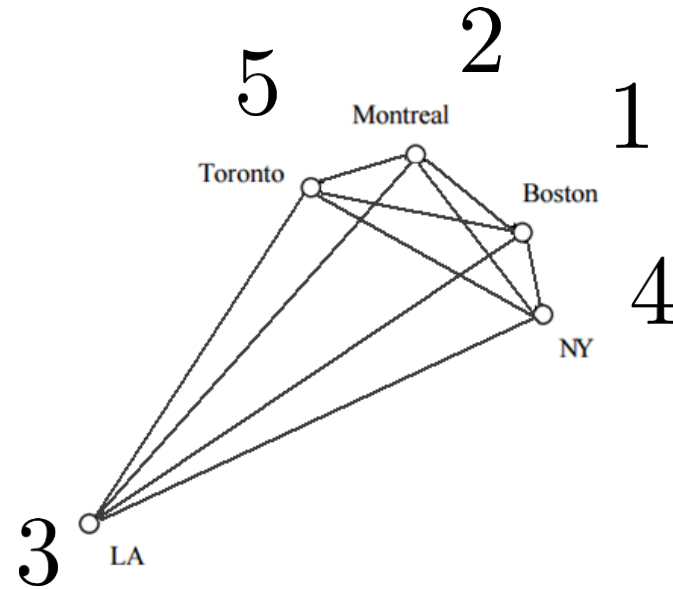


There are N cities, joining city i to city j .

The salesman wishes to find a way to visit the cities that is optimal in two way: each city is visited only once, and the total route is as short as possible.



A classical example: The Travelling Salesman Problem

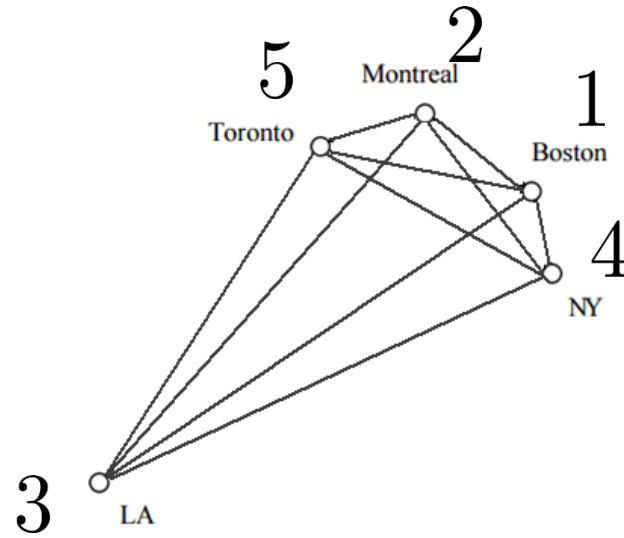


d_{ij} : the distance between city i and city j .

“Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?”

Objective: to minimizing the total distance with each city being visited only once (return to the first visited city).

A classical example: The Travelling Salesman Problem



d_{ij} : the distance between city i and city j .

$h_{ij} \in \{0, 1\}$: if the route from city i to city j has been chosen or not.

Minimize: $\sum_{ij} d_{ij} h_{ij}$

subject to: $\sum_i h_{ij} = 1, \forall j$

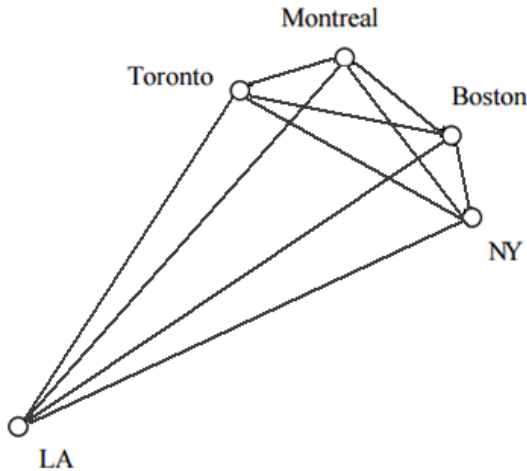
$\sum_j h_{ij} = 1, \forall i$

$h_{ij} \in \{0, 1\}$

NP-hard!

No polynomial time solution.

A classical example: ‘The Travelling Salesman Problem’



Neural network representation:

City index (i) \ Stop index (a)	1	2	3	4	5
1 (Boston)	?	?	?	?	?
2 (Montreal)	?	?	?	?	?
3 (LA)	?	?	?	?	?
4 (NY)	?	?	?	?	?
5 (Toronto)	?	?	?	?	?

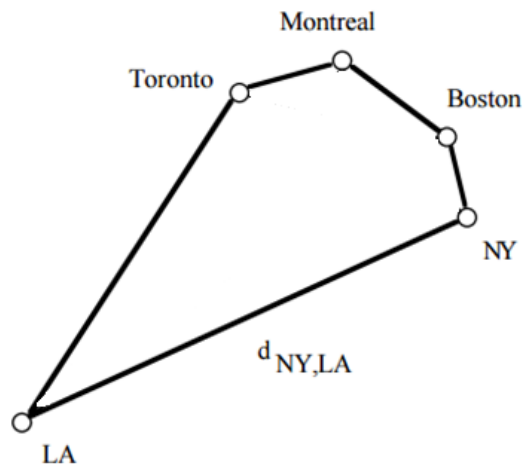
$$V_{i,a} \in \{0, 1\}, i = 1..5, a = 1..5$$



Each neuron: $V_{i,a} \in \{0, 1\}, i = 1..5, a = 1..5$,
where i indexes the city id and a indexes which the stop in the sequence.

$$\text{Total distance: } L = \frac{1}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1})$$

$$\text{Note: } V_{j0} \equiv V_{j5}$$

$$V_{i,a} \in \{0, 1\}, i = 1..5, a = 1..5$$


Stop sequence: 2(*Montreal*), 1(*Boston*), 4(*NY*), 3(*LA*), 5(*Toronto*)
Return back to the first stop (2 (Montreal) after visiting 5 (Toronto)).

Stop index (a) \ City index (i)	1	2	3	4	5
1 (Boston)	0	1	0	0	0
2 (Montreal)	1	0	0	0	0
3 (LA)	0	0	0	1	0
4 (NY)	0	0	1	0	0
5 (Tronto)	0	0	0	0	1



$$V_{i,a} \in \{0, 1\}, i = 1..5, a = 1..5$$

Each neuron: $V_{i,a} \in \{0, 1\}, i = 1..5, a = 1..5$,
where i indexes the city id and a indexes which the stop in the sequence.

Total distance: $L = \frac{1}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1})$

Note due to the return to the original city: $V_{j,0} \equiv V_{j,5}$ and $V_{j,6} \equiv V_{j,1}$.

$$L = d_{21} + d_{14} + d_{43} + d_{35} + d_{52}$$

|||

$$(L = d_{Montreal,Boston} + d_{Boston,NY} + d_{NY,LA} + d_{LA,Toronto} + d_{Toronto,Montreal})$$

A classical example: 'The Travelling Salesman Problem'

Total distance: $\sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1})$

data

Once per city: $\sum_{i,a,b}^{a \neq b} V_{ia} V_{ib}$

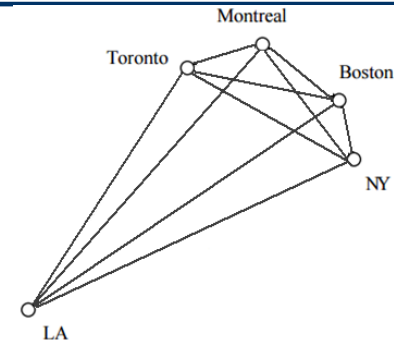
inhibitory connections on the stops (one stop each city)

One city per stop: $\sum_{i,j,a}^{i \neq j} V_{ia} V_{ja}$

inhibitory connections on the cities (one city each stop)

Five stops: $[\sum_{i,a} V_{ia} - 5]^2$

global inhibition (5 stops in total)



City index (i) \ Stop index (a)	Stop index (a)				
	1	2	3	4	5
1 (Boston)	?	?	?	?	?
2 (Montreal)	?	?	?	?	?
3 (LA)	?	?	?	?	?
4 (NY)	?	?	?	?	?
5 (Tronto)	?	?	?	?	?

$$V_{i,a} \in \{0, 1\}, i = 1..5, a = 1..5$$

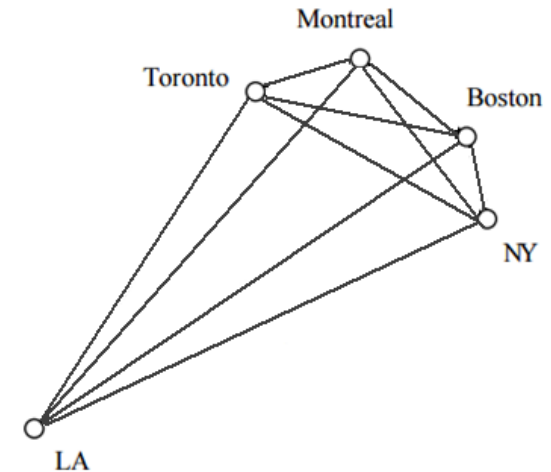
$$E(V) = \frac{D}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1}) + \frac{A}{2} \sum_{i,a,b}^{a \neq b} V_{ia} V_{ib} + \frac{B}{2} \sum_{i,j,a}^{i \neq j} V_{ia} V_{ja} + \frac{C}{2} [\sum_{i,a} V_{ia} - 5]^2$$

$$V^* = \arg \min_V E(V)$$

A classical example: ‘The Travelling Salesman Problem’

Optimization using discrete updates:

1. Initialize states matrix $V_{ia}, i = 1..5, a = 1..5$.
2. Updating



$$E(V) = \frac{D}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1}) + \frac{A}{2} \sum_{i,a,b}^{a \neq b} V_{ia} V_{ib} + \frac{B}{2} \sum_{i,j,a}^{i \neq j} V_{ia} V_{ja} + \frac{C}{2} [\sum_{i,a} V_{ia} - 5]^2$$

Stop index (a) \ City index (i)	1	2	3	4	5
1 (Boston)	?	?	?	?	?
2 (Montreal)	?	?	?	?	?
3 (LA)	?	?	?	?	?
4 (NY)	?	?	?	?	?
5 (Tronto)	?	?	?	?	?

$$V_{i,a} \in \{0, 1\}, i = 1..5, a = 1..5$$

Energy Function

$$E(V) = \frac{D}{2} \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1}) + \frac{A}{2} \sum_{i,a,b}^{a \neq b} V_{ia} V_{ib} + \frac{B}{2} \sum_{i,j,a}^{i \neq j} V_{ia} V_{ja} + \frac{C}{2} [\sum_{i,a} V_{ia} - 5]^2$$

The first term minimizes the tour distance

The next three terms makes V a permutation matrix.

$$W_{ia,jb} = -A\delta_{ij}(1 - \delta_{ab}) - B\delta_{ij}(1 - \delta_{ab}) - C - Dd_{ij}(\delta_{b,a+1} + \delta_{b,a-1})$$

$$\delta(i, j) = 1, \text{ if } i = j, 0, \text{ otherwise}$$

Updating:

$$\text{Then } V(t+1) = \text{sign}\{WV(t) + \mathbf{I}(t) - \mathbf{T}(t)\}$$

Note that the dimension of W is 25x25 here and special care needs to be taken with \mathbf{I} . One usually uses the algorithm described in the next few slides to solve the TSP problem using Hopfield networks.

Hopfield Net Energy Function

$$E(v) = -\frac{1}{2} \sum_{i \neq j} \sum_{i,j} w_{ij} v_i v_j - \sum_i I_i v_i + \sum_i \frac{1}{R_i} \int_0^{v_i} f_i^{-1}(z) dz$$

$$\nabla_v E(v) = Wv + I - u / R$$

- Hence, Hopfield net dynamic equation is to minimize $E(\mathbf{v})$ along descending gradient direction.
- Stability of Hopfield Net – If $w_{ij} = w_{ji}$ & $w_{ii} = 0$, the output will converge to a local minimum (instead of oscillating).

This slide helps you understand the general formulation of Hopfield networks, but the equations involve math in depth, which is beyond this class. It is ok for you not to get into the details.

Hopfield Network Algorithm

Hebbian Learning

1. Assign connection weights

$$w_{ij} = \begin{cases} \frac{1}{N} \sum_{s=0}^{M-1} x_i^s x_j^s & i \neq j \\ 0 & i = j, \end{cases}$$

where w_{ij} is the connection weight between node i and node j , and x_i^s is element i of the exemplar pattern s , and is either $+1$ or -1 . There are M patterns, from 0 to $M - 1$, in total.

2. Initialise with unknown pattern

$$\mu_i(0) = x_i \quad 0 \leq i \leq N - 1$$

where $\mu_i(t)$ is the output of node i at time t .

3. Iterate until convergence

$$\mu_i(t + 1) = f_h \left[\sum_{j=0}^{N-1} w_{ij} \mu_j(t) \right] \quad 0 \leq j \leq N - 1$$

The function f_h is the hard-limiting non-linearity, the step function,
Repeat the iteration until the outputs from the node
remain unchanged.

Probe pattern

Dynamical
evolution

Hopfield Neural Network

1. Define your weight matrix W (or d).
2. Define your energy function
3. Initialize states V
4. Update V for this Lyapunov energy function.

$$E = D \sum_{i,j,a} d_{ij} V_{ia} (V_{j,a+1} + V_{j,a-1}) + A \sum_{i,a,b}^{a \neq b} V_{ia} V_{ib} + B \sum_{i,j,a}^{i \neq j} V_{ia} V_{ja} + C \left[\sum_{i,a} V_{ia} - n \right]^2$$

$$\begin{aligned} du_{ia}/dt = & -u_{ia}/\tau - A \sum_{a \neq b} V_{ib} - B \sum_{j \neq i} V_{ja} \\ & -C(\sum_i \sum_n V_{ib} - n) \\ & -D \sum_j d_{ij} (V_{j,a+1} + V_{j,a-1}) \end{aligned}$$

$$V_{ia} = g(u_{ia}) = \frac{1}{2}(1 + \tanh(u_{ia}/u_0))$$

This slide helps you understand the general formulation of Hopfield networks, but the equations involve math in depth, which is beyond this class. It is ok for you not to get into the details.

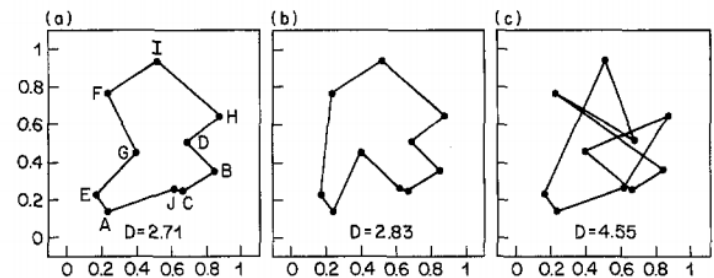


Fig. 3a-c. **a, b** Paths found by the analog convergence on 10 random cities. The example in **a** is also the shortest path. The city names **A ... J** used in Fig. 2 are indicated. **c** A typical path found using a two-state network instead of a continuous one

Summary of the Hopfield Neural Networks

1. Builds on top of the concept of brain dynamics.
2. An energy minimization approach.
3. Can be made in hardware realization easily.
4. Weak learning aspect.
5. Very limited modeling capability.
6. Useful to some tasks but no real/very little practical significance.
7. Led to the third time of “falling” of the neural network.

A bigger picture:

