DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Deep Learning

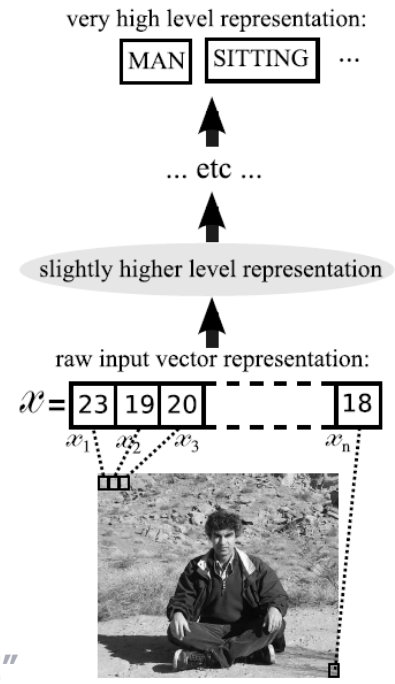
Fernando Berzal, berzal@acm.org

Deep Learning

- Neuronas estocásticas
- Redes de Hopfield
- Máquinas de Boltzmann
- Deep Belief Networks (DBNs)
- Deep Autoencoders
- Deep Stacking Networks (DSNs)

Deep Learning

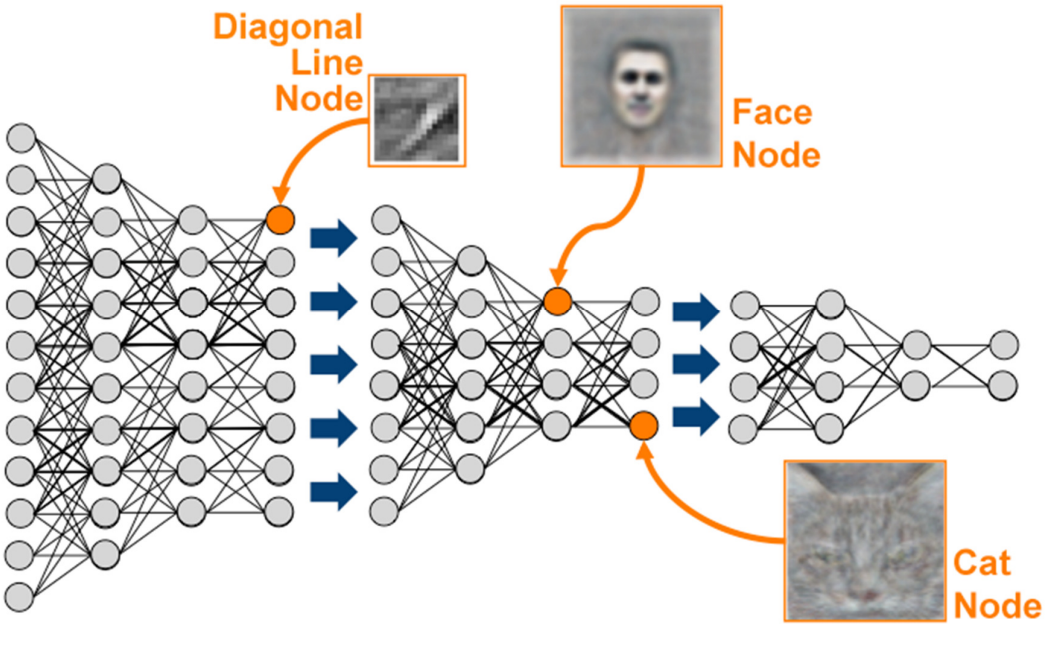
Motivación



Yoshua Bengio
"Learning Deep Architectures for AI"
2009

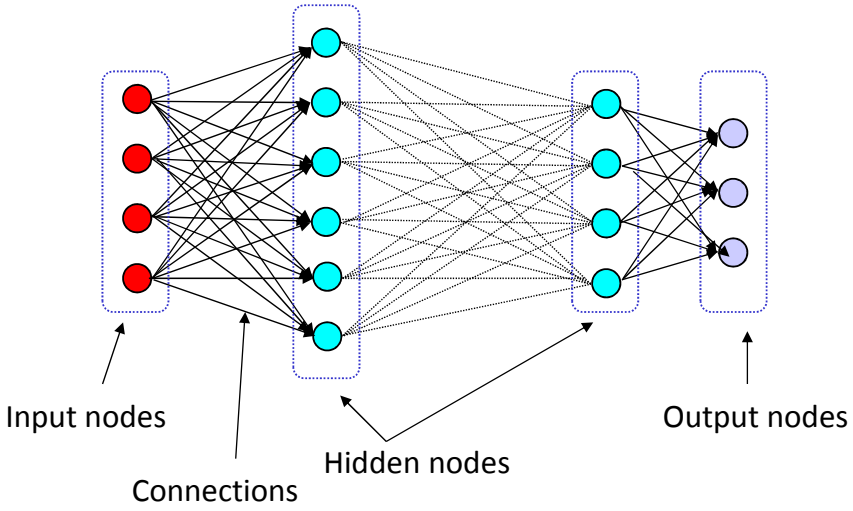
Deep Learning

Deep Learning as hierarchical feature representation



Deep Learning

Backpropagation no funciona bien
con redes que tengan varias capas ocultas
(salvo en el caso de las redes convolutivas)…



Deep Learning

Algunos hechos hicieron que backpropagation no tuviera éxito en tareas en las que luego se ha demostrado útil:

- Capacidad de cálculo limitada.
- Disponibilidad de conjuntos de datos etiquetados.
- "Deep networks" demasiado pequeñas
  (e inicializadas de forma poco razonable).

Deep Learning

2006: The Deep Breakthrough

- Hinton, Osindero & Teh « A Fast Learning Algorithm for Deep Belief Nets », *Neural Computation*, 2006
- Bengio, Lamblin, Popovici, Larochelle « Greedy Layer-Wise Training of Deep Networks », *NIPS'2006*
- Ranzato, Poultney, Chopra, LeCun « Efficient Learning of Sparse Representations with an Energy-Based Model », *NIPS'2006*

[Yoshua Bengio]

Deep Learning

Estadística	Inteligencia Artificial
Dimensionalidad baja (<100)	Dimensionalidad alta (>>100)
Mucho ruido en los datos	El ruido no es el mayor problema
Sin demasiada estructura en los datos (puede capturarse usando modelos simples)	Mucha estructura en los datos (demasiado complicada para modelos simples)
PRINCIPAL PROBLEMA	PRINCIPAL PROBLEMA
Separar estructura de ruido	Descubrir una forma de representar la estructura que se pueda aprender
TÉCNICAS	TÉCNICAS
SVM [Support Vector Machines]	Backpropagation

Deep Learning

¿Por qué las SVMs nunca fueron una buena opción en IA?

Sólo son una reencarnación de los perceptrones…

- Expanden la entrada a una capa (enorme) de características **no adaptativas**.
- Sólo tienen una capa de pesos **adaptativos**.
- Disponen de un algoritmo eficiente para ajustar los pesos controlando el sobreaprendizaje (una forma inteligente de seleccionar características y encontrar los pesos adecuados).

Deep Learning

Documento histórico

AT&T Adaptive Systems Research Dept., Bell Labs

1. Jackel bets (one fancy dinner) that by March 14, 2000 people will understand
quantitatively why big neural nets working on large databases are not so bad. (Understanding means that there will be clear conditions and bounds)

Vapnik bets (one fancy dinner) that Jackel is wrong.

But .. If Vapnik figures out the bounds and conditions, Vapnik still wins the bet.

****

2. Vapnik bets (one fancy dinner) that by March 14, 2005, no one in his right mind will use neural nets that are essentially like those used in 1995.

Jackel bets (one fancy dinner) that Vapnik is wrong

Deep Learning

¿Cuál es el problema de backpropagation?

- Requiere datos etiquetados,
  pero casi todos los datos disponibles no lo están.
- No resulta demasiado escalable:
  Demasiado lento en redes con múltiples capas ocultas.
- Se queda atascado en óptimos locales
  (lejos de ser óptimos en "deep networks").

Deep Learning

Una posibilidad

Mantener la eficiencia y la simplicidad de usar el gradiente para ajustar los pesos, pero usándolo para modelar la estructura de la entrada:

Ajustar los pesos para maximizar la probabilidad de que un modelo (generativo) genere los datos de entrada.

Maximizar p(x), no p(y|x)

Deep Learning

AI & Probabilidad

"Many ancient Greeks supported Socrates opinion that deep, inexplicable thoughts came from the gods. Today's equivalent to those gods is the erratic, even probabilistic neuron. It is more likely that increased randomness of neural behavior is the problem of the epileptic and the drunk, not the advantage of the brilliant."

P.H. Winston, "Artificial Intelligence"
(The first AI textbook, 1977)

Deep Learning

AI & Probabilidad

"All of this will lead to theories of computation which are much less rigidly of an all-or-none nature than past and present formal logic ... There are numerous indications to make us believe that this new system of formal logic will move closer to another discipline which has been little linked in the past with logic. This is thermodynamics primarily in the form it was received from Boltzmann."

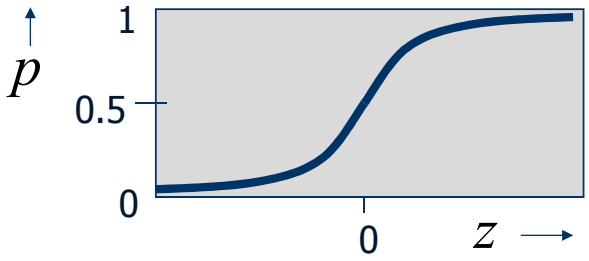
John von Neumann, "The Computer and the Brain"
(unfinished manuscript, 1958)

Neuronas estocásticas

Modelos de neuronas: Neuronas binarias estocásticas

$$z = \sum_i x_i w_i$$

$$p = \frac{1}{1+e^{-z}}$$

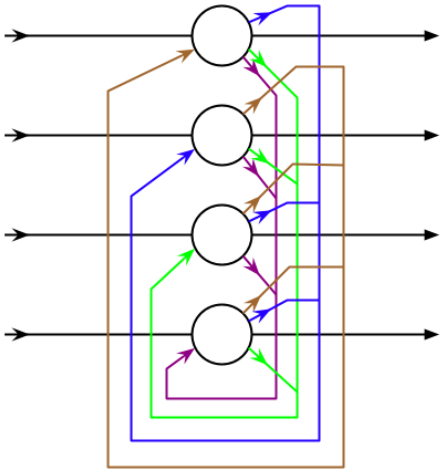


Las mismas ecuaciones que las neuronas sigmoidales, si bien su salida se interpreta como una probabilidad (de producir un spike en una pequeña ventana de tiempo)

Redes de Hopfield

Redes de Hopfield

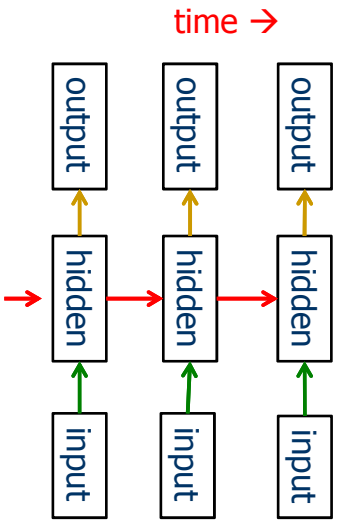
1982 Redes recurrentes
que funcionan como memorias asociativas



John J. Hopfield:
"Neural networks and physical systems
with emergent collective computational abilities"
Proceedings of the National Academy of Sciences
PNAS 79(8):2554–2558, 1982

Redes de Hopfield

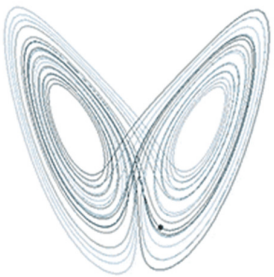
- Las redes recurrentes incluyen ciclos (como las redes neuronales biológicas).
- Las redes recurrentes tienen la capacidad de recordar.
- Son útiles para modelar secuencias (equivalen a redes multicapa con una capa por unidad de tiempo, capas que comparten los mismos pesos).



Redes de Hopfield

Sin embargo, el comportamiento dinámico de las redes recurrentes las hace difíciles de entrenar, ya que pueden

- llegar a un estado estable,
- oscilar entre varios estados,
- o comportarse de forma caótica (seguir trayectorias que no pueden predecirse a largo plazo).

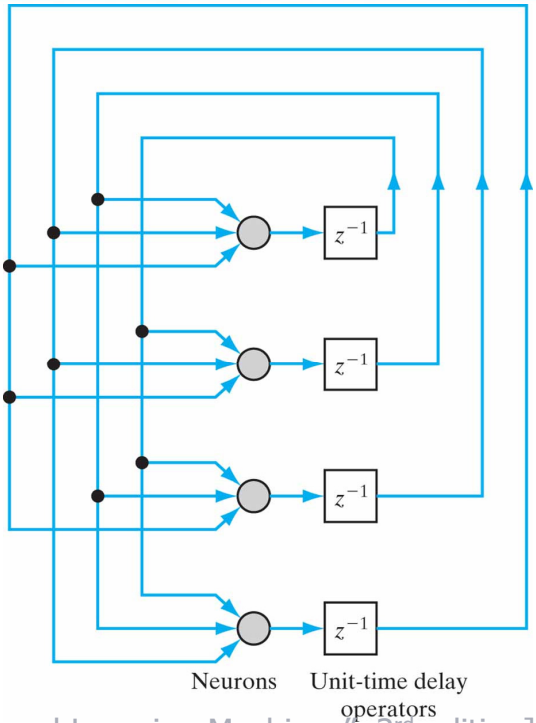


Redes de Hopfield

John Hopfield se dio cuenta de que las redes recurrentes con conexiones simétricas son más fáciles de analizar (y entrenar).

- Existe una función de energía global asociada a la red (cada configuración de la red tiene un nivel de energía).
- El comportamiento de las neuronas binarias estocásticas hace que la red tienda a alcanzar un mínimo de esa función de energía.
- Al obedecer a una función de energía, hay cosas que no pueden hacer (p.ej. modelar ciclos).

Redes de Hopfield



Red de Hopfield

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Redes de Hopfield

Función de energía

Definida como la suma de muchas contribuciones, cada una asociada al peso de una conexión y al estado de las dos neuronas conectadas:

$$E = -\sum_i s_i b_i - \sum_{i<j} s_i s_j w_{ij}$$

Esta función cuadrática de energía permite que cada neurona calcule localmente cómo afecta su estado a la energía global:

$$Energy\ gap = \Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i + \sum_j s_j w_{ij}$$

Redes de Hopfield

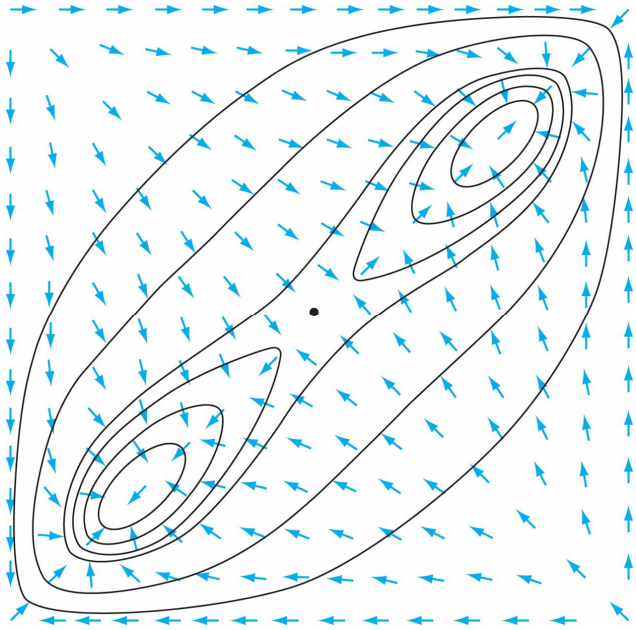
Función de energía

Para encontrar una configuración de mínima energía:

- Inicialmente, la red parte de un estado aleatorio.
- Se actualiza el estado de cada neurona, **una a una**, en un orden aleatorio (usando neuronas binarias estocásticas, se establece el estado que lleve a la red a una configuración de mejor energía).

  NOTA: Si las actualizaciones fuesen simultáneas, la energía de la red podría aumentar (no paralelizable).

Redes de Hopfield



Mapa de energía

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Redes de Hopfield

Memorias asociativas

- Hopfield propuso que la memoria podría corresponder a los mínimos de energía de una red.
- La regla de actualización de las neuronas binarias estocásticas puede servir para "limpiar" una memoria incompleta o corrupta.
- El uso de mínimos de energía proporciona memorias direccionables por su contenido en las que un elemento puede recuperarse a partir de parte de su contenido (a.k.a. memorias asociativas).

Redes de Hopfield

Memorias asociativas

Si usamos +1 y -1 como actividades de las neuronas binarias estocásticas, podemos almacenar un vector de estado binario incrementando el peso de la conexión entre dos unidades por el producto de sus actividades:

$$\Delta w_{ij} = s_i s_j$$

Si usamos 0 y 1, la regla es algo más complicada:

$$\Delta w_{ij} = 4\,(s_i - 1/2)(s_j - 1/2)$$

Redes de Hopfield

TABLE 13.2 Summary of the Hopfield Model

1. *Learning.* Let $\xi_1, \xi_2, \ldots, \xi_\mu$ denote a known set of N-dimensional fundamental memories. Use the outer-product rule (i.e., Hebb's postulate of learning) to compute the synaptic weights of the network as

$$w_{ji} = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{\mu=1}^{M} \xi_{\mu,j}\,\xi_{\mu,i}, & j \neq i \\ 0, & j = i \end{cases}$$

where w_{ji} is the synaptic weight from neuron i to neuron j. The elements of the vector ξ_μ equal ± 1. Once they are computed, the synaptic weights are kept fixed.

2. *Initialization.* Let ξ_{probe} denote an unknown N-dimensional input vector (probe) presented to the network. The algorithm is initialized by setting

$$x_j(0) = \xi_{j,\,\text{probe}}, \qquad j = 1, \ldots, N$$

where $x_j(0)$ is the state of neuron j at time $n = 0$ and $\xi_{j,\text{probe}}$ is the jth element of the probe ξ_{probe}.

3. *Iteration Until Convergence.* Update the elements of state vector $\mathbf{x}(n)$ asynchronously (i.e., randomly and one at a time) according to the rule

$$x_j(n+1) = \text{sgn}\left(\sum_{i=1}^{N} w_{ji}\,x_i(n)\right), \qquad j = 1, 2, \ldots, N$$

Repeat the iteration until the state vector $\mathbf{x}$ remains unchanged.

4. *Outputting.* Let $\mathbf{x}_{\text{fixed}}$ denote the fixed point (stable state) computed at the end of step 3. The resulting output vector $\mathbf{y}$ of the network is

$$\mathbf{y} = \mathbf{x}_{\text{fixed}}$$

Step 1 is the storage phase, and steps 2 through 4 constitute the retrieval phase.

Aprendizaje

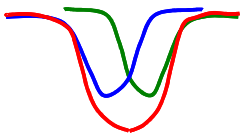
[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Redes de Hopfield

Memorias asociativas

- La capacidad de almacenamiento de una red de Hopfield completamente conectada con N unidades es sólo de M=0.15N "recuerdos": $0.15N^2$ bits de memoria requieren N^2 log (2M+1) bits para los pesos.

- Cada vez que memorizamos una configuración, pretendemos crear un mínimo de energía, pero dos mínimos cercanos pueden interferir, lo que limita la capacidad de la red de Hopfield.

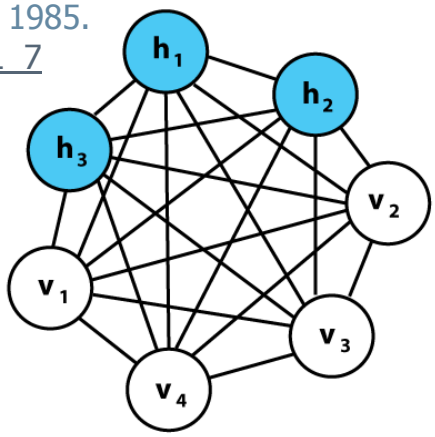


Máquinas de Boltzmann

Máquinas de Boltzmann

1985 Máquinas de Boltzmann
(redes de Hopfield con neuronas ocultas)

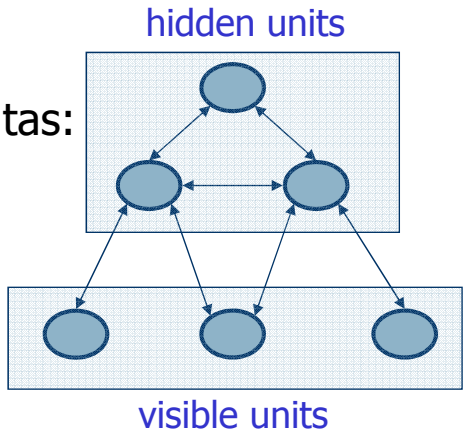
David H. Ackley, Geoffrey E. Hinton & Terrence J. Sejnowski:
"A Learning Algorithm for Boltzmann Machines"
Cognitive Science 9(1):147–169, 1985.
DOI 10.1207/s15516709cog0901_7



Máquinas de Boltzmann

Las máquinas de Boltzmann son redes de Hopfield con neuronas ocultas:

- Más "potentes" que las redes de Hopfield.
- Menos "potentes" que las redes recurrentes.



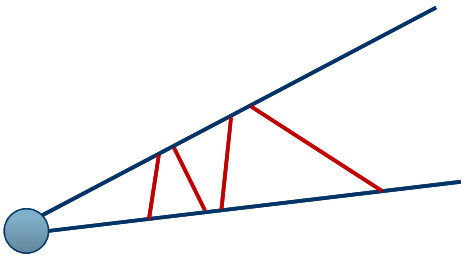
En vez de utilizar las redes para almacenar recuerdos, las utilizamos para construir interpretaciones de las entradas.

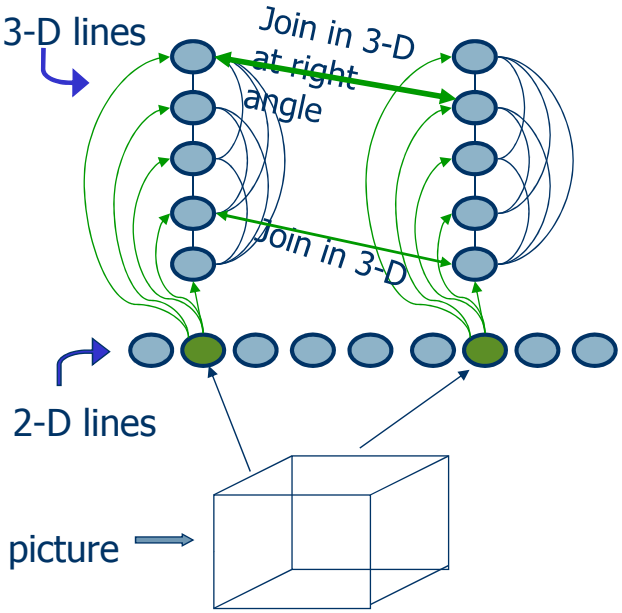
Tienen un algoritmo de aprendizaje sencillo y elegante...

Máquinas de Boltzmann

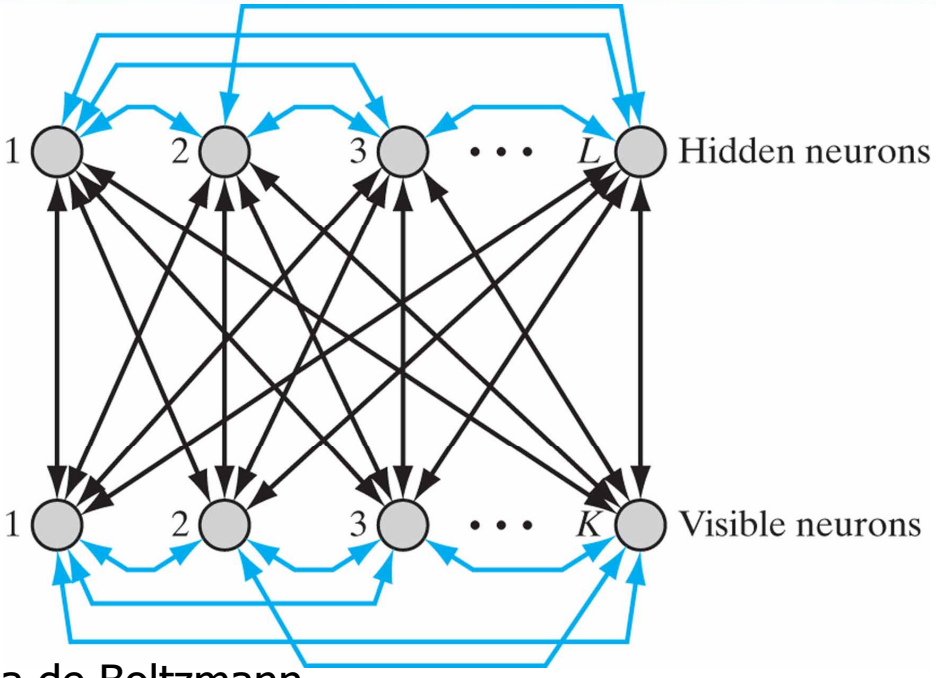
EJEMPLO [Hinton]

Aristas 3D a partir de imágenes 2D





Máquinas de Boltzmann



Máquina de Boltzmann

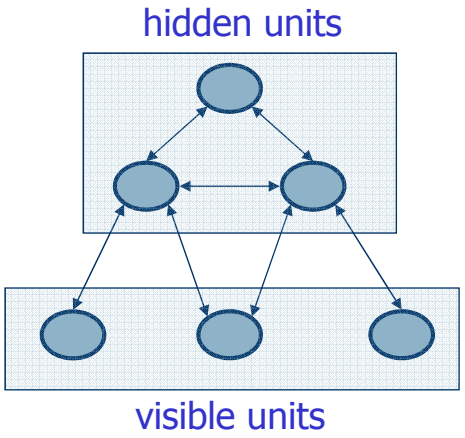
[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Máquinas de Boltzmann

Problemas computacionales

- Búsqueda del mínimo de la función de energía
  (los mínimos locales representan interpretaciones subóptimas).

  e.g. Enfriamiento simulado



- Aprendizaje:
  Cómo establecer los pesos de las conexiones con las unidades ocultas (y entre las neuronas ocultas).

Máquinas de Boltzmann

Equilibrio térmico

- El equilibro térmico de una máquina de Boltzmann se alcanza cuando se estabiliza la distribución de probabilidad de sus estados (no tiene por qué tratarse de un único estado/configuración de mínima energía).

- Dado un conjunto de entrenamiento de vectores binarios, ajustamos un modelo que asigna una probabilidad a cada vector binario posible:

$$p(Model\ \ i \mid data) = \frac{p(data \mid Model\ \ i)}{\sum_j p(data \mid Model\ \ j)}$$

Máquinas de Boltzmann

La energía de una configuración
está relacionada con su probabilidad:

- Simplemente, definimos la probabilidad como

$$p(\mathbf{v},\mathbf{h}) \propto e^{-E(\mathbf{v},\mathbf{h})}$$

- La probabilidad de una configuración será, por tanto,

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u},\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

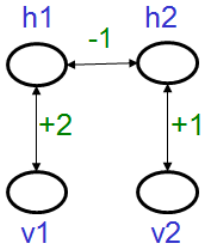
Función de partición

Máquinas de Boltzmann

La probabilidad de una configuración determinada para las neuronas visibles será la suma de las probabilidades de las configuraciones conjuntas que la contienen:

v	h	$-E$	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1 1	1 1	2	7.39	.186	0.466
1 1	1 0	2	7.39	.186	
1 1	0 1	1	2.72	.069	
1 1	0 0	0	1	.025	
1 0	1 1	1	2.72	.069	0.305
1 0	1 0	2	7.39	.186	
1 0	0 1	0	1	.025	
1 0	0 0	0	1	.025	
0 1	1 1	0	1	.025	0.144
0 1	1 0	0	1	.025	
0 1	0 1	1	2.72	.069	
0 1	0 0	0	1	.025	
0 0	1 1	-1	0.37	.009	0.084
0 0	1 0	0	1	.025	
0 0	0 1	0	1	.025	
0 0	0 0	0	1	.025	
			39.70		

$$p(\mathbf{v})=\frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u},\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$



Máquinas de Boltzmann

$$p(\mathbf{v},\mathbf{h}) \propto e^{-E(\mathbf{v},\mathbf{h})}$$

- Si existen muchas neuronas ocultas, no podemos calcular el término de normalización (la función de partición incluye un número exponencial de términos).
- Usamos MCMC [Markov Chain Monte Carlo], comenzando de una configuración aleatoria, hasta que alcance una distribución estacionaria (equilibrio térmico) para tomar muestras del modelo.

Máquinas de Boltzmann

Algoritmo de aprendizaje

OBJETIVO

Maximizar el producto de las probabilidades que la máquina de Boltzmann les asigna a los vectores del conjunto de entrenamiento (o, de forma equivalente, la suma de sus logaritmos).

Es lo mismo que maximizar la probabilidad de que obtengamos los N casos del conjunto de entrenamiento si dejamos que la red llegue a su distribución estacionaria N veces sin entradas externas y muestreamos el estado de sus unidades visibles.

Máquinas de Boltzmann

Algoritmo de aprendizaje

Todo lo que un peso debe conocer
está contenido en la diferencia entre dos correlaciones:

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{v}} - \langle s_i s_j \rangle_{model}$$

Valor esperado del producto de los estados en equilibrio térmico cuando se fija el estado de las unidades visibles

Valor esperado del producto de los estados en equilibrio térmico (sin fijar el estado de las unidades visibles)

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model}$$

Máquinas de Boltzmann

Algoritmo de aprendizaje

¿Por qué?

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model}$$

- La probabilidad de una configuración global en equilibrio térmico es una función exponencial de su energía (el equilibrio hace que el logaritmo de las probabilidades sea una función lineal de la energía).

$$-\frac{\partial E}{\partial w_{ij}} = s_i s_j$$

- El proceso de alcanzar el equilibrio se encarga propagar información acerca de los pesos, por lo que no necesitamos backpropagation.

Máquinas de Boltzmann

Algoritmo de aprendizaje

¿Por qué?

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model}$$

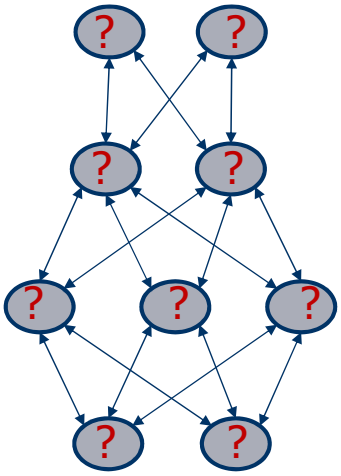
$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u}} \sum_{\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}}$$

La parte positiva encuentra configuraciones ocultas que funcionan bien con v (y baja su energía)

La parte negativa encuentra configuraciones conjuntas que mejor compiten (y sube su energía)

Máquinas de Boltzmann

- En una máquina de Boltzmann, las actualizaciones estocásticas de las distintas unidades deben ser secuenciales.



- Existe una arquitectura que admite actualizaciones paralelas alternas mucho más eficientes: **DBM [Deep Boltzmann Machine]**
  - Sin conexiones entre unidades de una misma capa.
  - Sin conexiones entre capas no adyacentes.

Máquinas de Boltzmann

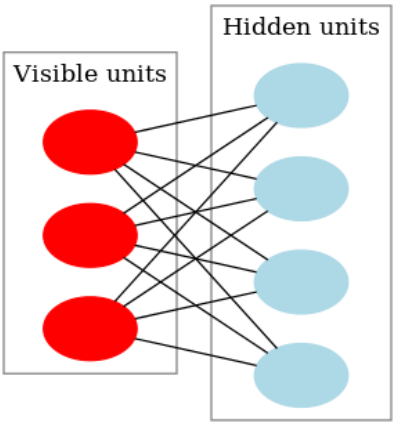
DBM [Deep Boltzmann Machine]

MNIST: Datos reales & muestras del modelo aprendido

Máquinas de Boltzmann

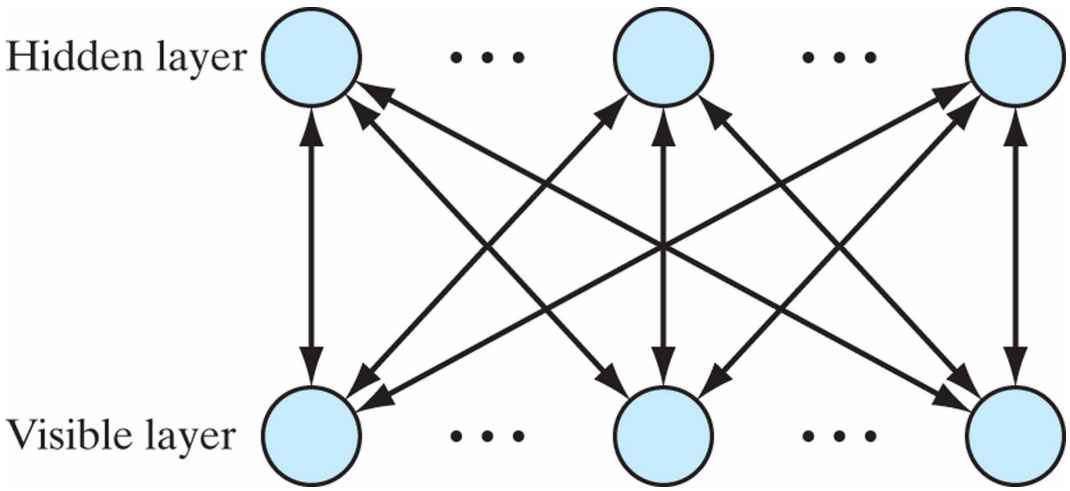
Máquinas de Boltzmann restringidas

1986 Harmonium = Restricted Boltzmann Machines
(máquinas de Boltzmann con estructura fija: grafos bipartidos con una capa de neuronas ocultas y una capa de neuronas "visibles", sin conexiones entre las neuronas de la misma capa)



Paul Smolensky: "Information Processing in Dynamical Systems: Foundations of Harmony Theory". In David E. Rumelhart & James L. McLelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. MIT Press, chapter 6, pp. 194-281. ISBN 0-262-68053-X.

Máquinas de Boltzmann



RBM: Máquina de Boltzmann restringida

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Máquinas de Boltzmann

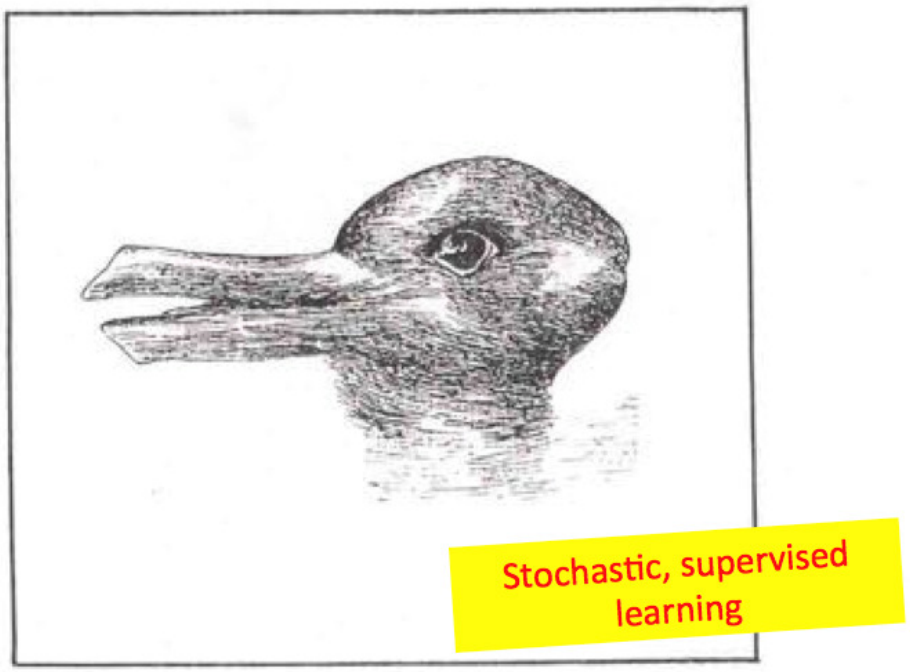
Máquinas de Boltzmann restringidas [RBMs]

Tipo particular de Markov Random Field [MRF] con una capa de neuronas estocásticas ocultas y una capa de neuronas estocásticas visibles u observables.

NOTA

Un campo aleatorio de Markov [MRF] es un modelo estocástico que representa una distribución conjunta de probabilidades mediante un grafo en el que cada nodo representa una variable aleatoria y cada arista una dependencia entre las variables que conecta.

Máquinas de Boltzmann



Máquinas de Boltzmann

Máquinas de Boltzmann restringidas [RBMs]

- La distribución sobre las unidades visibles (v) y ocultas (h) dados los parámetros del modelo (θ) se define en términos de una función de energía E:

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{Z}$$

donde Z es un factor de normalización o función de partición: $Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))$

- La probabilidad marginal que el modelo le asigna a un vector visible v es: $p(\mathbf{v}; \theta) = \frac{\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{Z}$

Máquinas de Boltzmann

Máquinas de Boltzmann restringidas [RBMs]

Función de energía

- Para RBMs Bernoulli (visible) – Bernoulli (oculta):

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{I} \sum_{j=1}^{J} w_{ij} v_i h_j - \sum_{i=1}^{I} b_i v_i - \sum_{j=1}^{J} a_j h_j$$

- Para RBMs Gaussian (visible) – Bernoulli (oculta):

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{I} \sum_{j=1}^{J} w_{ij} v_i h_j - \frac{1}{2} \sum_{i=1}^{I} (v_i - b_i)^2 - \sum_{j=1}^{J} a_j h_j$$

Máquinas de Boltzmann

Bernoulli-Bernoulli RBM

Neuronas estocásticas binarias

$$p(h_j = 1|\mathbf{v};\theta) = \sigma\left(\sum_{i=1}^{I} w_{ij}v_i + a_j\right)$$

$$p(v_i = 1|\mathbf{h};\theta) = \sigma\left(\sum_{j=1}^{J} w_{ij}h_j + b_i\right)$$

$$\sigma(x) = 1/(1+\exp(-x))$$

Máquinas de Boltzmann

Bernoulli-Gaussian RBM

Neuronas binarias en la capa oculta,
valores reales en la capa visible.

$$p(h_j = 1|\mathbf{v};\theta) = \sigma\left(\sum_{i=1}^{I} w_{ij}v_i + a_j\right),$$

$$p(v_i|\mathbf{h};\theta) = N\left(\sum_{j=1}^{J} w_{ij}h_j + b_i, 1\right),$$

Máquinas de Boltzmann

Máquinas de Boltzmann restringidas [RBMs]

Calculando el gradiente del logaritmo de la función de verosimilitud [log-likelihood], se puede derivar la regla de actualización de los pesos de una RBM:

$$\Delta w_{ij} = E_{\text{data}}(v_i h_j) - E_{\text{model}}(v_i h_j)$$

E_{data} es el valor esperado observado en el conjunto de entrenamiento (muestreando h_j dados los v_i de acuerdo con el modelo) y E_{model} es el valor esperado bajo la distribución definida por el modelo.

Máquinas de Boltzmann

Máquinas de Boltzmann restringidas [RBMs]

Al restringir la topología de la red,
facilitamos el aprendizaje:

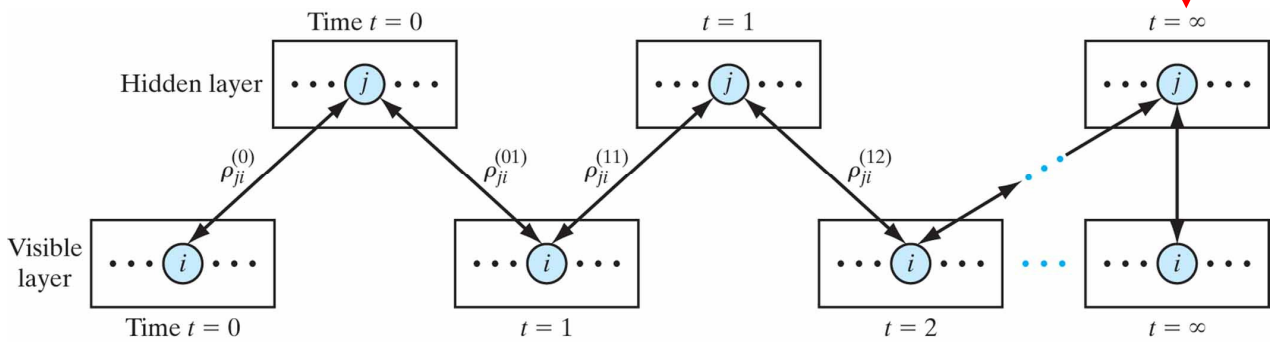
- En una RBM, se alcanza el equilibrio en un solo paso cuando se fija el valor de las unidades visibles:
  el cálculo del valor exacto de $E_{data}=\langle v_i h_j \rangle_{\mathbf{v}}$ es directo.

- Por desgracia, el cálculo de E_{model} es intratable…

Máquinas de Boltzmann

$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty)$$

Una fantasía



$\langle v_i h_j \rangle^0$ $\langle v_i h_j \rangle^\infty$

Muestreo de Gibbs @ RBM

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Máquinas de Boltzmann

Comenzamos con el estado de las neuronas visibles v_i:

$$\mathbf{h}_i \sim p(\mathbf{h}|\mathbf{v}_i, \boldsymbol{\theta})$$
$$\mathbf{v}'_i \sim p(\mathbf{v}|\mathbf{h}_i, \boldsymbol{\theta})$$
$$\mathbf{h}'_i \sim p(\mathbf{h}|\mathbf{v}'_i, \boldsymbol{\theta})$$

Podemos interpretar v_i' como el intento de reconstruir los datos originales v_i tras haberlos codificado en h_i (y volverlos a decodificar).

Máquinas de Boltzmann

Máquinas de Boltzmann restringidas [RBMs]

$$\Delta w_{ij} = E_{\text{data}}(v_i h_j) - E_{\text{model}}(v_i h_j)$$

La "divergencia contrastiva" [contrastive divergence] fue el primer método eficiente propuesto para aproximar E_{model}, consistente en ejecutar sólo algunos pasos del algoritmo de muestreo de Gibbs:

Initialize $\mathbf{v_0}$ at data

Sample $\mathbf{h_0} \sim \mathbf{p}(\mathbf{h}|\mathbf{v_0})$

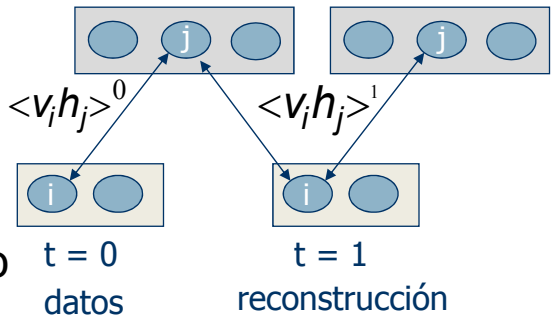
Sample $\mathbf{v_1} \sim \mathbf{p}(\mathbf{v}|\mathbf{h_0})$

Sample $\mathbf{h_1} \sim \mathbf{p}(\mathbf{h}|\mathbf{v_1})$

Máquinas de Boltzmann

Divergencia contrastiva

$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1)$$



- Comenzamos con un vector del conjunto de entrenamiento sobre las unidades visibles.
- Actualizamos todas las unidades ocultas en paralelo.
- Actualizamos todas las unidades visibles en paralelo para obtener una reconstrucción de los datos.
- Volvemos a actualizar las unidades ocultas de nuevo.

No estamos siguiendo el gradiente del log-likelihood, pero funciona...

Máquinas de Boltzmann

Máquinas de Boltzmann restringidas [RBMs]

$$\Delta w_{ij} = E_{\text{data}}(v_i h_j) - E_{\text{model}}(v_i h_j)$$

Si usamos (v_1,h_1) para aproximar $E_{model}(v_i h_j)$ obtenemos el **algoritmo CD-1**.

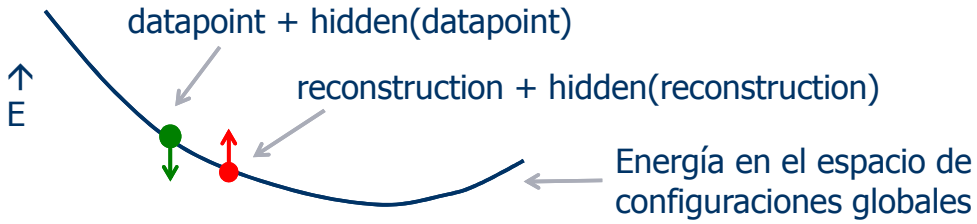
Si ejecutamos más pasos de la cadena de Markov hasta obtener (v_k,h_k) tenemos el **algoritmo CD-k**.

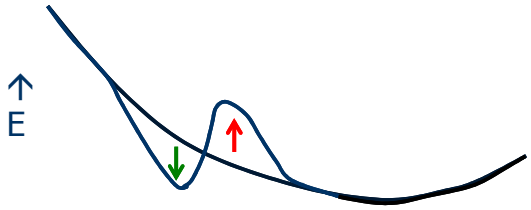
Existen mejores técnicas para estimar el gradiente de las RBMs, como la verosimiliud máxima estocástica o divergencia contrastiva persistente [**PCD**]

Máquinas de Boltzmann

Divergencia contrastiva

$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1)$$





Cambio en los pesos para modificar la energía en el punto correspondiente a los datos.

Cambio en los pesos para aumentar la energía en el punto correspondiente a la reconstrucción.

Máquinas de Boltzmann

Geoff Hinton doesn't need to make hidden units.
They hide by themselves when he approaches.

Geoff Hinton doesn't disagree with you,
he contrastively diverges

Deep Belief Nets actually
believe deeply in Geoff Hinton.

Yann LeCun: "Geoff Hinton facts"
http://yann.lecun.com/ex/fun/index.html

Máquinas de Boltzmann

Algorithm 27.3: CD-1 training for an RBM with binary hidden and visible units

Initialize weights $\mathbf{W} \in \mathbb{R}^{R \times K}$ randomly;
$t := 0$;
for *each epoch* **do**
  $t := t + 1$;
  for *each minibatch of size* B **do**
    Set minibatch gradient to zero, $\mathbf{g} := \mathbf{0}$;
    for *each case* $\mathbf{v}_i$ *in the minibatch* **do**
      Compute $\boldsymbol{\mu}_i = \mathbb{E}[\mathbf{h}|\mathbf{v}_i, \mathbf{W}]$;
      Sample $\mathbf{h}_i \sim p(\mathbf{h}|\mathbf{v}_i, \mathbf{W})$;
      Sample $\mathbf{v}'_i \sim p(\mathbf{v}|\mathbf{h}_i, \mathbf{W})$;
      Compute $\boldsymbol{\mu}'_i = \mathbb{E}[\mathbf{h}|\mathbf{v}'_i, \mathbf{W}]$;
      Compute gradient $\nabla_{\mathbf{W}} = (\mathbf{v}_i)(\boldsymbol{\mu}_i)^T - (\mathbf{v}'_i)(\boldsymbol{\mu}'_i)^T$;
      Accumulate $\mathbf{g} := \mathbf{g} + \nabla_{\mathbf{W}}$;
    Update parameters $\mathbf{W} := \mathbf{W} + (\alpha_t / B)\mathbf{g}$

CD-1: Contrastive Divergence

[Murphy: "Machine Learning: A Probabilistic Perspective", 2012]

Máquinas de Boltzmann

Algorithm 1

RBMupdate($\mathbf{x}_1, \epsilon, W, \mathbf{b}, \mathbf{c}$)

This is the RBM update procedure for binomial units. It can easily adapted to other types of units.

$\mathbf{x}_1$ is a sample from the training distribution for the RBM

ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence

W is the RBM weight matrix, of dimension (number of hidden units, number of inputs)

$\mathbf{b}$ is the RBM offset vector for input units

$\mathbf{c}$ is the RBM offset vector for hidden units

Notation: $Q(\mathbf{h}_2. = 1|\mathbf{x}_2)$ is the vector with elements $Q(\mathbf{h}_{2i} = 1|\mathbf{x}_2)$

for all hidden units i **do**
- compute $Q(\mathbf{h}_{1i} = 1|\mathbf{x}_1)$ (for binomial units, $\text{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{1j})$)
- sample $\mathbf{h}_{1i} \in \{0, 1\}$ from $Q(\mathbf{h}_{1i}|\mathbf{x}_1)$

end for

for all visible units j **do**
- compute $P(\mathbf{x}_{2j} = 1|\mathbf{h}_1)$ (for binomial units, $\text{sigm}(\mathbf{b}_j + \sum_i W_{ij}\mathbf{h}_{1i})$)
- sample $\mathbf{x}_{2j} \in \{0, 1\}$ from $P(\mathbf{x}_{2j} = 1|\mathbf{h}_1)$

end for

for all hidden units i **do**
- compute $Q(\mathbf{h}_{2i} = 1|\mathbf{x}_2)$ (for binomial units, $\text{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{2j})$)

end for

- $W \leftarrow W + \epsilon(\mathbf{h}_1\mathbf{x}_1' - Q(\mathbf{h}_2. = 1|\mathbf{x}_2)\mathbf{x}_2')$
- $\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{x}_1 - \mathbf{x}_2)$
- $\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{h}_1 - Q(\mathbf{h}_2. = 1|\mathbf{x}_2))$

CD-1: Contrastive Divergence

[Bengio: "Learning Deep Architectures for AI", 2009]

Máquinas de Boltzmann

PCD: Persistent Contrastive Divergence

Mini-batch learning @ RBMs [Tieleman 2008]

Fase positiva: $E_{data}=\langle v_i h_j \rangle_{\mathbf{v}}$

- Fijamos la unidades visibles al valor de un vector de1l conjunto de entrenamiento.
- Calculamos el valor exacto $\langle v_i h_j \rangle$ para todos los pares (unidad visible, unidad oculta).
- Para cada par de unidades conectadas, promediamos $\langle v_i h_j \rangle$ sobre los datos del mini-lote.

Máquinas de Boltzmann

PCD: Persistent Contrastive Divergence
Mini-batch learning @ RBMs [Tieleman 2008]

Fase negativa: E_{model}

- Mantenemos un conjunto de partículas (cada partícula es un vector que corresponde a una configuración global de la red RBM).
- Actualizamos cada partícula unas cuantas veces utilizando actualizaciones paralelas alternas.
- Para cada par de unidades conectadas, promediamos $<v_i h_j>$ sobre el conjunto de partículas.

Máquinas de Boltzmann

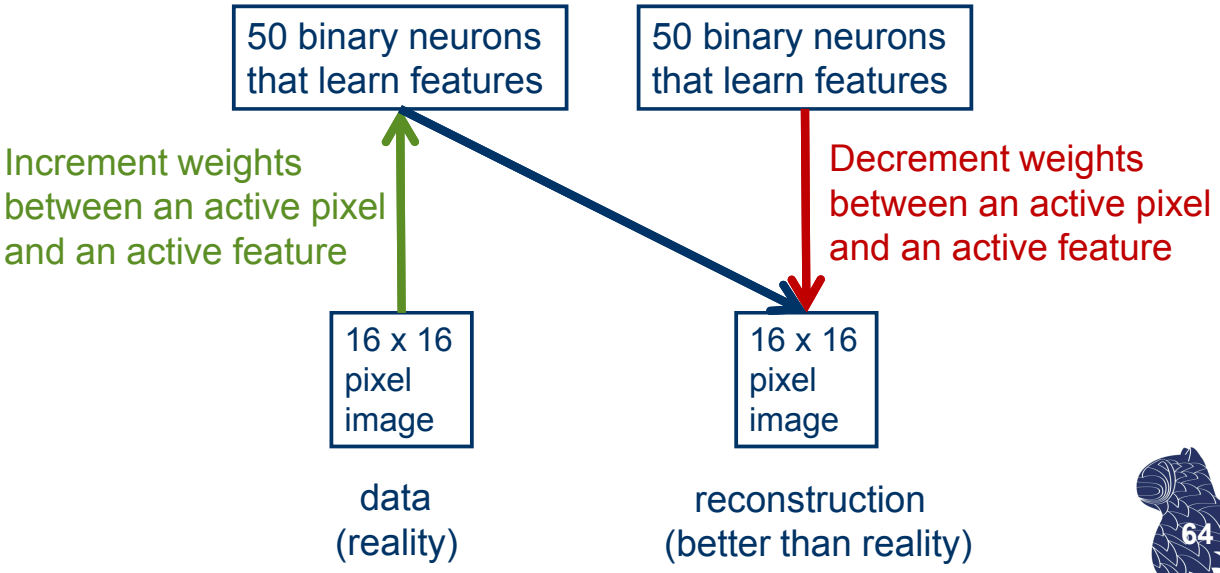
Algorithm 27.4: Persistent CD for training an RBM with binary hidden and visible units

Initialize weights $\mathbf{W} \in \mathbb{R}^{D \times L}$ randomly;
Initialize chains $(\mathbf{v}_s, \mathbf{h}_s)_{s=1}^{S}$ randomly ;
for $t = 1, 2, \ldots$ **do**
  // Mean field updates ;
  for *each case* $i = 1 : N$ **do**
    $\mu_{ik} = \text{sigm}(\mathbf{v}_i^T \mathbf{w}_{:,k})$
  // MCMC updates ;
  for *each sample* $s = 1 : S$ **do**
    Generate $(\mathbf{v}_s, \mathbf{h}_s)$ by brief Gibbs sampling from old $(\mathbf{v}_s, \mathbf{h}_s)$
  // Parameter updates ;
  $\mathbf{g} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{v}_i (\boldsymbol{\mu}_i)^T - \frac{1}{S} \sum_{s=1}^{S} \mathbf{v}_s (\mathbf{h}_s)^T$;
  $\mathbf{W} := \mathbf{W} + \alpha_t \mathbf{g}$;
  Decrease α_t

PCD: Persistent Contrastive Divergence
[Murphy: "Machine Learning: A Probabilistic Perspective", 2012]

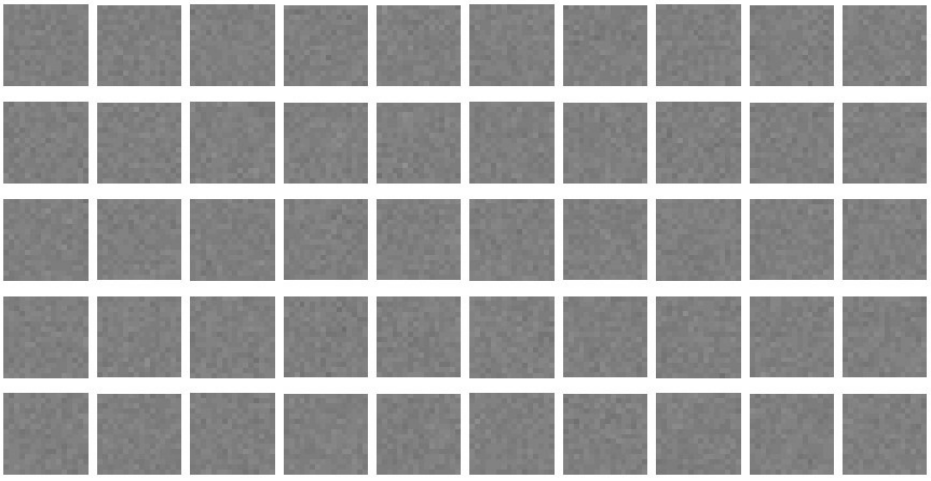
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]
Características que permiten reconstruir imágenes…



Máquinas de Boltzmann

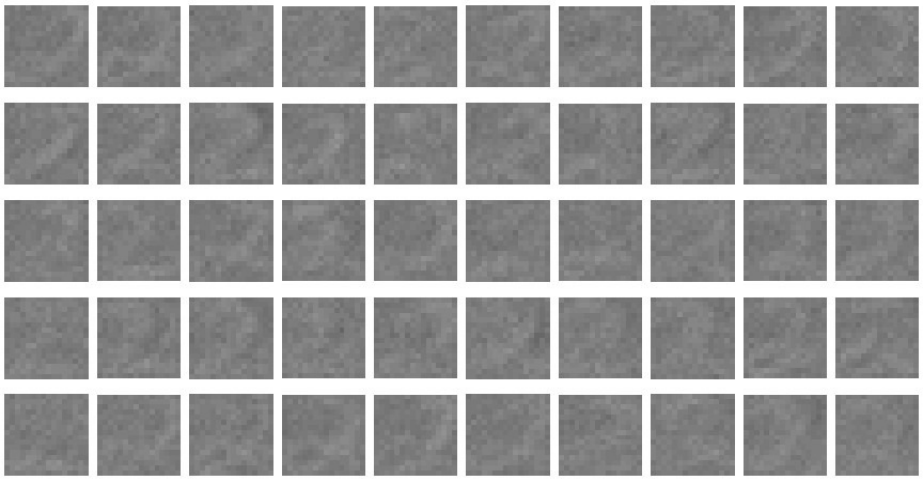
EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



Inicialización:
Pesos pequeños aleatorios para romper simetrías

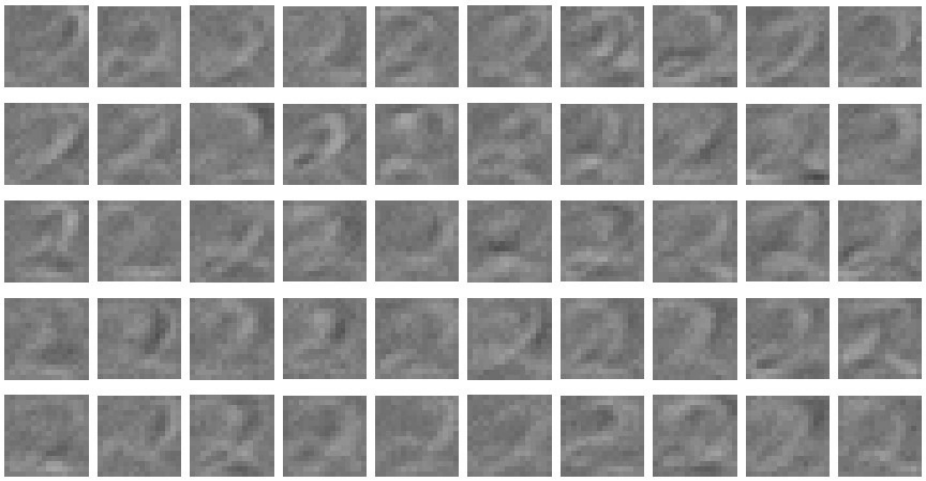
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



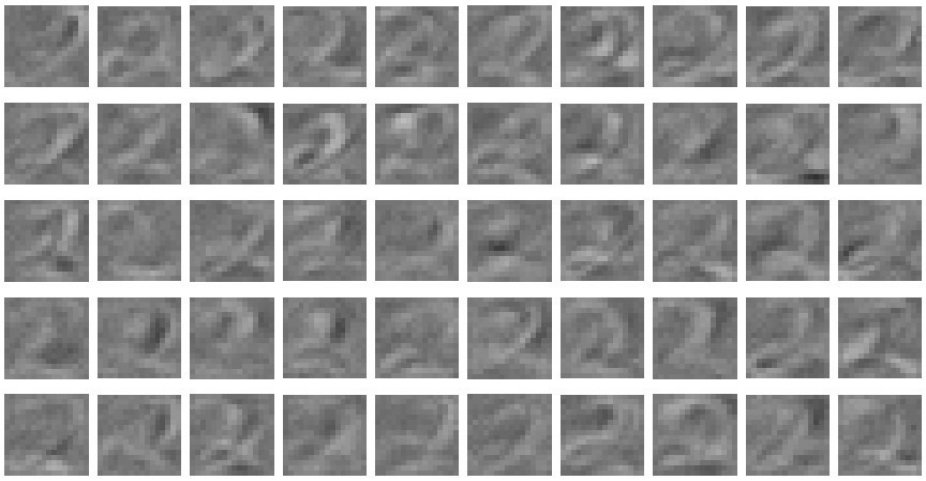
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



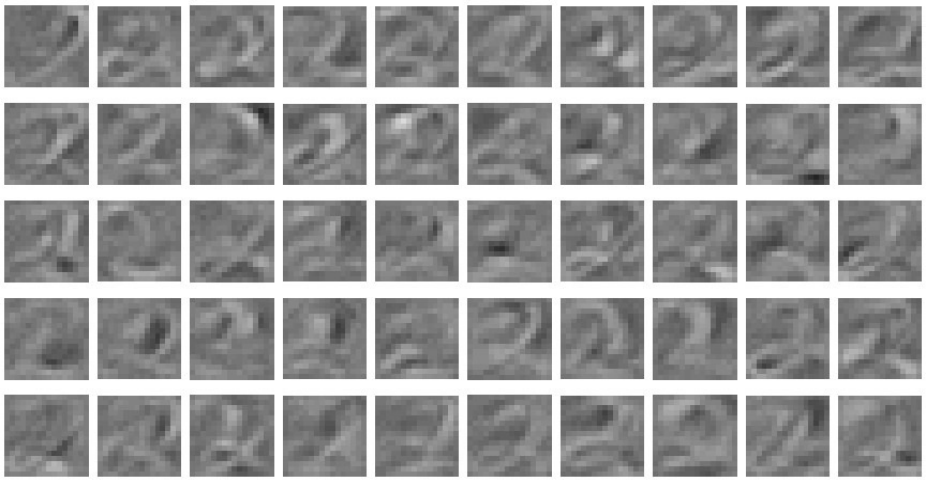
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



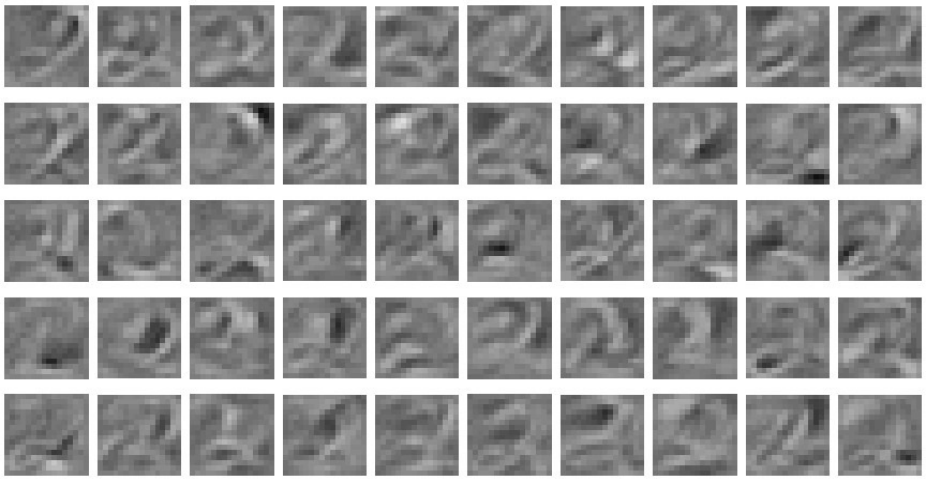
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



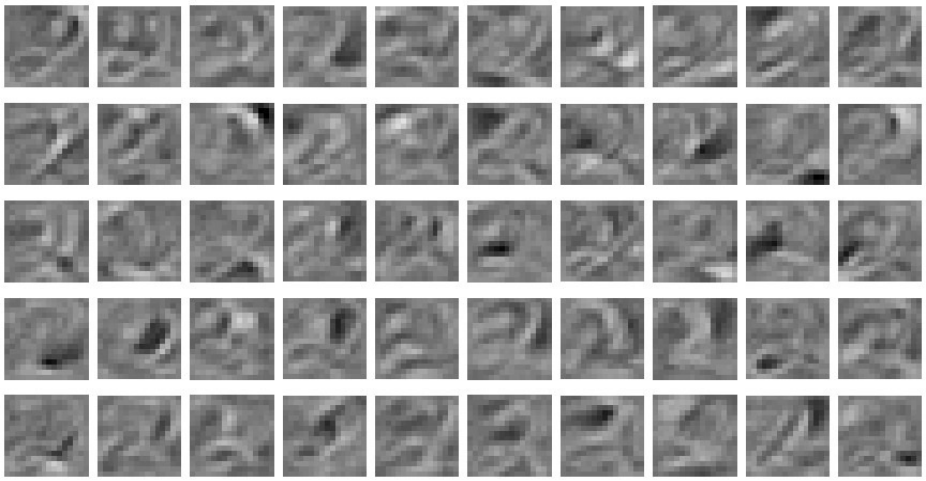
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



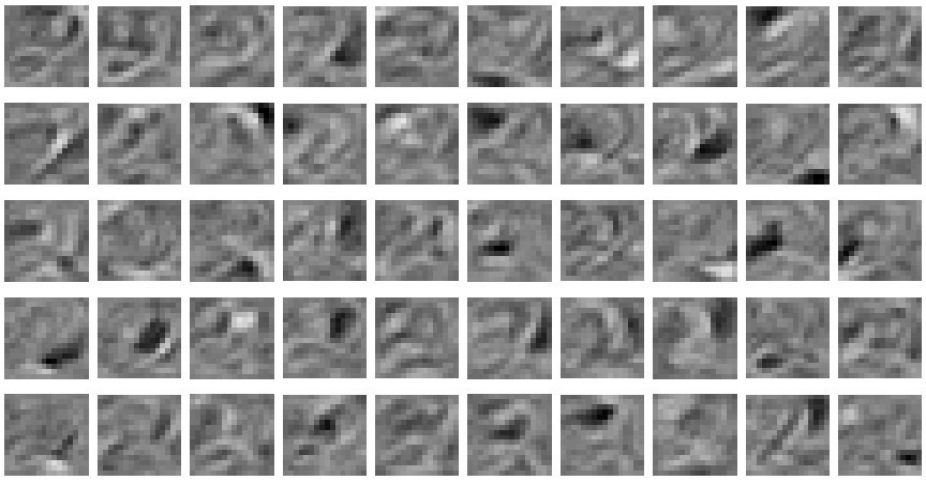
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



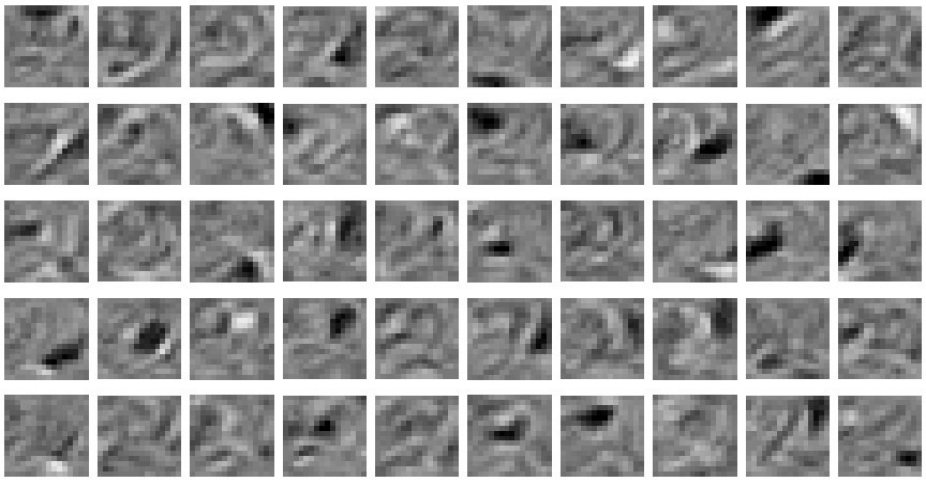
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



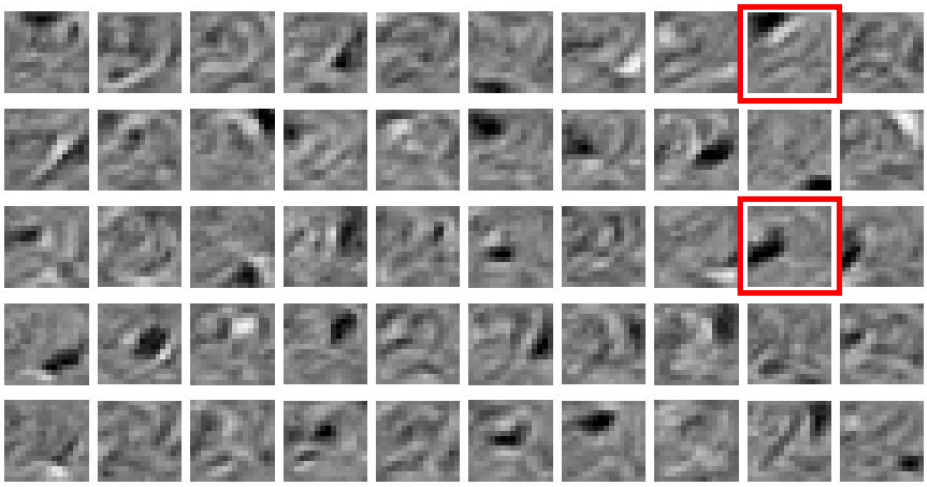
Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



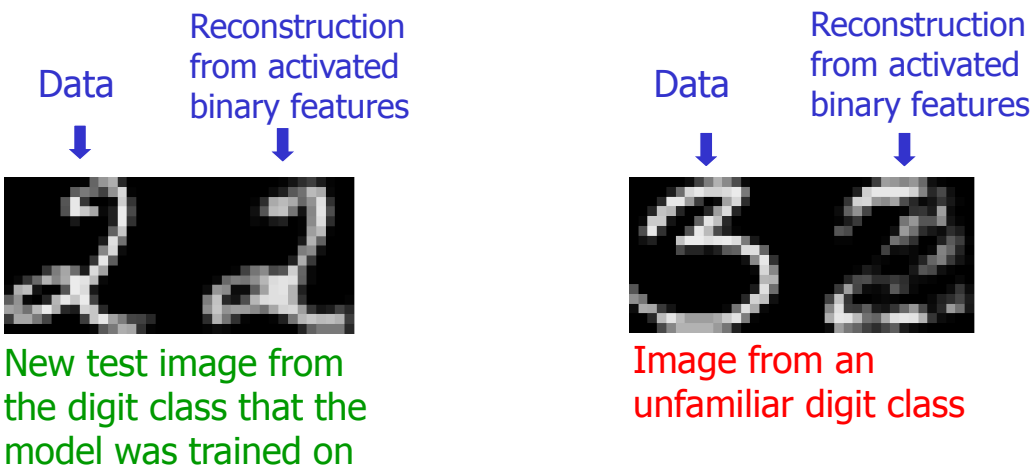
Configuración final de los pesos:

Cada neurona oculta aprende una característica distinta.

Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]

¿Cómo se reconstruyen las imágenes?

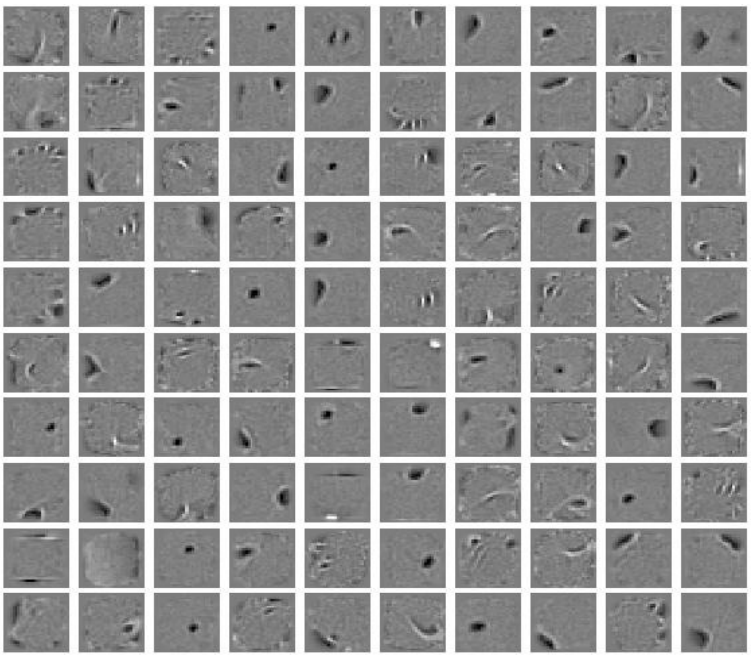


New test image from the digit class that the model was trained on

Image from an unfamiliar digit class

Máquinas de Boltzmann

EJEMPLO: DIVERGENCIA CONTRASTIVA [Hinton]



Características aprendidas por la primera capa oculta de un modelo de los 10 dígitos.

Máquinas de Boltzmann

APLICACIÓN: SISTEMAS DE RECOMENDACIÓN

La competición del millón de dólares

NETFLIX

- Tenemos las evaluaciones de medio millón de usuarios sobre 18000 películas en una escala de 1 a 5.
- Cada usuario sólo evalúa una pequeña fracción de las películas.

OBJETIVO

Predecir las evaluaciones de películas:
"filtrado colaborativo" [collaborative filtering]

Máquinas de Boltzmann

APLICACIÓN: SISTEMAS DE RECOMENDACIÓN

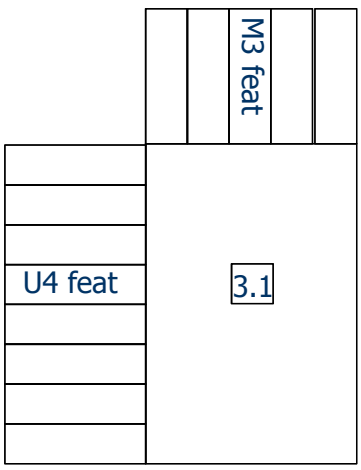
La competición del millón de dólares

	M1	M2	M3	M4	M5	M6
U1				3		
U2	5		1			
U3		3	5			
U4	4		?			5
U5			4			
U6					2	

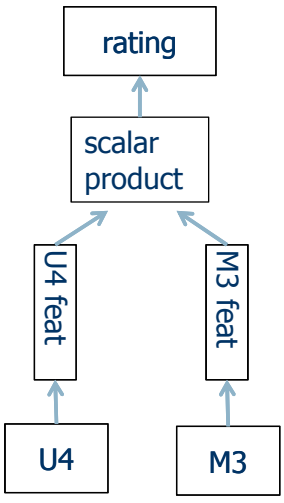
Máquinas de Boltzmann

APLICACIÓN: SISTEMAS DE RECOMENDACIÓN

La competición del millón de dólares



Factorización de matrices



Alternativa basada en RBM

Máquinas de Boltzmann

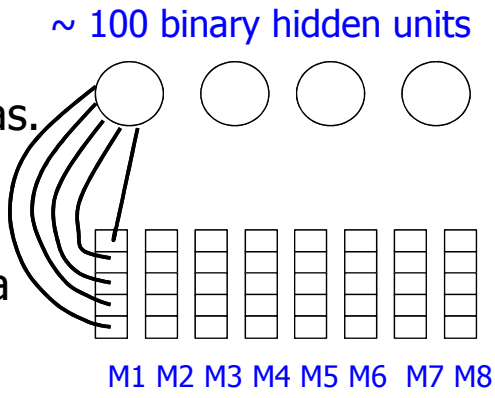
APLICACIÓN: SISTEMAS DE RECOMENDACIÓN

La competición del millón de dólares

Tratamos cada usuario
como un caso de entrenamiento:

- Vector de evaluaciones de películas.
- Una unidad visible por película (5-way softmax): la regla CD para softmax es la misma que para una unidad binaria
- ~100 unidades ocultas

Uno de los valores visibles es desconocido (tiene que proporcionarlo el modelo).



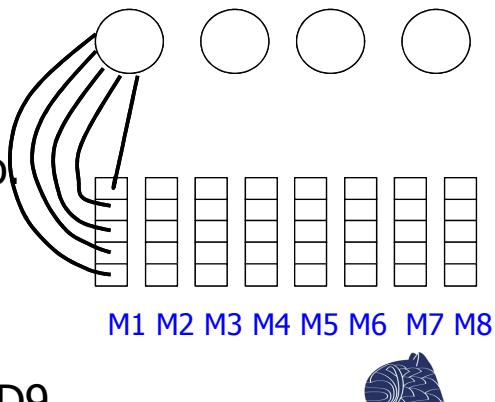
Máquinas de Boltzmann

APLICACIÓN: SISTEMAS DE RECOMENDACIÓN

La competición del millón de dólares

No tenemos evaluaciones de todos los usuarios para todas las películas:

- Para cada usuario, usamos una RBM con unidades visibles sólo para las películas que ha evaluado.
- Tenemos diferentes RBMs, pero se comparten los pesos.
- Los modelos se entrenan primero con CD1, luego con CD3, CD5 y CD9
- Cada RBM sólo tiene un caso de entrenamiento.



Máquinas de Boltzmann

APLICACIÓN: SISTEMAS DE RECOMENDACIÓN

La competición del millón de dólares

- Las RBMs funcionan tan bien como los métodos de factorización de matrices, pero dan distintos errores.
- Se pueden combinar sus predicciones.
- El equipo que se llevó el premio utilizaba diferentes RBMs, que combinaba con múltiples modelos.

Deep Belief Networks (DBNs)

Las redes de creencia [belief networks] son grafos dirigidos acíclicos compuestos de variables estocásticas.

Cuando observamos algunas variables, queremos resolver dos problemas:

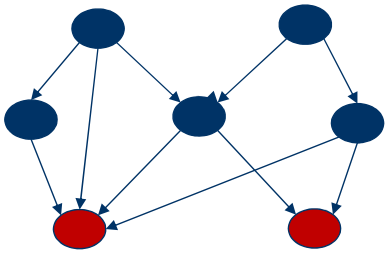
- **Inferencia:**
  Inferir los estados de las variables no observadas.
- **Aprendizaje:**
  Ajustar las interacciones entre las variables para que sea más probable que la red genere los datos.

Deep Belief Networks (DBNs)

Dos tipos de redes neuronales
compuestas de neuronas estocásticas

- Redes de Hopfield y máquinas de Boltzmann
  (conexiones simétricas & funciones de energía)

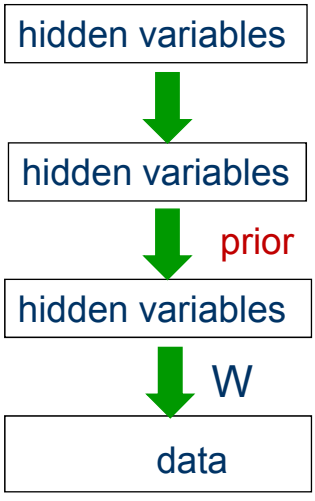
- Redes causales (grafos dirigidos acíclicos):
  Sigmoid Belief Nets [Neal 1992]



Deep Belief Networks (DBNs)

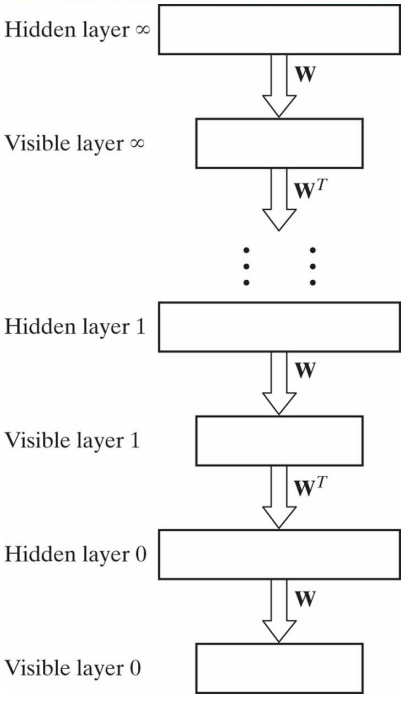
¿Por qué es difícil aprender SBNs?

Para aprender W,
necesitamos muestrear de la
distribución de la primera capa oculta:



- Aunque dos causas ocultas sean independientes, se vuelven dependientes cuando observamos el efecto en el que ambas pueden influir ["explaining away"].
- Necesitamos conocer los pesos de las capas superiores (todos los pesos interactúan).
- Tenemos que integrar sobre todas las posibles configuraciones de las capas superiores !!!

Deep Belief Networks (DBNs)



RBM ≡ SBN of infinite depth

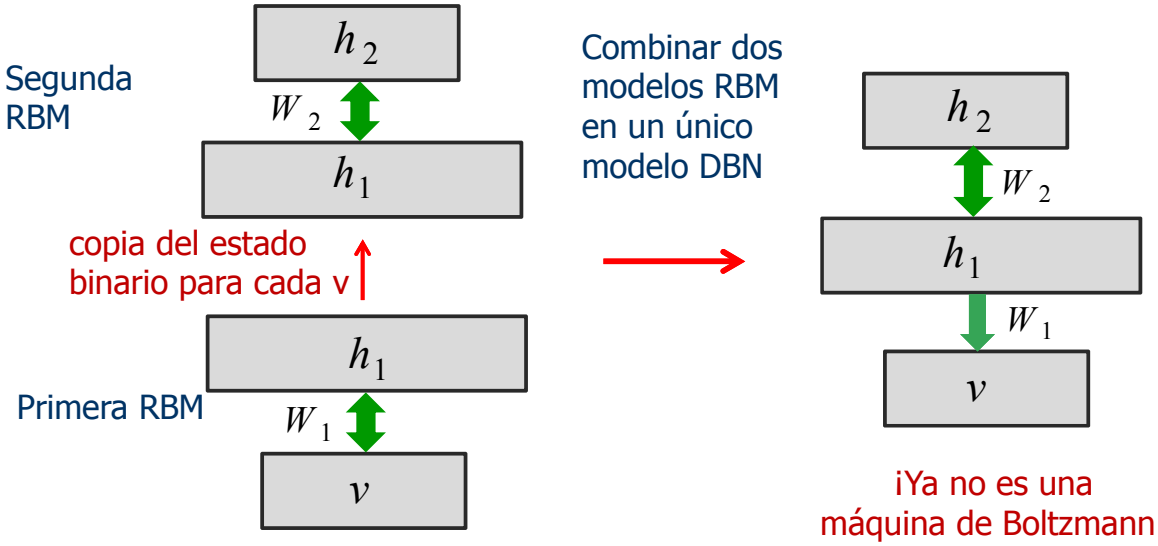
[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Deep Belief Networks (DBNs)

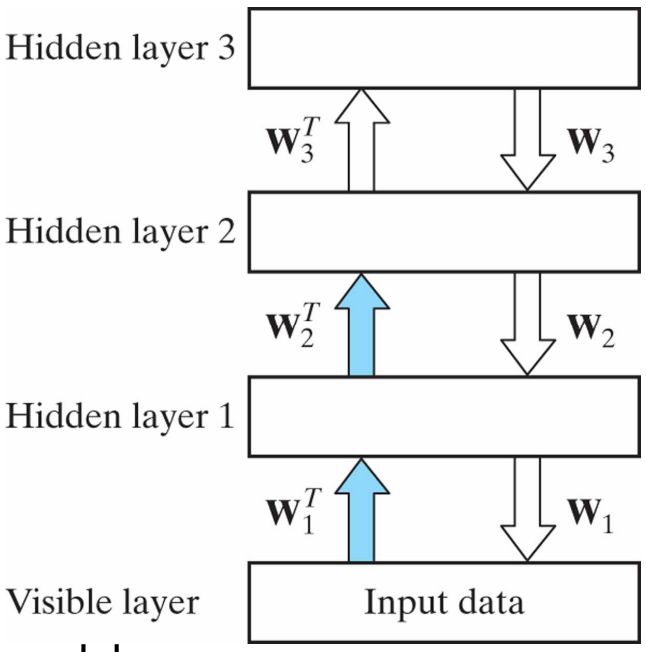
IDEA: "Stacking"

- Entrenamos una capa de características que reciben directamente los datos de entrada.
- A continuación, usamos las activaciones de las características aprendidas como entradas para aprender las características de una segunda capa de características.
- Repetimos…

Deep Belief Networks (DBNs)



Deep Belief Networks (DBNs)



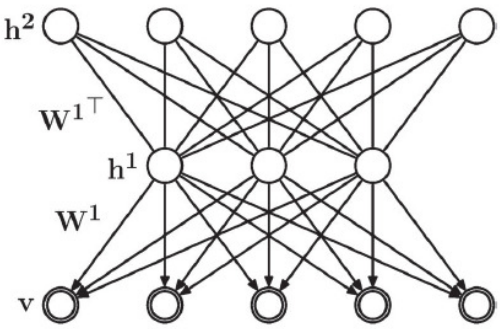
DBN generative model

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Deep Belief Networks (DBNs)

Algoritmo greedy de aprendizaje por capas
[Greedy layer-wise learning of DBNs]

- Ajustar una RBM para aprender W_1 (CD-1 o PCD).
- Desenrollar la RBM en una DBN con dos capas ocultas:



Deep Belief Networks (DBNs)

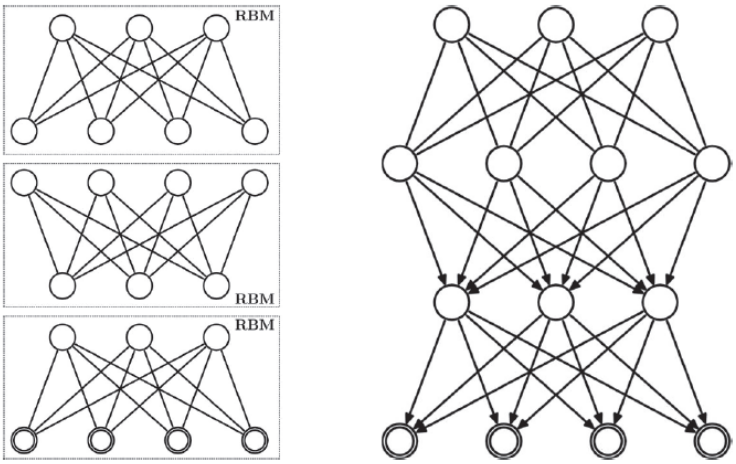
Algoritmo greedy de aprendizaje por capas
[Greedy layer-wise learning of DBNs]

- "Congelamos" los pesos dirigidos W_1 y dejamos que W_2 no tenga que coincidir con W_1^T.
- Ajustamos una segunda RBM para aprender $p(h_1|W_2)$. La entrada de esta segunda RBM será la activación de las unidades ocultas $E[h_1|v,W_1]$.
- Seguimos añadiendo capas ocultas…

Deep Belief Networks (DBNs)

Algoritmo greedy de aprendizaje por capas

[Greedy layer-wise learning of DBNs]



Stack of RBMs and corresponding DBN

Ruslan Salakhutdinov: Deep Generative Models
Ph.D. thesis, University of Toronto, 2009

Deep Belief Networks (DBNs)

Algorithm 2

`TrainUnsupervisedDBN`$(\widehat{P}, \epsilon, \ell, W, \mathbf{b}, \mathbf{c}, \texttt{mean\_field\_computation})$
Train a DBN in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an RBM (e.g. by Contrastive Divergence).
$\widehat{P}$ is the input training distribution for the network
ϵ is a learning rate for the RBM training
ℓ is the number of layers to train
W^k is the weight matrix for level k, for k from 1 to ℓ
$\mathbf{b}^k$ is the visible units offset vector for RBM at level k, for k from 1 to ℓ
$\mathbf{c}^k$ is the hidden units offset vector for RBM at level k, for k from 1 to ℓ
`mean_field_computation` is a Boolean that is true iff training data at each additional level is obtained by a mean-field approximation instead of stochastic sampling

for $k = 1$ to ℓ **do**
  • initialize $W^k = 0$, $\mathbf{b}^k = 0$, $\mathbf{c}^k = 0$
  while not stopping criterion **do**
    • sample $\mathbf{h}^0 = \mathbf{x}$ from $\widehat{P}$
    for $i = 1$ to $k-1$ **do**
      if `mean_field_computation` **then**
        • `assign` $\mathbf{h}^i_j$ `to` $Q(\mathbf{h}^i_j = 1|\mathbf{h}^{i-1})$, `for all elements` j `of` $\mathbf{h}^i$
      else
        • `sample` $\mathbf{h}^i_j$ `from` $Q(\mathbf{h}^i_j|\mathbf{h}^{i-1})$, `for all elements` j `of` $\mathbf{h}^i$
      end if
    end for
    • `RBMupdate`$(\mathbf{h}^{k-1}, \epsilon, W^k, \mathbf{b}^k, \mathbf{c}^k)$ {thus providing $Q(\mathbf{h}^k|\mathbf{h}^{k-1})$ for future use}
  end while
end for

Algoritmo de entrenamiento de una DBN

[Bengio: "Learning Deep Architectures for AI", 2009]

Deep Belief Networks (DBNs)

Algoritmo greedy de aprendizaje por capas

[Greedy layer-wise learning of DBNs]

¿Por qué funciona?

- El model RBM inferior puede expresarse como

$$p(v) = \sum_h p(h)\ p(v \mid h)$$

- Si dejamos intacto p(v|h) y mejoramos p(h), mejoraremos p(v)

Deep Belief Networks (DBNs)

Ajuste final: "backfitting" / "fine-tuning"

Se suelen afinar los pesos finales usando una versión "contrastiva" del algoritmo wake-sleep para SBNs:

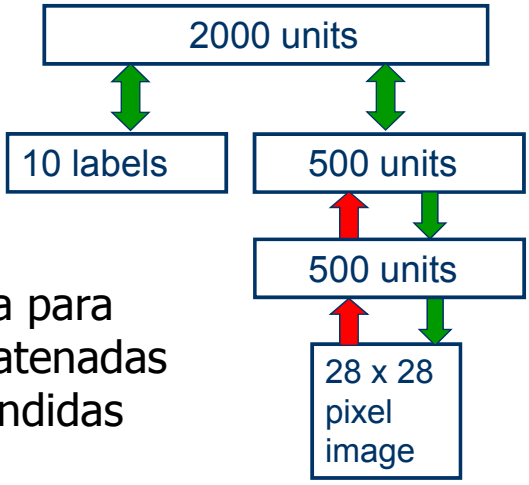
- Muestreo ascendente para ajustar los pesos descendentes (para que sean buenos a la hora de reconstruir las actividades de la capa inferior).
- Muestreo de Gibbs en la RBM superior (ajustar los pesos de la RBM usando CD).
- Muestreo descendente para ajustar los pesos ascendentes (para que sean buenos a la hora de reconstruir las actividades de la capa superior).

Por desgracia, este procedimiento es muy lento.

Deep Belief Networks (DBNs)

MNIST

- Las primeras dos capas ocultas se aprenden sin utilizar las etiquetas.

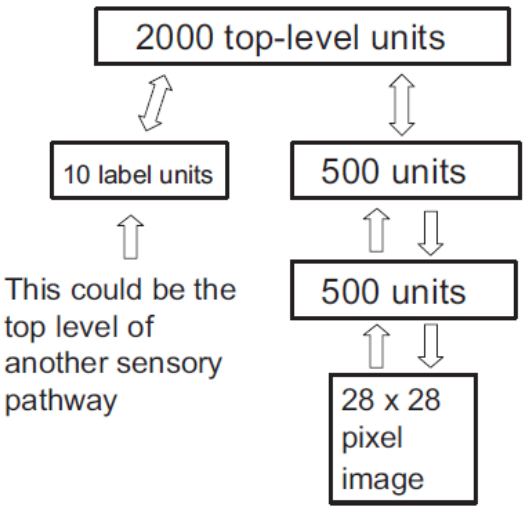


- La capa superior se entrena para modelar las etiquetas concatenadas con las características aprendidas en la segunda capa oculta.

- Los pesos finales se afinan para obtener un modelo generativo mejor usando "contrastive wake-sleep".

Deep Belief Networks (DBNs)

MNIST: 1.25% error



Geoffrey E. Hinton, Simon Osindero & Yee-Whye Teh:
A fast learning algorithm for deep belief nets.
Neural Computation 18, 1527–1554, 2006.

Deep Belief Networks (DBNs)

DEMO

Geoffrey Hinton: "The Next Generation of Neural Networks"
Google Tech Talks, 2007
https://www.youtube.com/watch?v=AyzOUbkUf3M

DBN-DNN

Generative pre-training of deep neural nets

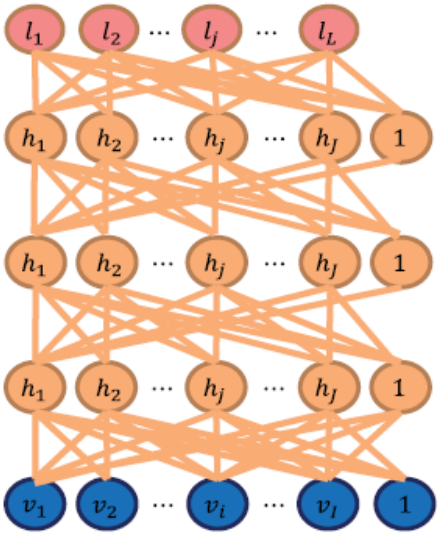
- Backpropagation no funciona bien cuando existen varias capas ocultas, debido al problema conocido como "desaparición del gradiente" [vanishing gradient]: Cuanto más nos alejamos de los datos, más pequeños son los gradientes.

- Sin embargo, podemos inicializar los parámetros de la red utilizando aprendizaje no supervisado: forzamos que el modelo tenga una respuesta multidimensional (los vectores de entrada) en vez de escalar (la clase) y con eso se mejora el entrenamiento de la red.

DBN-DNN

Generative pre-training of deep neural nets

Las redes multicapa con varias capas ocultas [deep neural networks] que se entrenan con una fase de pre-entrenamiento no supervisado DBN seguido de backpropagation se denominan DBN-DNN.



DBN-DNN

Para resolver problemas de aprendizaje usando DBNs:

- Utilizamos el algoritmo greedy de aprendizaje por capas para apilar un conjunto de RBMs.
- Consideramos este proceso como un "pre-entrenamiento" que nos ayuda a encontrar un buen conjunto inicial de pesos, que luego afinaremos utilizando una técnica de **búsqueda local**.
- "Contrastive wake-sleep" es mejor para modelos generativos, mientras que **backpropagation** se usa para modelos discriminativos.

DBN-DNN

Este proceso (pre-training + backpropagation):

- Solventa muchas de las limitaciones del uso del backpropagation estándar.
- Facilita el aprendizaje de redes con varias capas ocultas [deep nets].
- Permite que las redes generalicen mejor.

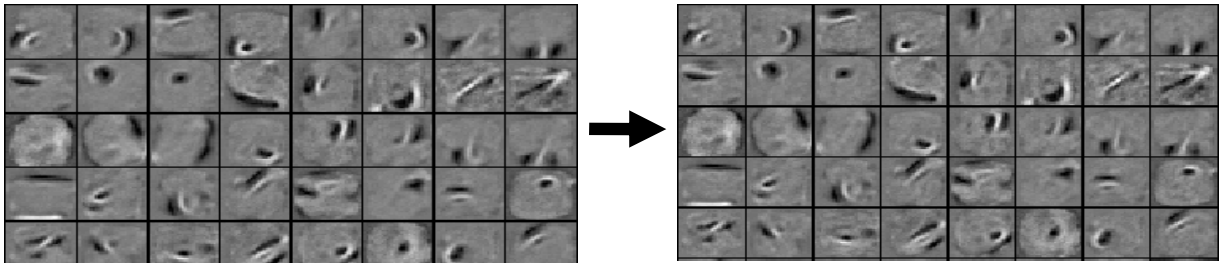
DBN-DNN

¿Por qué funciona?

- **Optimización**: El algoritmo greedy por capas es escalable y no empezamos a utilizar backpropagation hasta que tenemos un conjunto razonable de predictores de características (que pueden ser muy útiles para discriminar entre clases).
- **Sobreaprendizaje**: La mayor parte de la información proviene de los datos de entrada; el ajuste final sólo modifica ligeramente las fronteras de decisión.

  Además, podemos aprovechar datos no etiquetados.

DBN-DNN

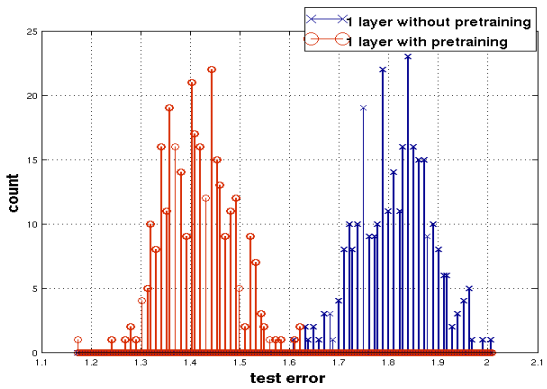


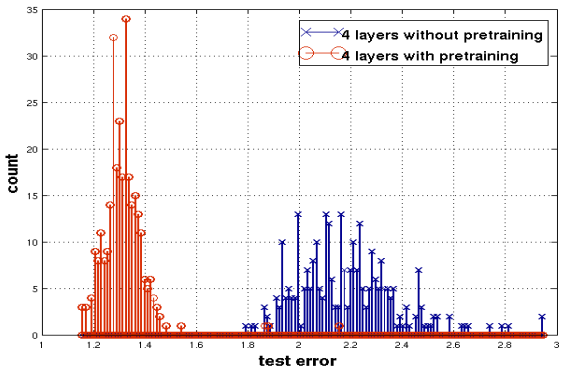
Before fine-tuning

After fine-tuning

DBN-DNN

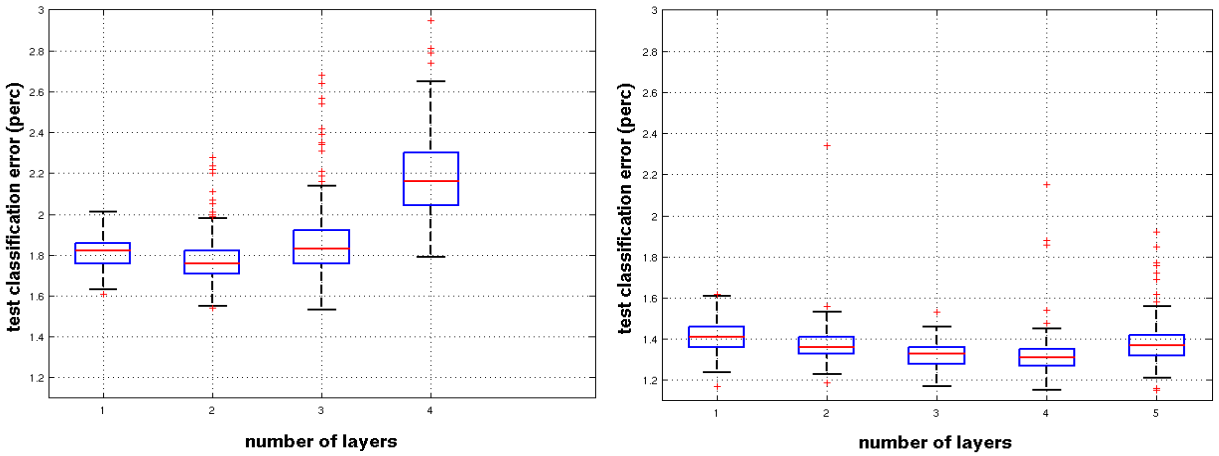
El efecto del pre-entrenamiento no supervisado
[Erhan et al., AISTATS'2009]





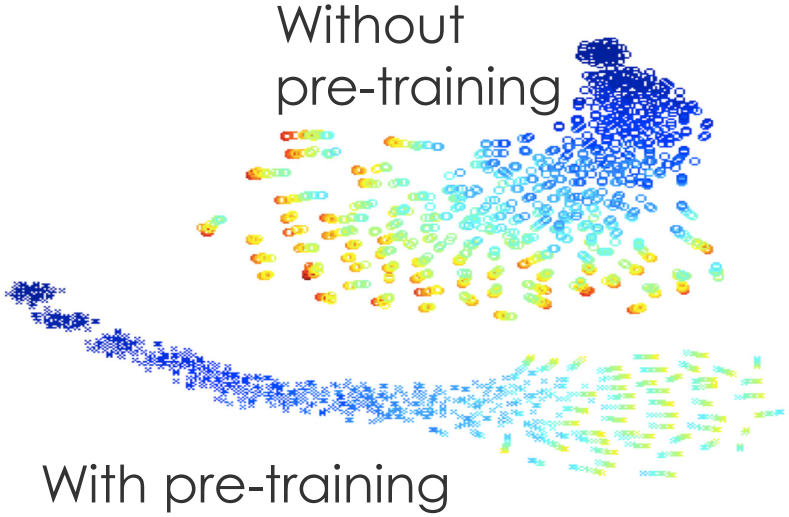
DBN-DNN

El efecto del pre-entrenamiento no supervisado
[Erhan et al., AISTATS'2009]



DBN-DNN

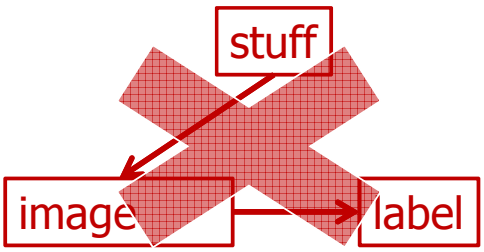
El efecto del pre-entrenamiento no supervisado
[Erhan et al., AISTATS'2009]

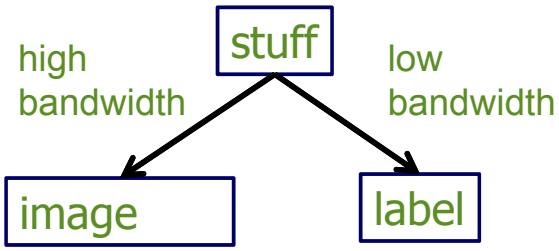


DBN-DNN

Pre-entrenamiento no supervisado

¿Por qué funciona?



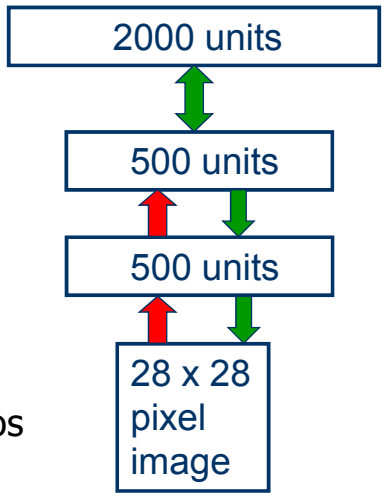


Tiene sentido descubrir primero qué es lo que generó la imagen para luego determinar a qué clase corresponde.

DBN-DNN

MNIST

1. Entrenamos la DBN.
2. Añadimos un bloque softmax sobre la capa superior…



Tasa de error

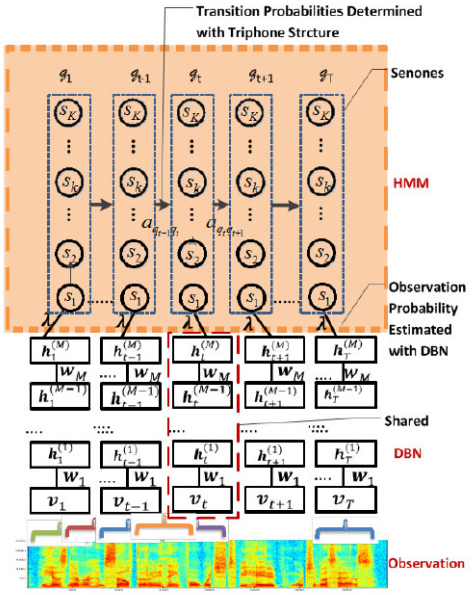
1.6%	Backprop usando 1 ó 2 capas ocultas
1.5%	Backprop con restricciones L2 sobre pesos
1.4%	Support Vector Machines
1.25%	DBN Modelo generativo (contrastive wake-sleep)
1.15%	DBN-DNN (modelo generativo + backpropagation)
0.49%	CNN 600,000 dígitos distorsionados
0.39%	CNN 600,000 dígitos distorsionados (pre-training+backprop)

DBN-DNN

Ejemplo
DBN-HMM

Red DBN-DNN utilizada para ajustar los parámetros de un modelo oculto de Markov [HMM] en sistemas de reconocimiento de voz.

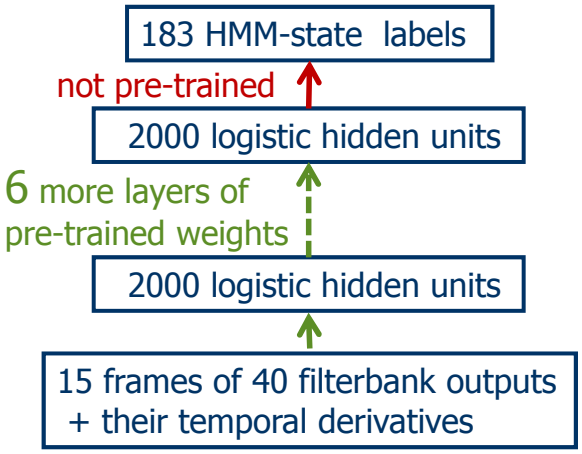
@ Microsoft Research



DBN-DNN

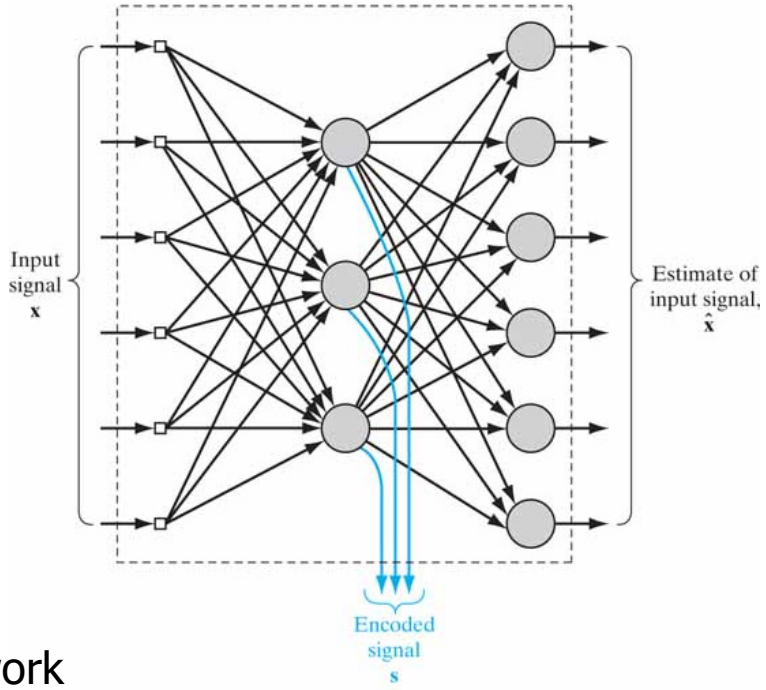
Ejemplo
DBN-HMM
TIMIT benchmark

- El mejor sistema de reconocimiento de voz independiente del hablante requería combinar varios modelos y tenía una tasa de error del **24.4%**.



- Una red DBN-HMM con 8 capas bajó el error directamente al **20.7%** y cambió la forma de diseñar sistemas de reconocimiento de voz.

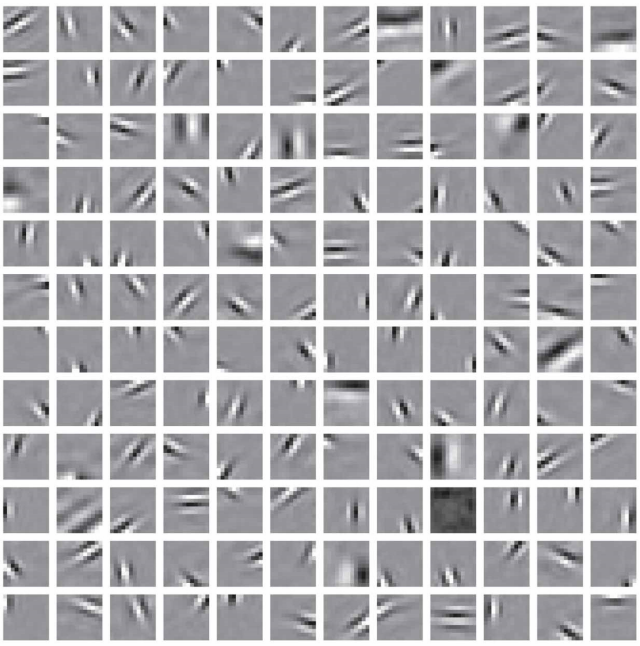
Autoencoders



Replicator network

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

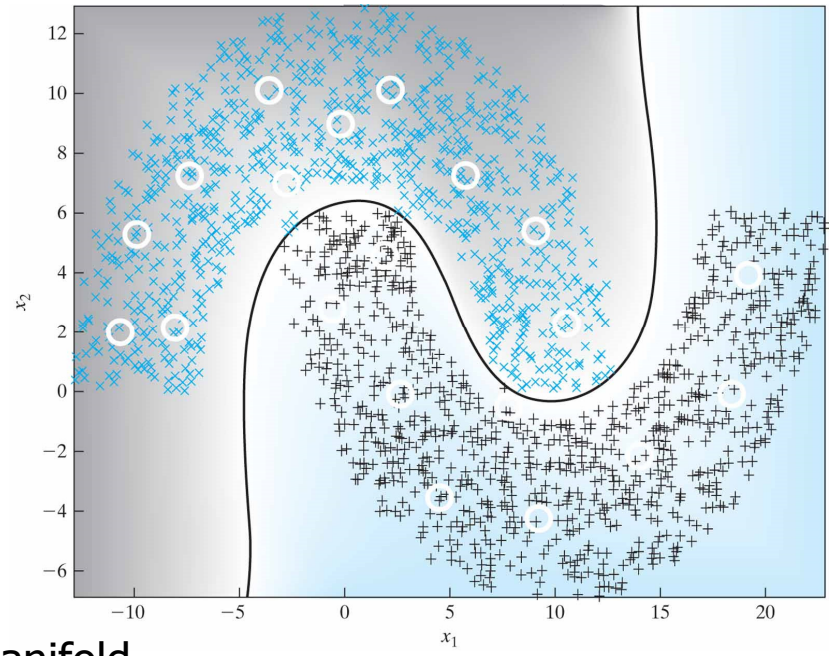
Autoencoders



Sparse coding

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Autoencoders



Optimal manifold

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Deep Autoencoders

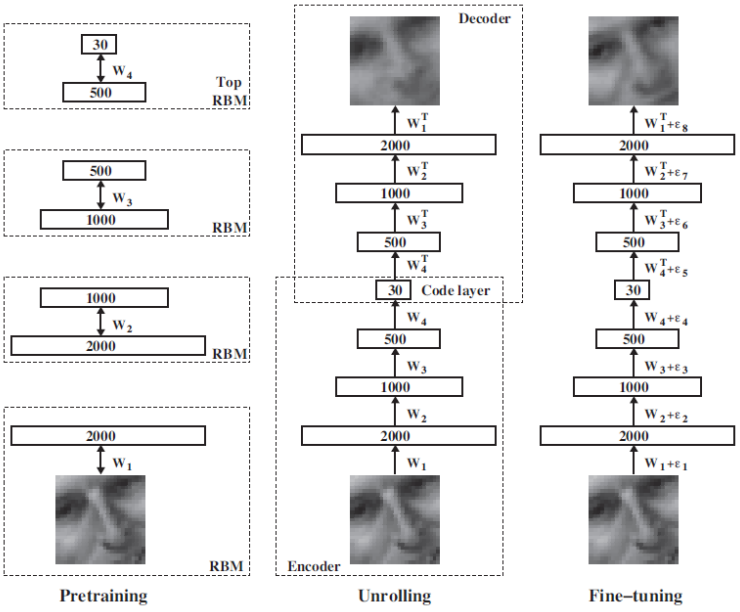
Autoencoders con múltiples capas ocultas
y un algoritmo de entrenamiento similar a las DBNs.

- Red neuronal no supervisada utilizada
  para reducir la dimensionalidad de los datos
  y aprender nuevas características.

- Como las DBNs, pueden utilizarse para inicializar los
  pesos de una red neuronal [generative pre-training].

Deep Autoencoders

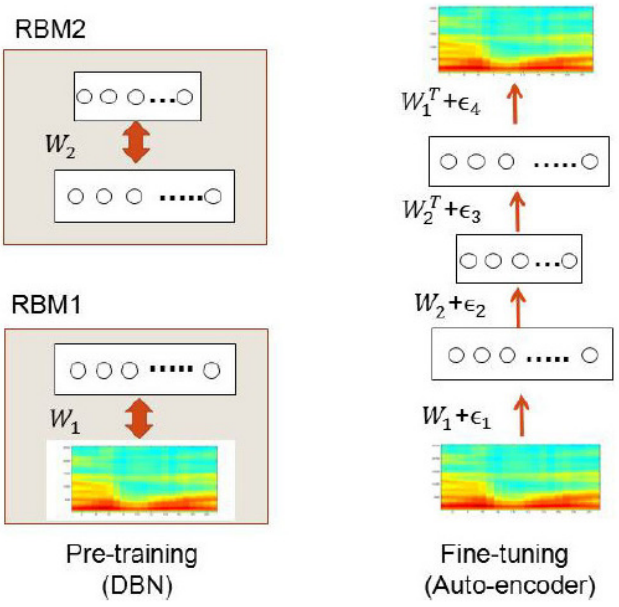
Algoritmo de entrenamiento



Geoffrey Hinton & Ruslan Salakhutdinov: "Reducing the dimensionality of data with neural networks." Science 313(5786), 504, July 2006

Deep Autoencoders

Ejemplo



L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed & G. Hinton: "Binary coding of speech spectrograms using a deep autoencoder" Interspeech'2010.

Deep Autoencoders

- Para evitar que un "autoencoder" aprenda la función identidad, lo normal es crear un cuello de botella de forma que existan menos unidades ocultas que unidades visibles (i.e. capas de codificación de menor dimensión que la capa de entrada).
- Sin embargo, hay aplicaciones en las que interesa que las capas de codificación sean más anchas que la capa de entrada (para capturar la riqueza de la distribución de los datos de entrada). En estos casos, son útiles técnicas como introducir ruido anulando datos de entrada, como "dropout" pero para las entradas [**denoising autoencoders**].

Deep Stacking Networks (DSNs)

- Las técnicas convencionales para entrenar DNNs, en su etapa final de ajuste [fine tunning] requieren la utilización del gradiente descendente estocástico, que resulta difícil de paralelizar.
- La arquitectura de las DSNs (originalmente llamadas "Deep Convex Networks" o DCNs) se diseñó con el problema de la escalabilidad del aprendizaje en mente.

Li Deng & Dong Yu: "Deep convex network: A scalable architecture for speech pattern classification". Interspeech'2011.

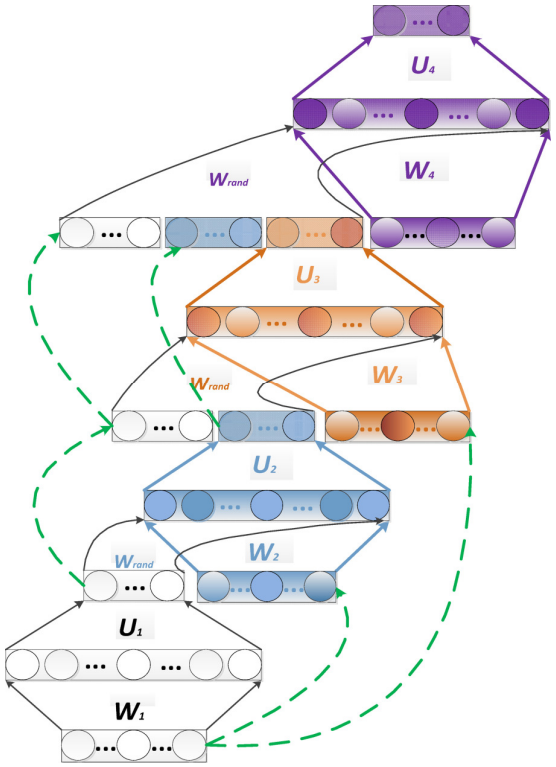
Deep Stacking Networks (DSNs)

- Las DSNs apilan redes multicapa como módulos básicos en los que las unidades de salida son lineales y las unidades ocultas son sigmoidales (no lineales).
- La linealidad de las unidades de salida permite una estimación eficiente y paralelizable de los pesos de la capa de salida dada la actividad de la capa oculta (al tratarse de un problema de optimización convexa).
- Las restricciones impuestas por la arquitectura hacen mucho más fácil el aprendizaje de los demás parámetros de la red (los pesos de la capa oculta).

Deep Stacking Networks (DSNs)

La matriz de pesos W conecta la capa lineal de entrada con la capa oculta no lineal.

La matriz de pesos U conecta la capa oculta (no lineal) con la capa lineal de salida.



Deep Stacking Networks (DSNs)

Algoritmo de aprendizaje

Dados los vectores de salida para el conjunto de entrenamiento T, los pesos U y W se ajustan para minimizar el error:

$$E = \frac{1}{2}\sum_i \|\boldsymbol{y}_i - \boldsymbol{t}_i\|^2 = \frac{1}{2}\mathrm{Tr}[(\boldsymbol{Y}-\boldsymbol{T})(\boldsymbol{Y}-\boldsymbol{T})^{\mathrm{T}}]$$

donde Y es la salida de la red:

$$\boldsymbol{y}_i = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{h}_i = \boldsymbol{U}^{\mathrm{T}}\sigma(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i) = G_i(\boldsymbol{UW})$$

Deep Stacking Networks (DSNs)

Algoritmo de aprendizaje

U puede aprenderse una vez que se tiene la matriz de actividad H para la capa oculta (o lo que es equivalente, cuando se conoce W).

Fijando la derivada del error con respecto a U a cero, obtenemos

$$\boldsymbol{U} = (\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}})^{-1}\boldsymbol{H}\boldsymbol{T}^{\mathrm{T}} = \mathrm{F}(\boldsymbol{W}), \quad \text{where } \boldsymbol{h}_i = \sigma(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}_i).$$

Esto nos proporciona una restricción explícita entre U y W que podemos aprovechar en la minimización del error.

Deep Stacking Networks (DSNs)

Algoritmo de aprendizaje

Dada la restricción U=F(W),
aplicamos el método de los multiplicadores de Lagrange:

$$E = \frac{1}{2}\sum_i \|G_i(\boldsymbol{U}, \boldsymbol{W}) - \boldsymbol{t}_i\|^2 + \lambda\|\mathbf{U} - \mathrm{F}(\boldsymbol{W})\|$$

Ahora, podemos utilizar "full-batch learning":

$$\frac{\partial E}{\partial \boldsymbol{W}} = 2\boldsymbol{X}[\boldsymbol{H}^{\mathrm{T}} \circ (\mathbf{1} - \boldsymbol{H})^{\mathrm{T}} \circ [\boldsymbol{H}^{\dagger}(\boldsymbol{H}\boldsymbol{T}^{\mathrm{T}})(\boldsymbol{T}\boldsymbol{H}^{\dagger}) - \boldsymbol{T}^{\mathrm{T}}(\boldsymbol{T}\boldsymbol{H}^{\dagger})]]$$

$$\boldsymbol{H}^{\dagger} = \boldsymbol{H}^{\mathrm{T}}(\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}})^{-1}$$

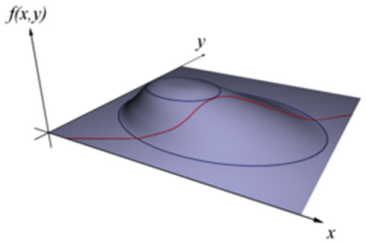
Deep Stacking Networks (DSNs)

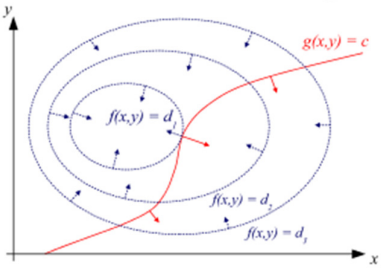
Método de Lagrange

Método de optimización de funciones
de múltiples variables sujetas a restricciones

Función objetivo f(θ)

Restricción g(θ) = 0





Los gradientes ∇f(θ) y ∇g(θ)
son paralelos, por lo que

∇f(θ) = λ∇g(θ)

Deep Stacking Networks (DSNs)

Método de Lagrange

Podemos incorporar la restricción $g(\theta)=0$ a nuestra función objetivo, añadiendo una nueva variable λ:

$$\mathbf{F(\theta,\lambda) = f(\theta) - \lambda g(\theta)}$$

F se denomina Lagrangiano.
Optimizamos, como siempre, igualando a cero su gradiente:

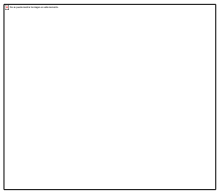
$$\nabla \mathbf{F(\theta,\lambda) = 0}$$

Deep Stacking Networks (DSNs)

Método de Lagrange

EJEMPLO

Cómo minimizar la superficie de una caja de volumen V.



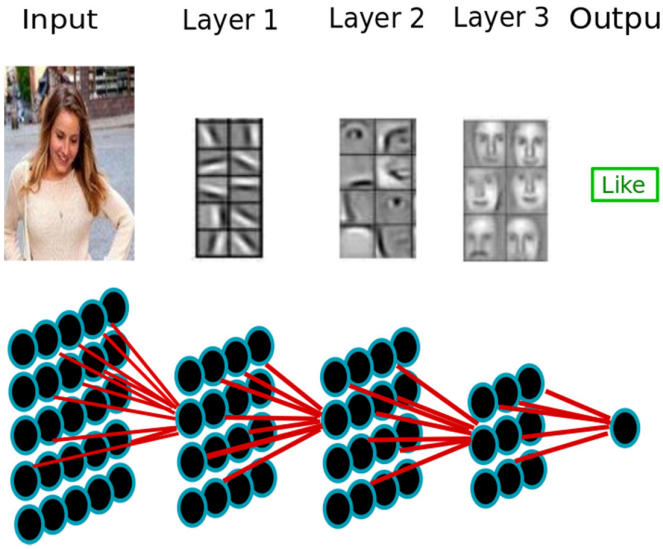
Función objetivo: $f(x,y,z) = 2(xy+xz+yz)$
Restricción: $g(x,y,z) = xyz - V = 0$

Gradiente $\nabla f(x,y,z) = < 2(y+z), 2(x+z), 2(x+y) >$
$\nabla g(x,y,z) = < yz, xz, xy >$

Al hacer $\nabla f(x,y,z) = \lambda \nabla g(x,y,z)$
obtenemos 3 ecuaciones que nos dan la solución:
$x = y = z = 4/\lambda$ (un cubo :-)

Aplicaciones

Citas por Internet, e.g. Tinder & OKCupid.com

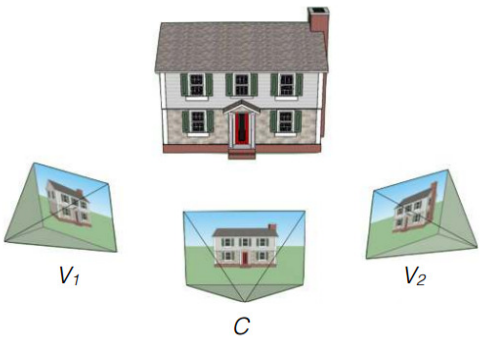


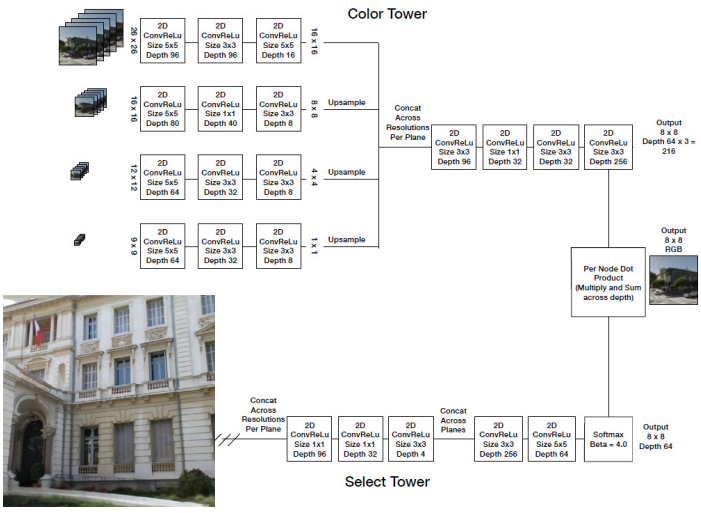
Harm De Vries & Jason Yosinski:
Can deep learning help you find the perfect match?
ICML'2015 Deep Learning Workshop

Aplicaciones

Síntesis de imágenes, e.g. Google Street View

Generación de nuevas perspectivas a partir de imágenes 2D





John Flynn, Ivan Neulander, James Philbin & Noah Snavely
DeepStereo: Learning to Predict New Views from the World's Imagery
CoRR, arXiv:1506.06825, June 2015.

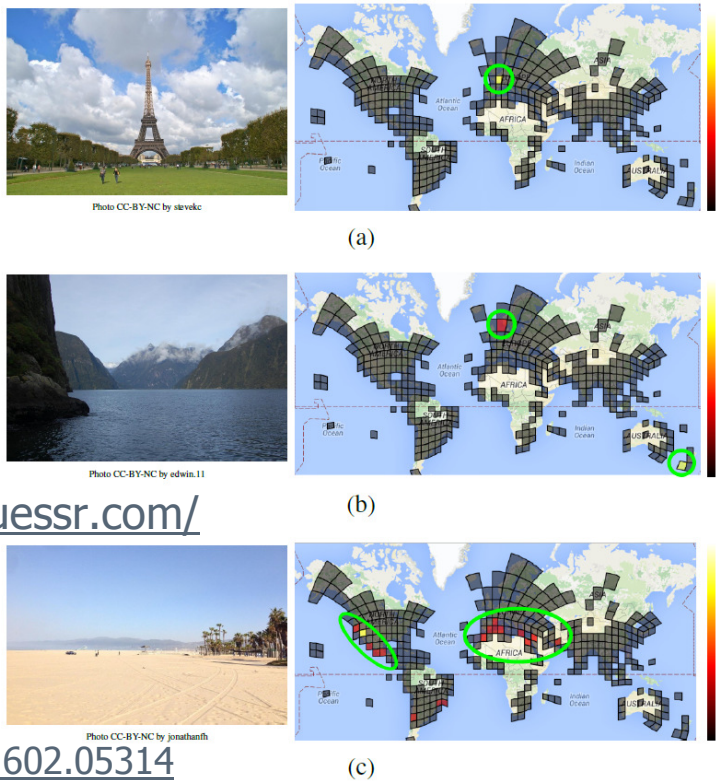
Aplicaciones

Geolocalización de imágenes

PlaNet

CNN 377MB

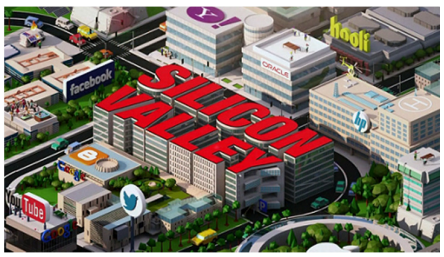
Mejor que los humanos en http://www.geoguessr.com/



http://arxiv.org/abs/1602.05314

Aplicaciones

Compresión de imágenes



http://www.piedpiper.com/

Full Resolution Image Compression with Recurrent Neural Networks arXiv, August 2016, http://arxiv.org/abs/1608.05148

Aplicaciones

Traducción automática de señales

Google app

Otavio Good (Google Translate):

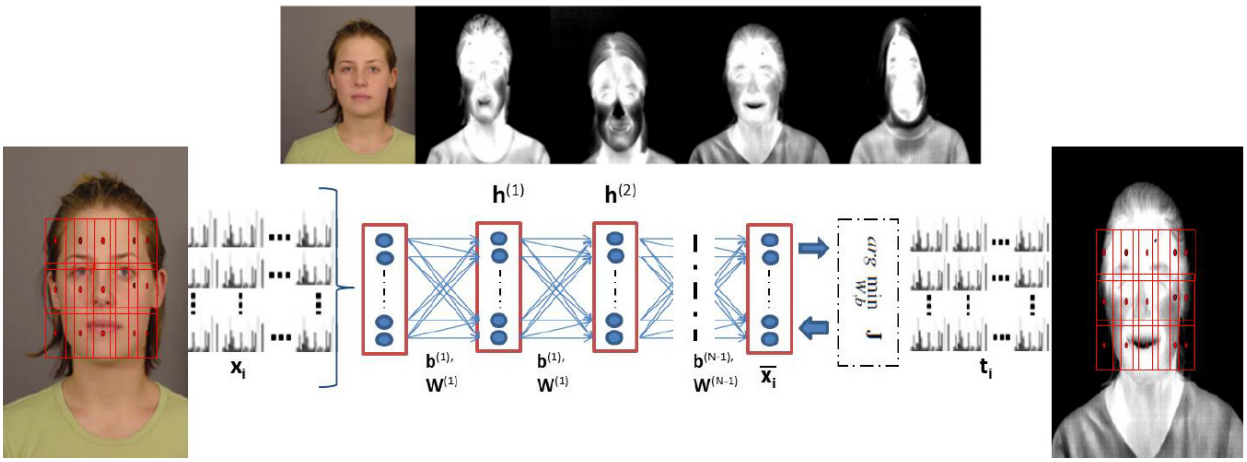
How Google Translate squeezes deep learning onto a phone

http://googleresearch.blogspot.com.es/2015/07/how-google-translate-squeezes-deep.html

Aplicaciones

Reconocimiento de caras

en imágenes térmicas (cámaras de infrarrojos)



M. Saquib Sarfraz & Rainer Stiefelhagen:

Deep Perceptual Mapping for Thermal to Visible Face Recognition

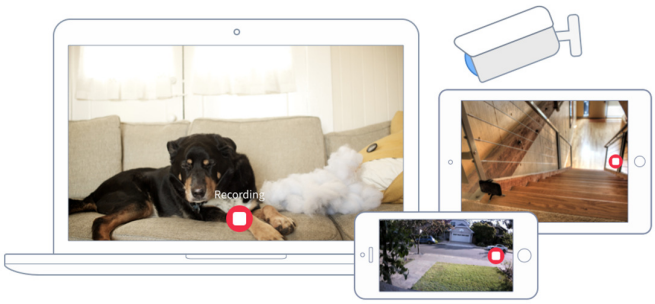
BMVC'2015

Aplicaciones

Videovigilancia

camio

If people are approaching the storage room, then call my phone.



Recipe

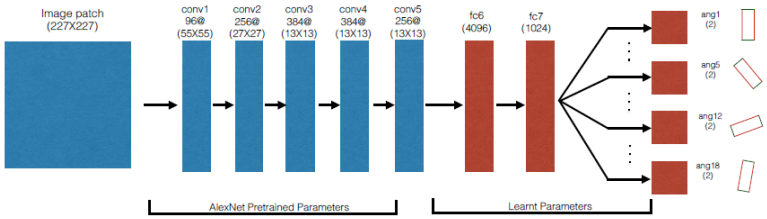
if this then that

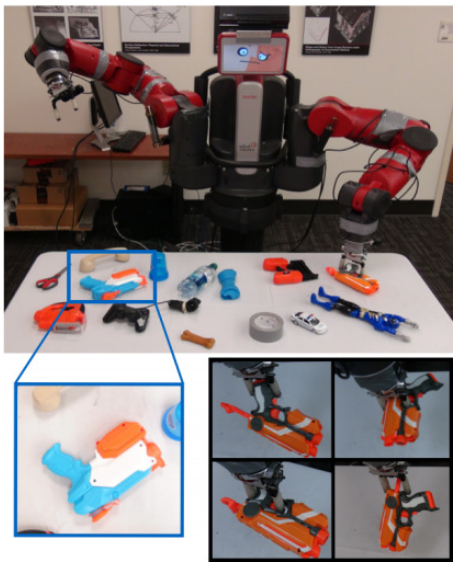
Trigger Action

Rachel Metz:
Using Deep Learning to Make Video Surveillance Smarter
MIT Technology Review, August 2015

Aplicaciones

Robótica



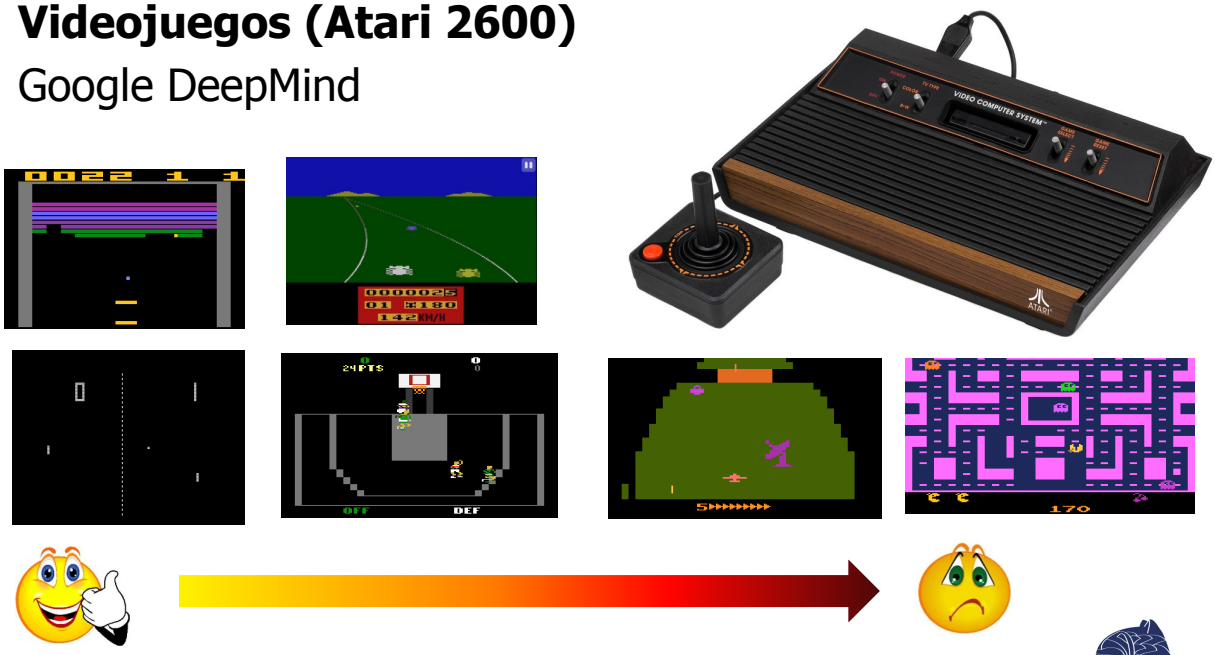


Lerrel Pinto & Abhinav Gupta (CMU): **Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours**
arXiv, September 2015. http://arxiv.org/abs/1509.06825

Aplicaciones

Videojuegos (Atari 2600)

Google DeepMind

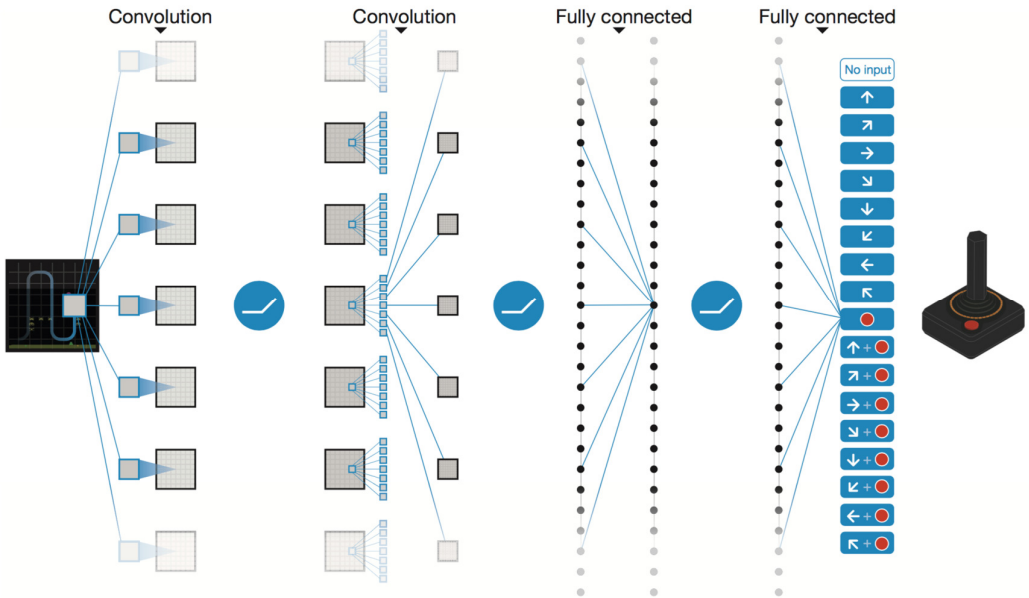


"Google AI beats humans at more classic arcade games than ever before"
http://arxiv.org/pdf/1509.06461v1.pdf (September 2015)

Aplicaciones

Videojuegos

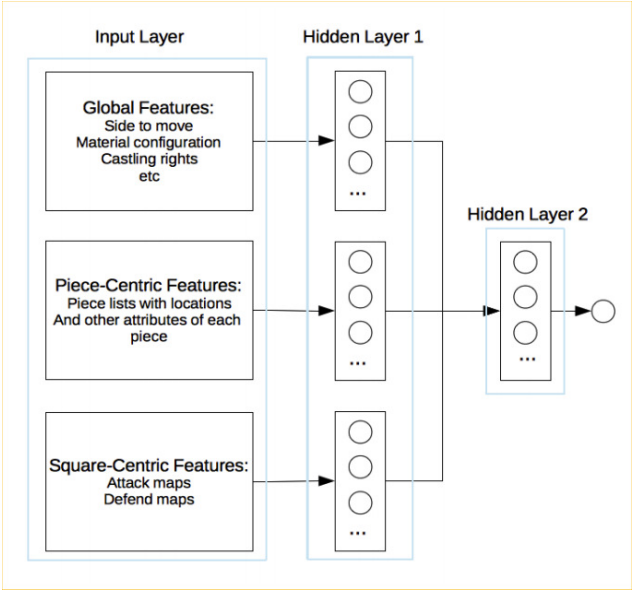
Deep Q Learning (Nature, 2015)



Aplicaciones

Juegos

Ajedrez



Matthew Lai (Imperial College London):
"Giraffe: Using Deep Reinforcement Learning to Play Chess"
http://arxiv.org/abs/1509.01549 (September 2015)

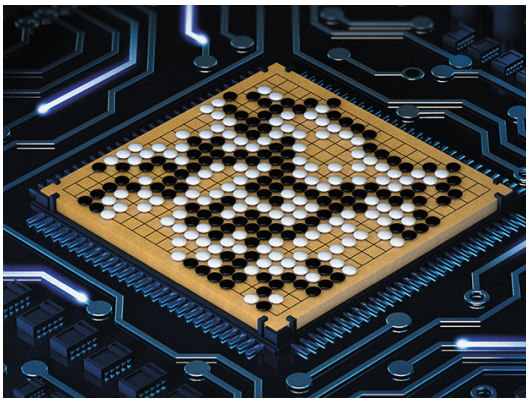
Aplicaciones

Juegos

Go: Los campeones humanos se negaban a jugar contra ordenadores porque eran demasiado malos (b>300)…

Octubre 2015, Londres:
AlphaGo (Google DeepMind) vence al campeón europeo Fan Hui [2-dan], 5-0.

Marzo de 2016, Seúl: $1M
AlphaGo (Google DeepMind) vence a Lee Sedol [9-dan], 4-1.



https://en.wikipedia.org/wiki/AlphaGo

AlphaGo

Herramientas

- Caffe (University of California, Berkeley)
  http://caffe.berkeleyvision.org/
- Theano (University of Montreal)
  http://deeplearning.net/software/theano/
- Torch (New York University), e.g. Facebook
  http://torch.ch/
- TensorFlow (Google)
  https://www.tensorflow.org/
- CNTK: Computational Network Toolkit (Microsoft)
  http://www.cntk.ai/
- DL4J: Deep Learning for Java (Skymind)
  http://deeplearning4j.org/

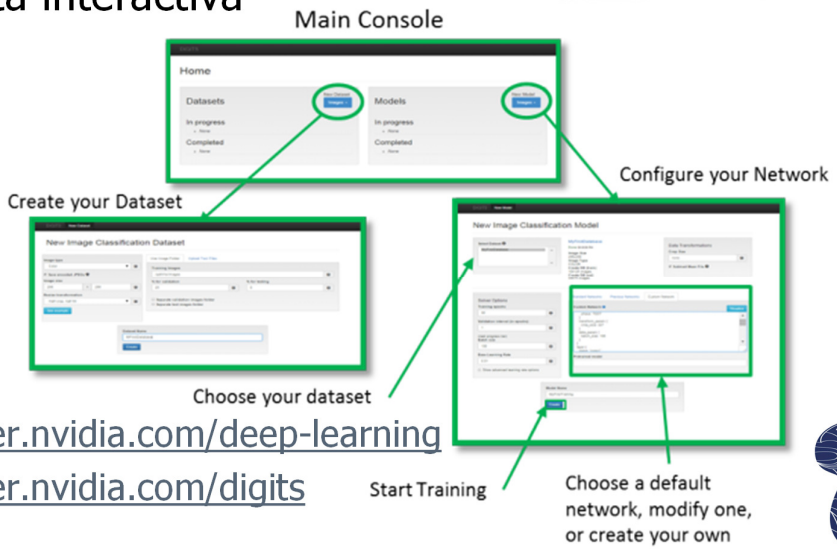
http://deeplearning.net/software_links/
https://en.wikipedia.org/wiki/Deep_learning#Commercial_activities

Herramientas

NVIDIA DIGITS:
Deep Learning GPU Training System

GPUs [Graphics Processing Units]

- Herramienta interactiva

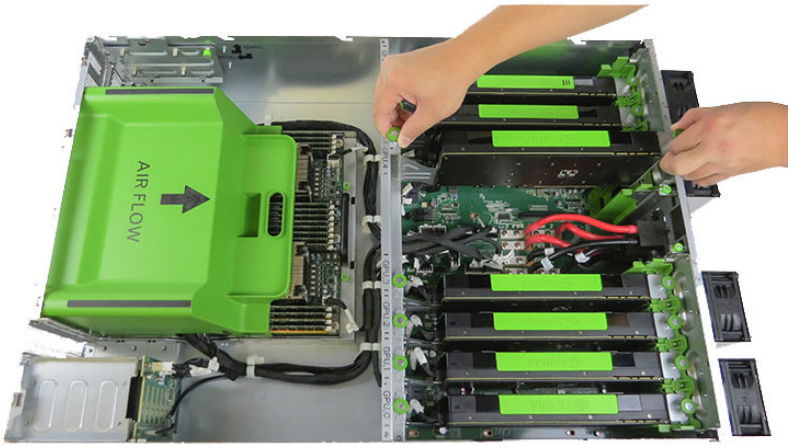


https://developer.nvidia.com/deep-learning
https://developer.nvidia.com/digits

Herramientas

Facebook Big Sur

Servidor con 8 GPUs NVIDIA Tesla para deep learning

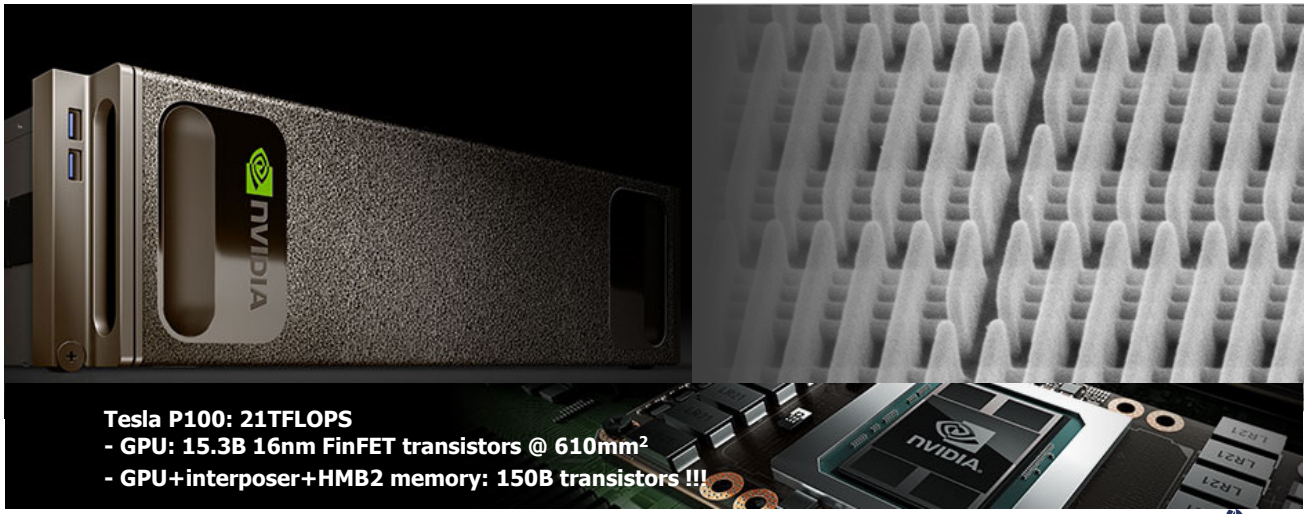


Facebook Engineering Blog, December 2015
https://code.facebook.com/posts/1687861518126048/facebook-to-open-source-ai-hardware-design/

Herramientas

NVIDIA DGX-1 deep learning supercomputer $ 129 000

8x GPUs NVIDIA Tesla P100, 28672 CUDA cores



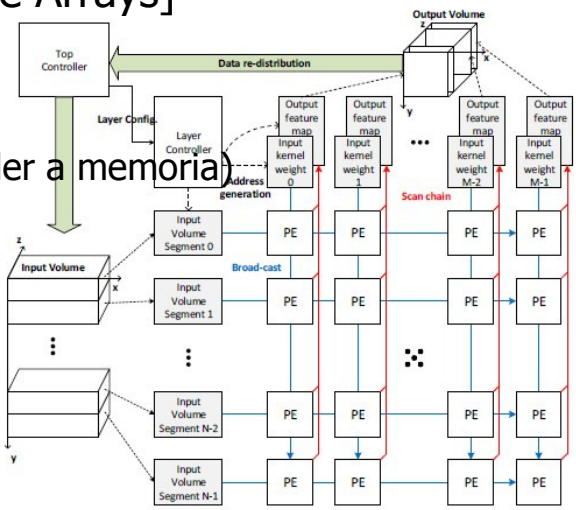
DGX-1: 8xP100, 512 GB DDR4, 4x1.92TB SSD, 170 TFLOPS, 60kg, 3200W
A $2 Billion Chip to Accelerate Artificial Intelligence, MIT Technology Review, April 2016
https://www.technologyreview.com/s/601195/a-2-billion-chip-to-accelerate-artificial-intelligence/
http://www.nvidia.com/object/deep-learning-system.html

Herramientas

Microsoft Research Catapult

FPGAs [Field Programmable Gate Arrays]

- Menor consumo de energía
- Menor ancho de banda
  (datos en la propia FPGA, sin acceder a memoria)



CPU-FPGA minimalista:
FPGA Altera Stratix V @ Open CloudServer

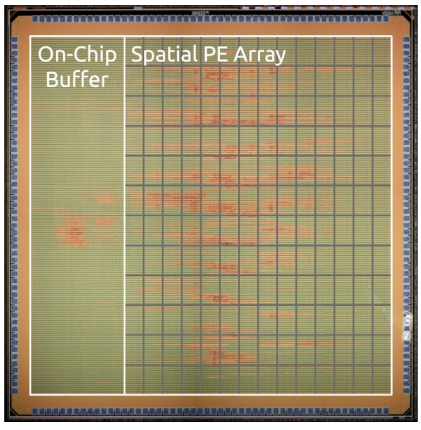
Toward Accelerating Deep Learning at Scale Using Specialized Logic
HOTCHIPS'2015: A Symposium on High Performance Chips, August 2015

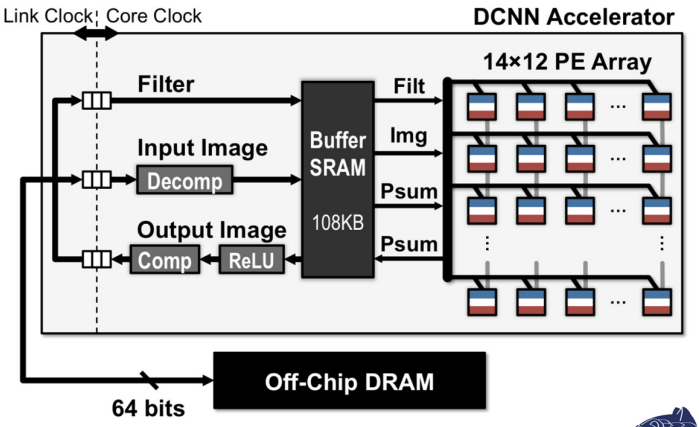
Herramientas

MIT Eyeriss

168 PE [Processing Elements], 0.3W (<10% mobile GPU)

http://www.mit.edu/~sze/eyeriss.html





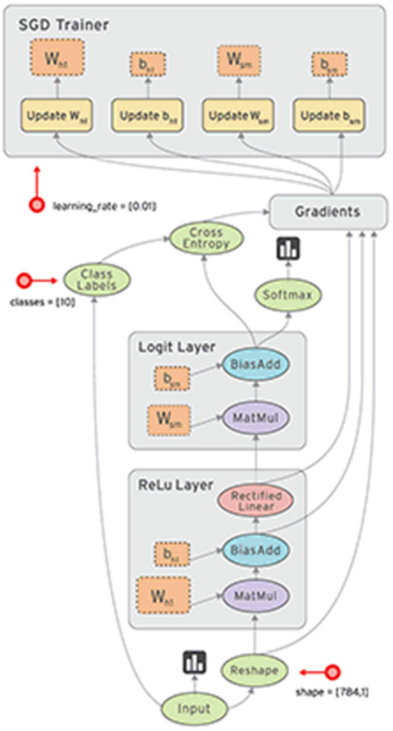
A Deep Learning AI Chip for Your Phone, IEEE Spectrum, February 2016
http://spectrum.ieee.org/tech-talk/semiconductors/processors/a-deep-learning-ai-chip-for-your-phone

Herramientas

Google TensorFlow

https://www.tensorflow.org/

Licencia Apache 2.0



Data flow graph

Herramientas

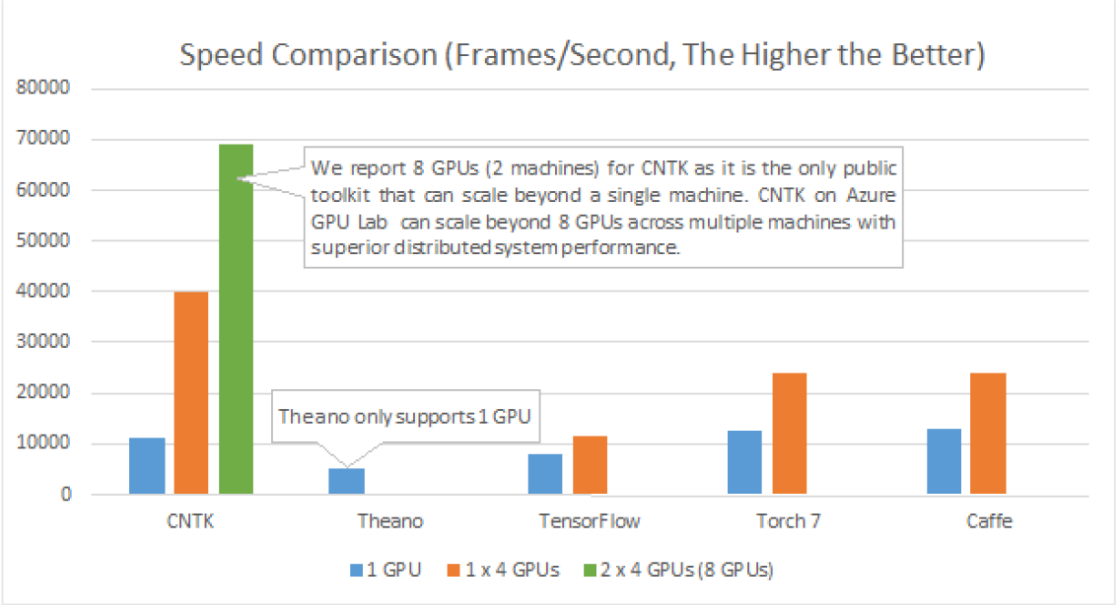
Microsoft CNTK [Computational Network Toolkit]

http://www.cntk.ai/

CNTK
compute graph

Herramientas

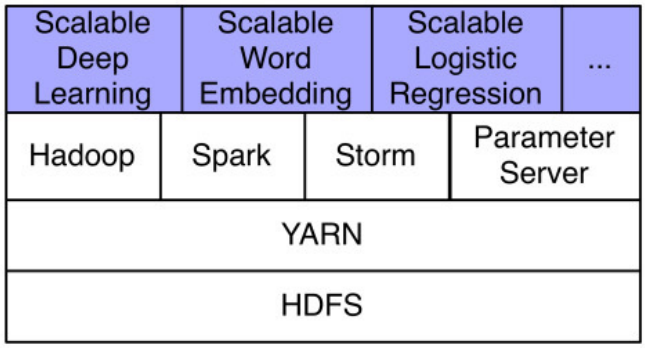
Rendimiento (enero de 2016)

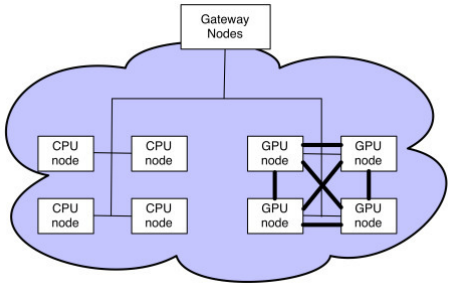


Herramientas

Yahoo Deep Learning Cluster

40 000 Hadoop nodes, 600 Petabytes, 100Gb InfiniBand





Hardware: NVIDIA Tesla GPUs

Software: Spark & Caffe

Inside Yahoo's Super-Sized Deep Learning Cluster, October 2015
http://www.datanami.com/2015/10/12/inside-yahoos-super-sized-deep-learning-cluster/

Herramientas

Servicios ofrecidos por empresas

- Ersatz Labs (cloud-based deep learning platform)
  http://www.ersatzlabs.com/
- Microsoft Azure Machine Learning
  https://azure.microsoft.com/en-us/services/machine-learning/
  CNTK Computational Network ToolKit
  https://github.com/Microsoft/CNTK
- Amazon Machine Learning
  https://aws.amazon.com/machine-learning/
  DSSTNE: Deep Scalable Sparse Tensor Network Engine
  https://github.com/amznlabs/amazon-dsstne
- IBM Watson
  https://www.ibm.com/smarterplanet/us/en/ibmwatson/
  IBM Deep Learning Framework (SystemML)
  http://systemml.apache.org/
  AlchemyAPI
  http://www.alchemyapi.com/

Herramientas

Más bibliotecas y frameworks:

- **Neon**, de Nervana Systems ➔ Intel
  (Python)
  https://github.com/NervanaSystems/neon
- **Keras**
  (Python: TensorFlow & Theano)
  http://keras.io/
- **Mocha**
  (Julia [http://julialang.org/], inspirado en Caffe [C++])
  https://github.com/pluskid/Mocha.jl
- **Apache Singa**
  (C++ & Python)
  https://singa.incubator.apache.org/

Herramientas

Más bibliotecas y frameworks:

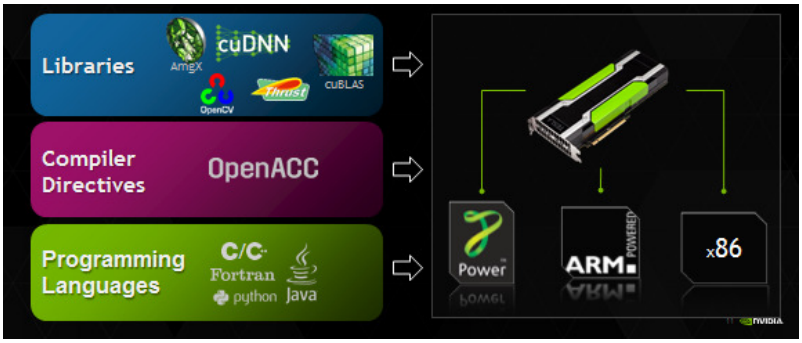
- **PaddlePaddle**, de Baidu
  [PArallel Distributed Deep Learning] (C++ & Python)
  http://www.paddlepaddle.org/
- **VELES**, de Samsung
  (Python)
  https://velesnet.ml/ & https://github.com/samsung/veles
- **Brainstorm**, del IDSIA, Lugano, Suiza
  (Python)
  https://github.com/IDSIA/brainstorm
- **Marvin**, Princeton Computer Vision & Robotics Lab
  (C/C++)
  http://marvin.is/

Herramientas

Más bibliotecas y frameworks:

cuDNN

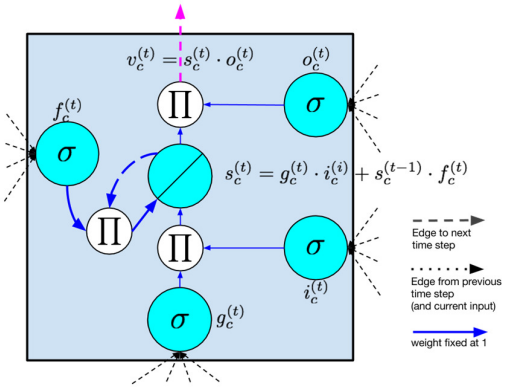
- **cuDNN**, de NVIDIA: GPU Accelerated Deep Learning (CUDA), usada por Caffe, Theano, TensorFlow & Torch
  https://developer.nvidia.com/cudnn



Más información…
https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

Deep Learning

Para algunos investigadores, que intentan obtener garantías teóricas basadas en resultados matemáticos: "[deep learning] might seem to be a regression" ;-)



LSTM (red recurrente)

En la práctica, los algoritmos con las mejores propiedades teóricas no son siempre los que mejor funcionan (sin restar importancia al estudio de las propiedades de los algoritmos de aprendizaje).

Deep Learning

Las técnicas heurísticas tienen éxito gracias a la disponibilidad de grandes conjuntos de datos (en los que el riesgo de sobreaprendizaje es menor) y la capacidad de cálculo de los sistemas actuales.

La validación con conjuntos de datos de prueba independientes ofrece una estimación de su comportamiento esperado en situaciones reales (los análisis teóricos se centran en el peor caso).

Deep Learning

Few Things Are Guaranteed

When attainable, theoretical guarantees are beautiful. They reflect clear thinking and provide deep insight to the structure of a problem. Given a working algorithm, a theory which explains its performance deepens understanding and provides a basis for further intuition. Given the absence of a working algorithm, theory offers a path of attack.

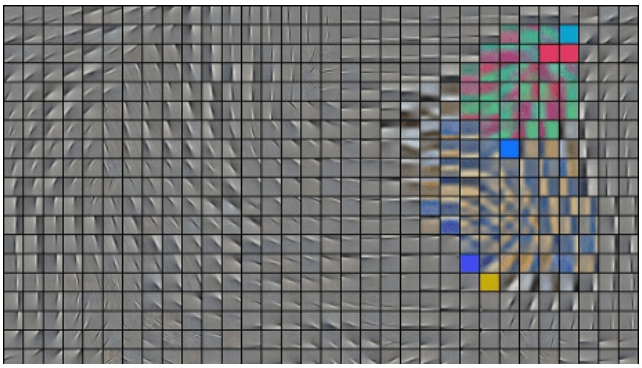
However, there is also beauty in the idea that well-founded intuitions paired with rigorous empirical study can yield consistently functioning systems that outperform better-understood models, and sometimes even humans at many important tasks. Empiricism offers a path forward for applications where formal analysis is stifled, and potentially opens new directions that might eventually admit deeper theoretical understanding in the future.

Zachary Lipton:
"Deep Learning and the Triumph of Empiricism"
KDnuggets, July 2015

Referencias

Neural Networks for Machine Learning
by Geoffrey Hinton
(University of Toronto & Google)
https://www.coursera.org/course/neuralnets

coursera



UNIVERSITY OF TORONTO

Referencias

- **Deep Learning Tutorial**
  Andrew Ng et al. (Stanford University)
  http://ufldl.stanford.edu/tutorial/

- **Deep Learning: Methods and Applications**
  Li Deng & Dong Yu (Microsoft Research)
  http://research.microsoft.com/apps/pubs/default.aspx?id=209355

- **Deep Learning**
  Joshua Bengio et al. (University of Montréal)
  http://www.iro.umontreal.ca/~bengioy/dlbook/

- **Deep Learning for Natural Language Processing**
  Richard Socher et al. (Stanford University CS224d)
  http://cs224d.stanford.edu/

Investigadores destacados

Geoffrey Hinton
(University of Toronto & Google)

Yann LeCun
(AT&T Labs → New York University → Facebook)

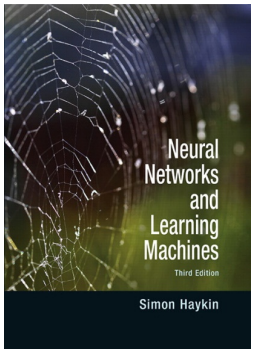
Joshua Bengio
(University of Montréal & IBM Watson)

Andrew Ng
(Stanford University → Coursera → Baidu)

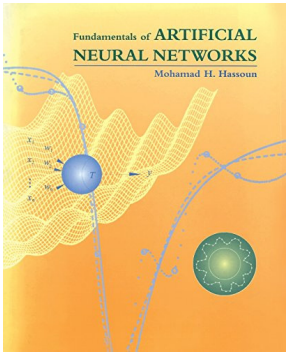
Bibliografía

Lecturas recomendadas

- Simon Haykin:
  Neural Networks and Learning Machines
  Prentice Hall, 3rd edition, 2008
  ISBN 0131471392



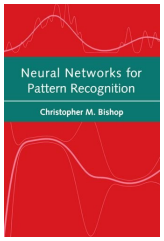
- Mohamad Hassoun:
  Fundamentals of Artificial Neural Networks
  MIT Press, 2003
  ISBN 0262514672

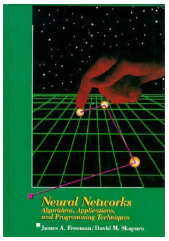


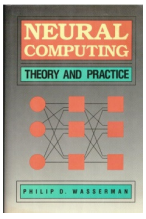
Bibliografía complementaria

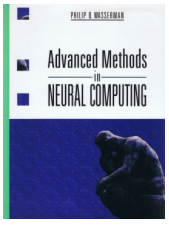
Redes neuronales artificiales

- Christopher M. Bishop:
  Neural Networks for Pattern Recognition
  Oxford University Press, 1996. ISBN 0198538642
- James A. Freeman & David M. Skapura:
  Neural Networks: Algorithms, Applications, and Programming Techniques
  Addison-Wesley, 1991. ISBN 0201513765
- Philip D. Wasserman:
  Neural Computing: Theory and Practice,
  Van Nostrand Reinhold, 1989. ISBN 0442207433
- Philip D. Wasserman:
  Advanced Methods in Neural Computing
  Van Nostrand Reinhold, 1993. ISBN 0442004613





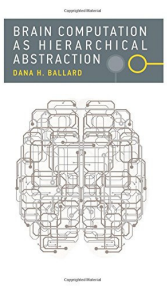


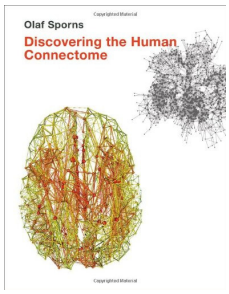


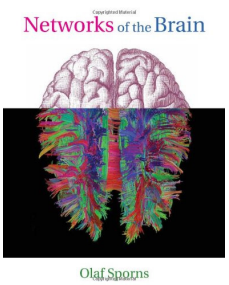
Bibliografía complementaria

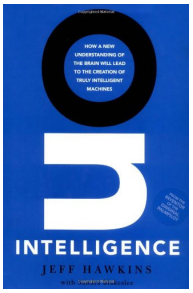
"wetware"

- Dana H. Ballard: **Brain Computation as Hierarchical Abstraction.** MIT Press, 2015. ISBN 0262028611
- Olaf Sporns: **Discovering the Human Connectome.** MIT Press, 2012. ISBN 0262017903
- Olaf Sporns: **Networks of the Brain**. MIT Press, 2010. ISBN 0262014696
- Jeff Hawkins: **On Intelligence.** Times Books, 2004. ISBN 0805074562





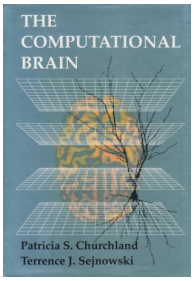


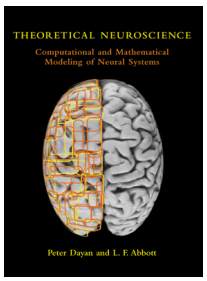


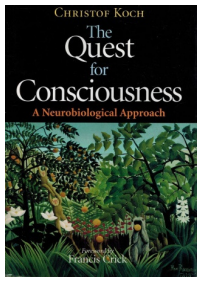
Bibliografía complementaria

Computational Neuroscience

- Patricia S. Churchland & Terrence J. Sejnowski: **The Computational Brain**. MIT Press, 1992. ISBN 0262031884
- Peter Dayan & L.F. Abbott: **Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.** MIT Press, 2001. ISBN 0262041995.
- Christof Koch: **The Quest for Consciousness: A Neurobiological Approach**. Roberts & Company Publishers, 2004. ISBN 0974707708







Bibliografía

Bibliografía en castellano

- James A. Freeman & David M. Skapura:
  Redes Neuronales: Algoritmos, aplicaciones y técnicas de programación
  Addison-Wesley / Díaz de Santos, 1993.
  ISBN 020160115X