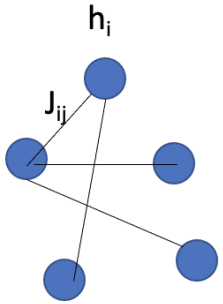
Machine Learning for Cognitive Sciences: Principles and Applications

Simona Cocco (Physics Departement of ENS)
simona.cocco@phys.ens.fr CM &TD

Vito Dichio (Physics Departement of ENS)
vito.dichio@phys.ens.fr TD

Week 10

Recall of previous lesson

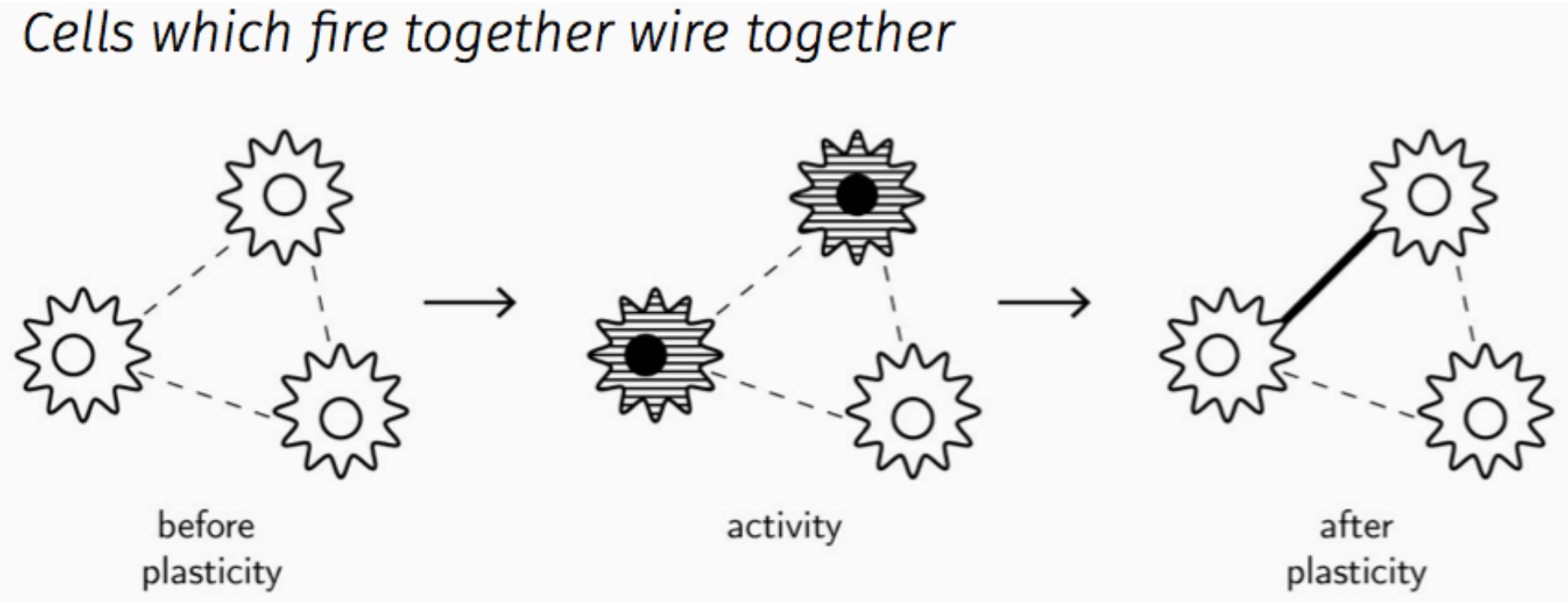


Network inference

Different approximations and heuristics to infer couplings in an efficient way : Multivariate Gaussion variables,
exact maximization of Log-Likelihood

Hopfield Model:: Model for auto-associative memory: patterns/memories are stored it the coupling matrix.

Hebbian learning through synaptic Potentiation



- Already suggested by Ramon Y Cajal in 1894: memory results from the change of couplings between neurons, rather than from the appearance of new neurons

Hebb (1949)

- Cell assemblies hypothesis (Hebb 1949): Information is represented , replayed and stored trough a group of neurons firing together

Hopfield model for auto-associative memories

One can built up a coupling matrix storing P features or 'patterns' as energy minima of the model (associative memories of the network):

Hebbe rule $$J_{ij} = \sum_{\mu=1}^{P} w_i^{\mu} \; w_j^{\mu}$$

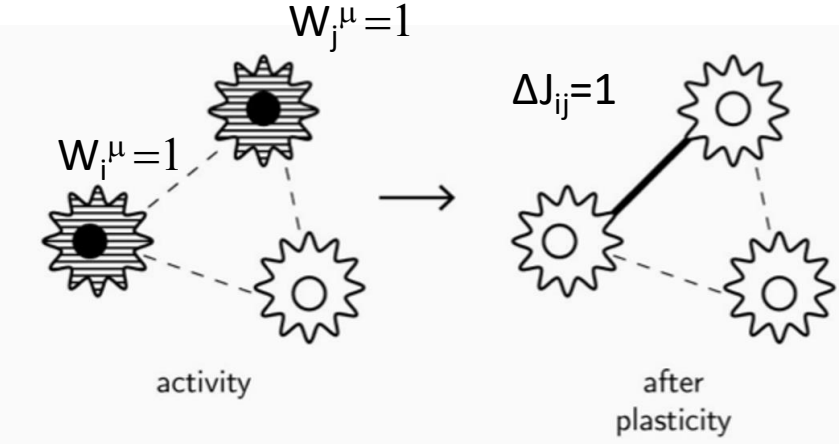
$W_i^{\mu}=[-1,1]$

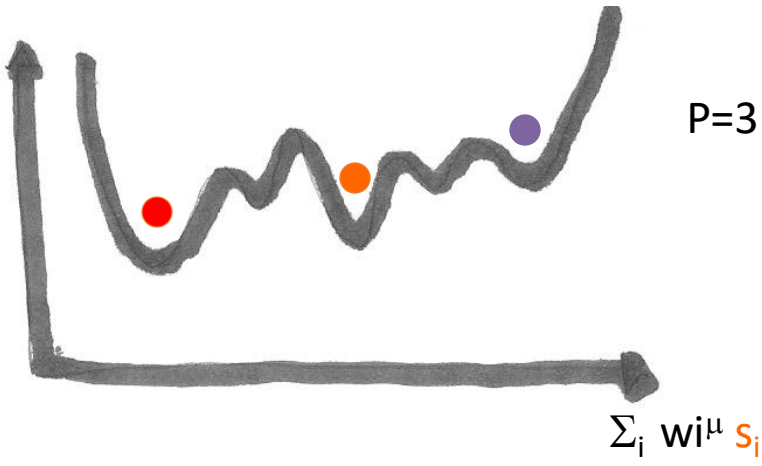
$S_i=[-1,1]$

Configurations which look like the patterns are minima of the energy

$$E(s)=-\frac{1}{2}\sum_{i<j} J_{ij}\, s_i, s_j$$

$$E(s)=-\frac{1}{2}\sum_{\mu=1}^{P}(\sum_i w_i^{\mu}\, s_i)^2$$





Hopfield model for auto-associative memories

One can built up a coupling matrix storing M features or 'patterns' as energy minima of the model (associative memories of the network):

Hebb rule $$J_{ij} = \sum_{\mu=1}^{P} w_i^{\mu} \; w_j^{\mu}$$

$W_i^{\mu}=[-1,1]$

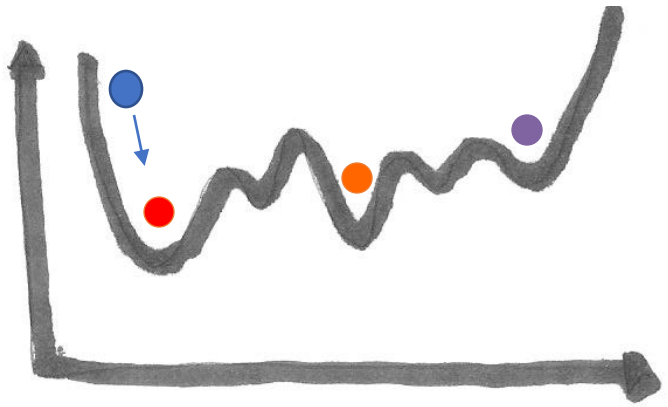
$S_i=[-1,1]$

Configurations which look like the patterns are minima of the energy:

$$E(s)=-\frac{1}{2}\sum_{i<j} J_{ij}\, s_i, s_j$$

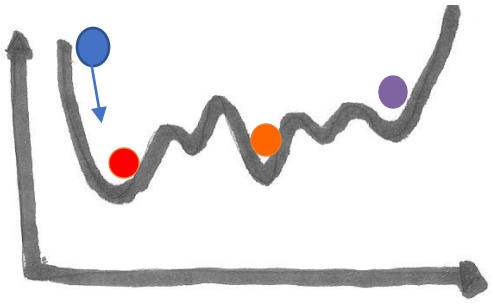
$$E(s)= -\frac{1}{2}\sum_{\mu=1}^{P}(\sum_i w_i^{\mu}\, s_i)^2$$

This networks have properties of associative Memory: dynamics of the system starting on ● fall into the nearest minima ●



Hopfield model for auto-associative memories

$$s_i^{t+1} = sign\left(\sum_{j\neq i} J_{i,j} s_j^t\right)$$



Σ_i wi^{μ} s_i

Starting from a pattern $\mathbf{s}^t = \mathbf{w}^{\mu'}$ (or nearby a pattern),
the pattern is a stable fixed point of the dynamics above if the number of stored patterns
P <L (number of neurons)

Proof:

$$s_i^{t+1} = sign\left(\sum_{\mu=1}^{P} w_i^{\mu} \sum_{j\neq i} w_j^{\mu} s_j^t\right)$$

Starting pattern

Other patterns

$$s_i^{t+1} = sign\left(w_i^{\mu'}(L-1) + \sum_{\mu\neq\mu'} \sum_{j\neq i} w_i^{\mu} w_j^{\mu} w_j^{\mu'}\right)$$

The second term is a random variables with

z: Sum of (P-1)(L-1) variables [-1,1]

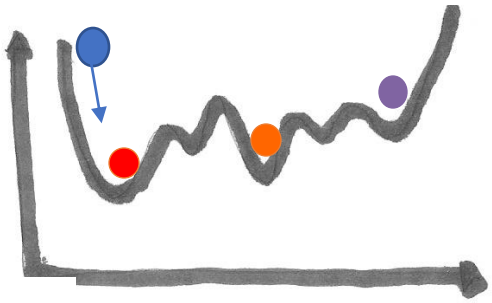
$$< z >= 0 \qquad \sigma_z^2 = (P-1)(L-1)$$

Hopfield model for auto-associative memories

Starting pattern

Other patterns

$$s_i^{t+1} = sign\left(w_i^{\mu'}(L-1) + z\right)$$



$$< z >= 0 \quad \sigma_z^2 = (P-1)(L-1)$$

Σ_i wi$^{\mu}$ s_i

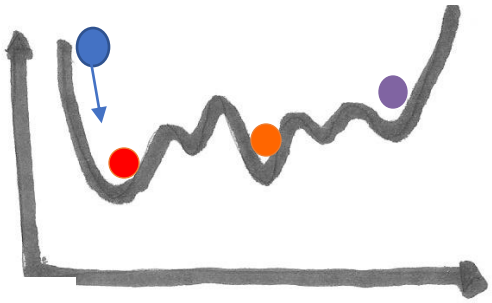
Starting from a pattern $\mathbf{s}^t = \mathbf{w}^{\mu'}$ (or nearby a pattern),
the pattern is a stable fixed point of the dynamics above if the number of stored patterns
P <L (number of neurons)

Hopfield model for auto-associative memories

Starting pattern

Other patterns

$$s_i^{t+1} = sign\left(w_i^{\mu'}(L-1) + z\right)$$



$$< z >= 0 \quad \sigma_z^2 = (P-1)(L-1)$$

Σ_i wi$^{\mu}$ s_i

Starting from a pattern $\mathbf{s}^t = \mathbf{w}^{\mu'}$ (or nearby a pattern),
the pattern is a stable fixed point of the dynamics above if the number of stored patterns
P <L (number of neurons)

Proof:

Probability of an error: $s_i^{t+1} \neq w_i^{\mu'}$

$$P_{error} \approx \int_L^{\infty} \frac{dz}{2\pi LP} e^{-z^2/(2LP)} = \int_{\sqrt{L/P}}^{\infty} \frac{dz}{2\pi} e^{-z^2/2}$$

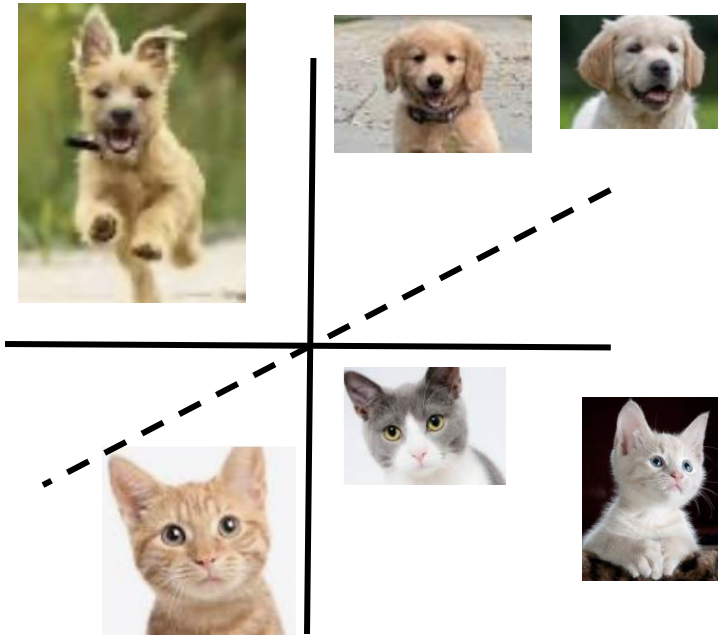
Ratio of number of synapses
(on each neuron) and pattern to be stored

Example: P_{error}=0.001 for P<0.1 L

Perceptron Algorithm

Up to know we have seen **Unsupervised** learning, where data are generic configuration of the system we want to model.

Now we will study **Supervised** learning where a label is associated to the data.



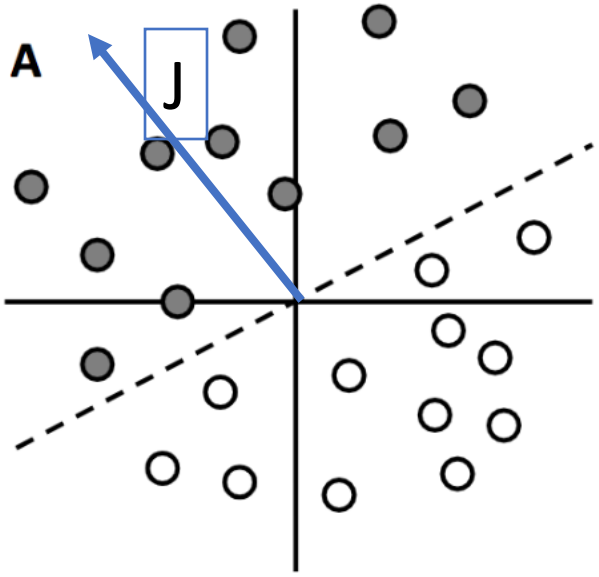
If the two classes are linearly separable

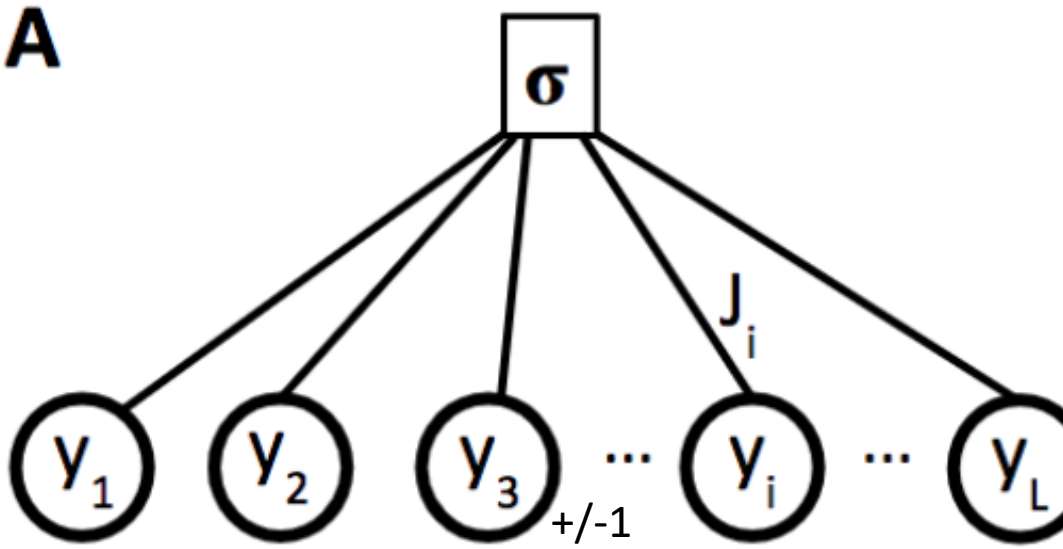
The Perceptron algorithm learn the rule to separate the class and

Learn to generalise the rule for the label.

Perceptron Algorithm

The perceptron learns to generalise the rule for the label. It is also calle Support Vector Machine.





$$\sigma = \text{sign}\left(\sum_{i=1}^{L} J_i\, y_i - \theta\right)$$

Learn a vector **J** which has a scalar product positif with a class and negative with the other

Inspired from the behavior of a real neurons

Θ is an additional term to bias the output σ.

Perceptron Algorithm

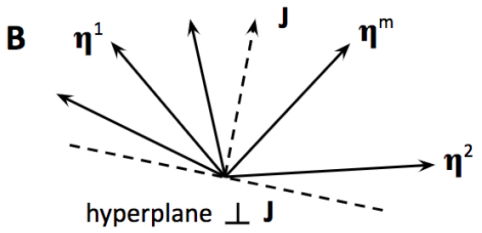
$$\sigma = \text{sign}\left(\sum_{i=1}^{L} J_i\, y_i - \theta\right)$$

To treat the bias Θ it is convenient to add a component to the weight vector $J_{L+1} = -\theta$ and to each input vector: $y_{L+1} = 1$

In a classification problem we have M examples (m=1..M) and we look for the weight vector such that these exemples are correctly classified:

$$\sigma^m = \text{sign}(\boldsymbol{J} \cdot \boldsymbol{y}^m) \ , \qquad \forall m = 1, \ldots, M \ .$$

By defining: $\boldsymbol{\eta}^m = \sigma^m \, \boldsymbol{y}^m \ .$



We obtain the condition:

$$\boldsymbol{J} \cdot \boldsymbol{\eta}^m > 0 \ , \qquad \forall m = 1, \ldots, M \ .$$

Perceptron Algorithm

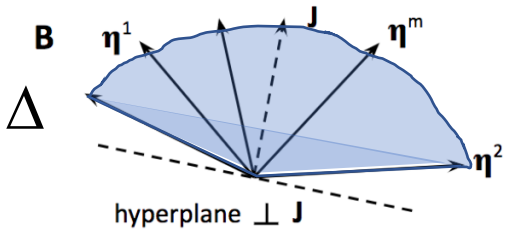
$$\sigma = \text{sign}\left(\sum_{i=1}^{L} J_i\, y_i - \theta\right)$$

To treat the bias Θ it is convenient to add a component to the weight vector $J_{L+1} = -\theta$ and to each input vector: $y_{L+1} = 1$

In a classification problem we have M examples (m=1..M) and we look for the weight vector such that these exemples are correctly classified:

$$\sigma^m = \text{sign}(\boldsymbol{J} \cdot \boldsymbol{y}^m) \ , \qquad \forall m = 1, \ldots, M \ .$$

By defining: $\boldsymbol{\eta}^m = \sigma^m \, \boldsymbol{y}^m \ .$



We obtain the condition:

$$\boldsymbol{J} \cdot \boldsymbol{\eta}^m > 0 \ , \qquad \forall m = 1, \ldots, M \ .$$

To make the perceptron-based classification more robust we write:

$$\boldsymbol{J} \cdot \boldsymbol{\eta}^m > \Delta \ , \qquad \forall m = 1, \ldots, M \ ,$$

Δ is called the stability or margin

Perceptron Algorithm

We suppose that a normalized weight vector J* satisfying the perceptron condition for any example exists. We define the stability as

$$\min_m \boldsymbol{J}^* \cdot \boldsymbol{\eta}^m = \Delta^* > 0$$

Under the normalization condition $|\boldsymbol{J}^*| = 1.$

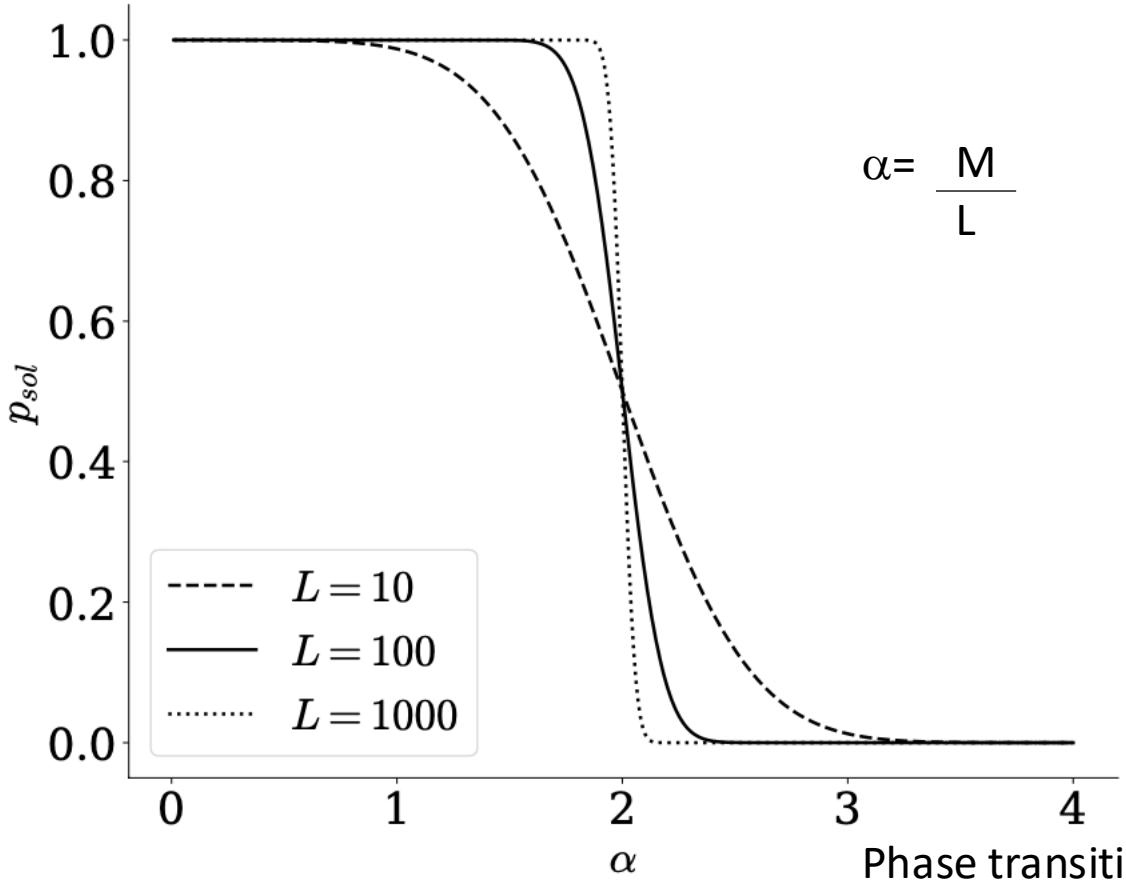
A simple iterative algorithm guaranted to output the vector J* is the following:

Perceptron learning algorithm

- Set $\boldsymbol{J}^{(0)} = \mathbf{0}$, $t = 0$, and let c be a strictly positive real number.
- Repeat as long as $\min_m \boldsymbol{J}^{(t)} \cdot \boldsymbol{\eta}^m < c$
  - ☐ Define $m_t = \operatorname{argmin}_m \boldsymbol{J}^{(t)} \cdot \boldsymbol{\eta}^m$;
  - ☐ Update $\boldsymbol{J}^{(t+1)} \leftarrow \boldsymbol{J}^{(t)} + \boldsymbol{\eta}^{m_t}$, $t \leftarrow t + 1$.
- Output $\boldsymbol{J} = \boldsymbol{J}^{(t)}/|\boldsymbol{J}^{(t)}|$.

At each step t, the algorithm looks for the least stable pattern, i.e. having the smaller scalar product with $J^{(t)}$, and pushes the weight vector in its direction at step t+1 .
It stops when the stability with all patterns is larger than c.

Critical Capacity of the Perceptron



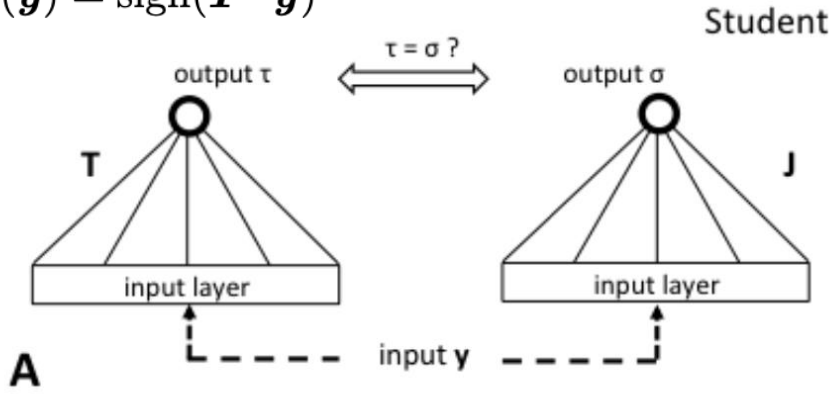
$\alpha = \frac{M}{L}$

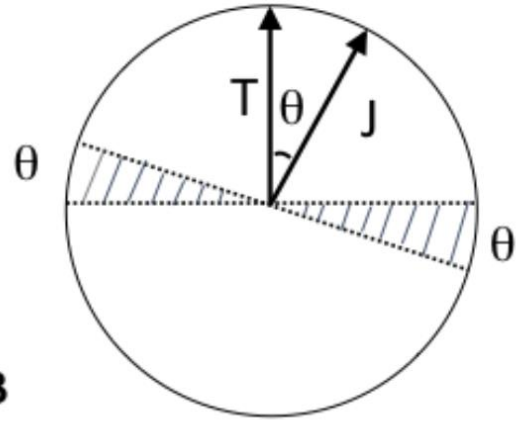
Phase transition!

$$p_{sol}(\alpha) \equiv \lim_{L\to\infty} p_{sol}(M = \alpha L, L) = \begin{cases} 1 \text{ if } \alpha < 2 \ , \\ 0 \text{ if } \alpha > 2 \ . \end{cases}$$

Generalization and Teacher Student Framework

$$\tau(\boldsymbol{y}) = \mathrm{sign}(\boldsymbol{T} \cdot \boldsymbol{y})$$





Generalization error

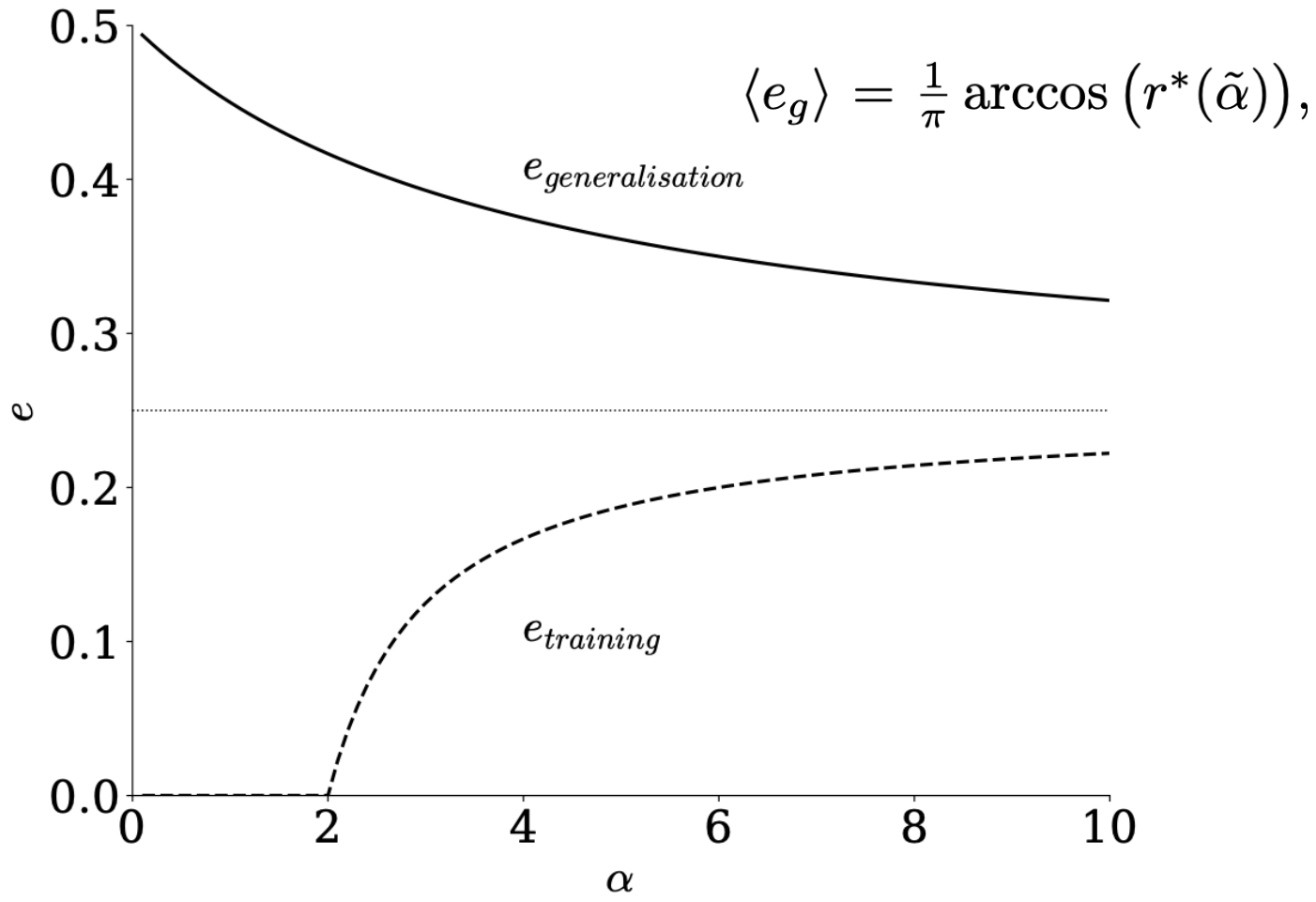
After training: test on a new example

$$e_g = (2\theta)/(2\pi)$$

$$e_g(\boldsymbol{J}|\boldsymbol{T}) = \mathrm{Proba}\big((\boldsymbol{J} \cdot \boldsymbol{y}) \times (\boldsymbol{T} \cdot \boldsymbol{y}) < 0\big)$$

Generalization error

The student weight vector J gets more and more aligned with the teacher weight vector T as the number of data increases



Take Home Message

Hopfield Model: associative memory for P<L

The perceptron as a classifier ,

Algorithm to find the weight vector if it does exists

- Algorithm to find the weight vectors in the linear case

Does Weight Vector exists?

- Critical capacity α_c=M/L=2

- Use statistical mechanics to compute generalization error as a function of α

It goes from

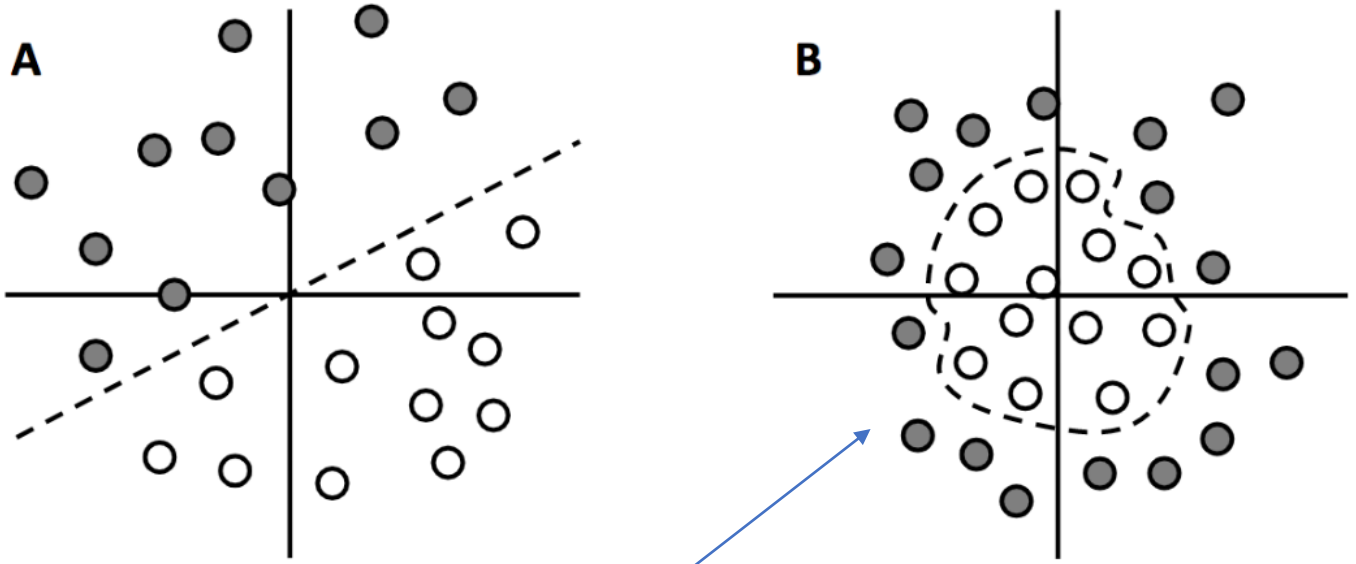
Generalization Error

Can be computed in teacher-student framework, decays as the number of examples increases

Supplementary Slides

Non linear classification and Kernel Method

The version of the perceptron we have studied above allows one to classify data points that are linearly separable



But sometimes they are not:

Non linear classification and Kernel Method

To handle non-linear classification tasks one has to replace the decision for classification:

$$\sigma = \mathrm{sign}(\boldsymbol{J} \cdot \boldsymbol{y})$$

by

$$\sigma = \mathrm{sign}\left(\widetilde{\boldsymbol{J}} \cdot \widetilde{\boldsymbol{\phi}}(\boldsymbol{y})\right)$$

Where Φ is a non-linear function.

Eg.

$$\widetilde{\boldsymbol{\phi}}(\boldsymbol{y}) = (y_1, \cdots, y_L, y_1 y_2, y_1 y_3, \cdots, y_{L-1} y_L)$$

$$\widetilde{\boldsymbol{J}} = (J_1, \cdots, J_L, J_{1,2}, J_{1,3}, \cdots, J_{L-1,L}).$$

$$\sigma = \mathrm{sign}\left(\sum_i J_i\, y_i + \sum_{i<j} J_{i,j}\, y_i\, y_j\right)$$

Going to higher dimension D ≫ L may make data linearly classifiable.