Redes Neuronales

Eduardo Morales, Hugo Jair Escalante

INAOE

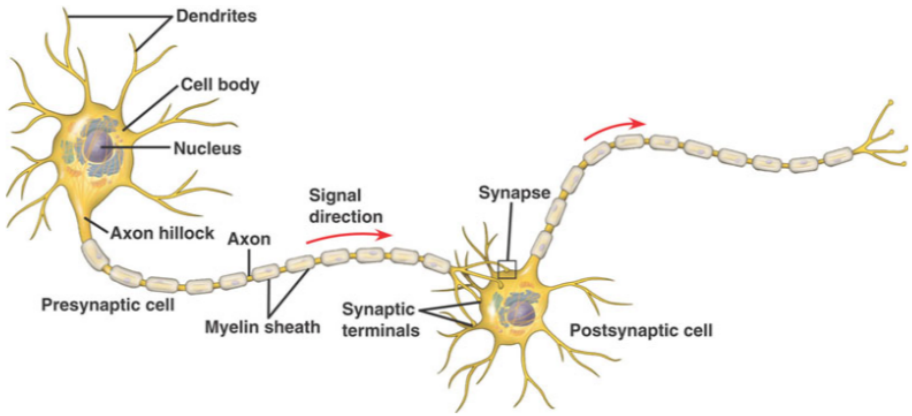
Contenido

Introducción

- A las redes neuronales (conneccionismo, proceso paralelo distribuido, computación neuronal, redes adaptivas, computación colectiva) las podemos entender desde dos puntos de vista:
  - **Computacional**: Representar funciones usando redes de elementos con cálculo aritmético sencillo, y métodos para aprender esa representación a partir de ejemplos. La repesentación es útil para funciones complejas con salidas continuas y datos con ruido
  - **Biológico:** Modelo matemático de la operación del cerebro. Los elementos sencillos de cómputo corresponden a neuronas, y la red a una colección de éstas.

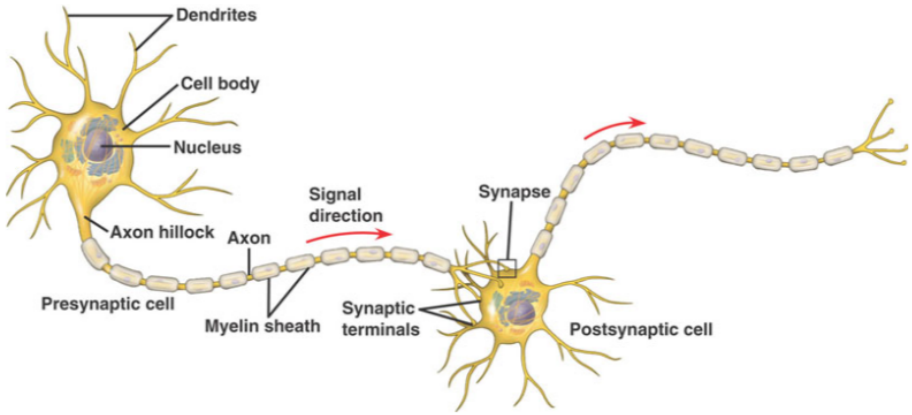
Introducción

- La neurona es la unidad funcional fundamental del sistema nervioso
- Cada neurona tiene un cuerpo (soma) que tiene un nucleo y tiene un grupo de fibras (dendritas) y una de las cuales es más larga (axón)



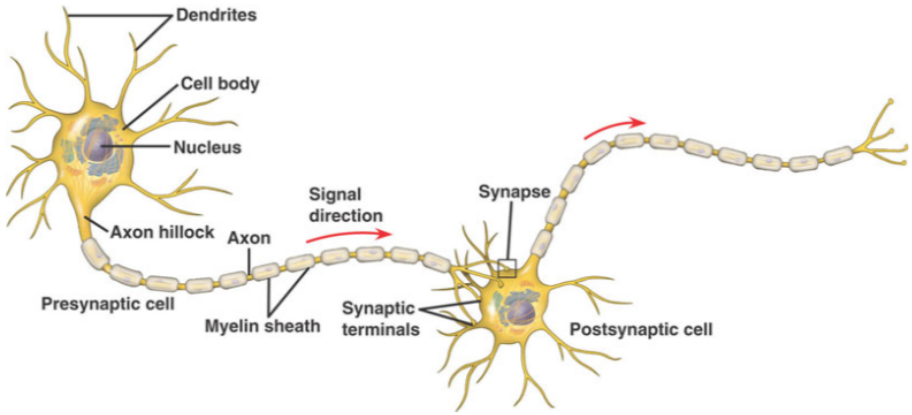
Introducción

- El axón se bifurca eventualmente en sinapses. Las señales se propagan en una reacción electroquímica complicada.
- Las substancias químicas transmisoras se liberan de las sinapses y entran a la dendrita, aumentando o disminuyendo el potencial eléctrico del cuerpo de la célula.



Introducción

- Cuando el potencial alcanza un umbral se transmite un pulso elétrico o acción potencial a través del axón. Las sinapses que aumentan el potencial se llaman exitatorias y los que disminuyen, inhibidoras.
- La conección “sináptica” es *plástica* (cambia con la estimulación).
- Se pueden formar nuevas conecciones y las neuronas migran de un lugar a otro. Esto se cree que forman la base de aprendizaje en el cerebro.



Introducción

- En general el mapeo de regiones con funciones puede ser múltiple y cambiar cuando un área es dañada (pero no se sabe bien como se hace).
- Lo sorprendente es que una colección de células simples puedan dar pensamiento, acción y conciencia (*cerebros causan mentes* (Searle 92)).

Introducción

Comparación gruesa de las capacidades computacionales de cerebros y computadoras (1994).

	Computadora	Cerebro Humano
Unidades Computacionales	1 CPU, 10^5 compuertas	10^{11} neuronas
Unidades de Almacenamiento	10^9 bits RAM, 10^{10} bits disco	10^{11} neuronas, 10^{14} sinapses
Ciclo (tiempo)	10^{-8} seg.	10^{-3} seg.
Anchobanda	10^9 bits/seg.	10^{14} bits/seg.
Actualizaciónes/seg.	10^5	10^{14}

A pesar de que una computadora es millones de veces más rápida por proceso individual, el cerebro finalmente es billones de veces más rápido

Introducción

- Una de las atracciones, es construir un mecanismo que combine el paralelismo del cerebro con la velocidad de las máquinas.
- Los cerebros son mucho más tolerantes (en 70-80 años, no se tiene que reemplazar una tarjeta de memoria, llamar al servicio o hacer *reboot*).
- La tercera atracción es su degradación gradual.

Historia

- Existió mucho desarrollo en los primeros años de la computación: McCulloch y Pitts (43), Hebb (49), Minsky (51) (primera red), Ashby (52), Rosenblatt (57) (perceptrón), Selfridge (59) (pandemonium), Widrow y Hoff (60) (adalines), Nilsson (65 - 90), Minsky y Papert (69).
- Durante 10 años prácticamente no se hizo nada.
- El resurgimiento comenzó en la decada de los 80's: Hinton y Anderson (81), Hopfield (82), Hinton y Sejnowski (83 y 86) y los dos volumens de PDP (Parallel Distributed Processing) anthology (Rumelhart *et al.* 86).

Historia (reciente)

- Durante los 95's-03's hubo otra época de oscurantismo en RNs, debido al surgimiento y popularización de las SVMs
- Las RNs tuvieron (otro) *segundo aire* a finales de la primera decada del presente siglo

Introducción

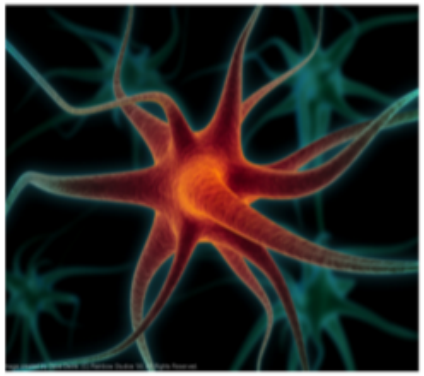
- El funcionamiento de las neuronas y del cerebro en general sirve como *inspiración* para el desarrollo de sistemas de aprendizaje computacional
- El equivalente computacional de una neurona es una *unidad* que almacena pesos asociados a un problema de aprendizaje
- Redes de neuronas imitan, de manera burda, el funcionamiento del cerebro

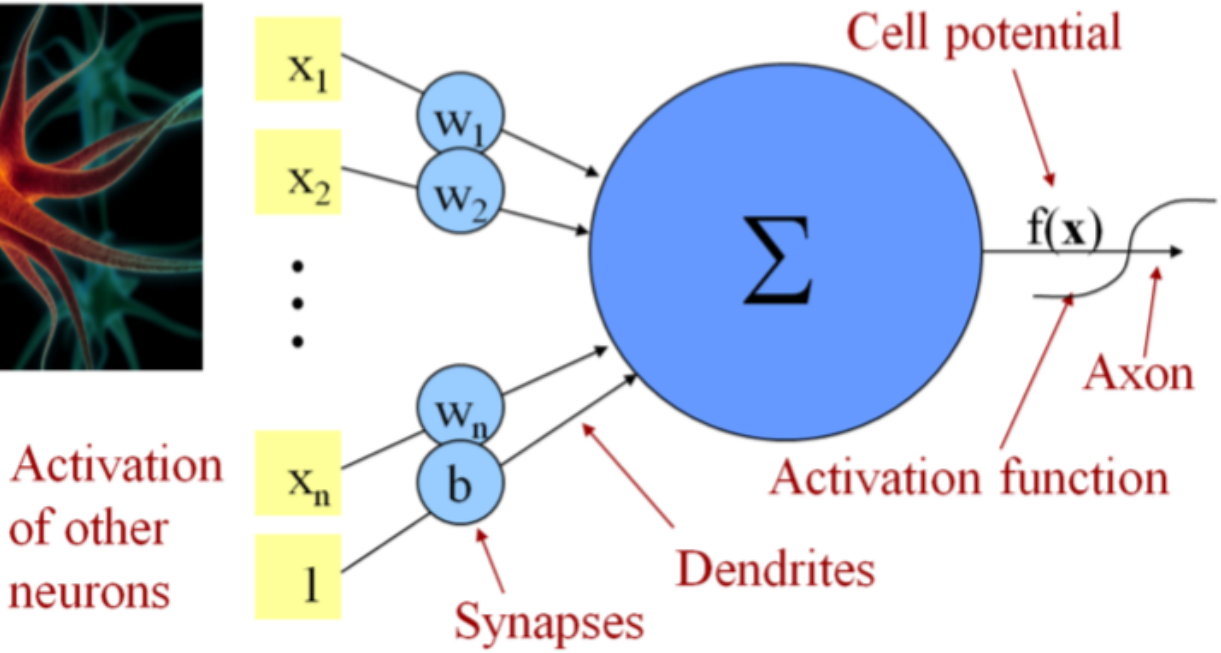
Introducción

En pocas palabras una RNA es:

- Un modelo no lineal formado por muchos modelos (unidades) lineales con funciones de activación no-lineal
- Un modelo que modifica los valores de sus elementos para hacer correspondser sus salidas con las salidas esperadas/verdaderas

Introducción





McCulloch and Pitts, 1943

$$f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x} + b$$

Neurona artificial (diapositiva I. Guyon)

Introducción

- Una red neuronal está compuesta por nodos o unidades, conectados por ligas
- Cada liga tiene un peso numérico asociado
- Los pesos son el medio principal para almacenamiento a largo plazo en una red neuronal, y el aprendizaje normalmente se hace sobre la actualización de pesos.

Introducción

- Algunas unidades están conectadas al medio ambiente externo y pueden diseñarse como unidades de entrada o salida.
- Los pesos se modifican para tratar de hacer que el comportamiento entrada/salida se comporte como el del ambiente.

Introducción

Cada unidad tiene:

- Un conjunto de ligas de entrada (provenientes de otras unidades)
- Un conjunto de ligas de salida (hacia otras unidades)
- Un nivel de activación, y
- Una forma de calcular su nivel de activación en el siguiente paso en el tiempo, dada su entrada y sus pesos (cada unidad hace un cálculo local basado en las entradas de sus vecinos)

Introducción

- La computación se hace en función de los valores recibidos y de los pesos.
- Se divide en dos:
  1. Un componente lineal, llamado la función de entrada (in_i), que calcúla la suma de los valores de entrada.
  2. Un componente no lineal, llamado función de activación (g), que transforma la suma pesada en una valor final que sirve como su valor de activación (a_i).
- Normalmente, todas las unidades usan la misma función de activación.

Introducción

- La suma pesada es simplemente las entradas de activación por sus pesos correspondientes:

$$in_i = \sum_j w_{j,i} a_j = \mathbf{w_i} \cdot \mathbf{a_i}$$

$\mathbf{w_i}$: vector de los pesos que llegan a la unidad i
$\mathbf{a_i}$: vector de los valores de activación de las entradas a la unidad i

- El nuevo valor de activación se realiza aplicando una función de activación g:

$$a_i \leftarrow g(in_i) = g(\sum_j w_{j,i} a_j)$$

Introducción

- En muchos casos, es matemáticamente conviente cambiar el umbral por un peso de entrada extra. Esto permite un elemento de aprendizaje más simple, ya que sólo hay que ajustar pesos, y no pesos y umbral.
- Una de las motivaciones iniciales en el diseño de unidades individuales fué la representación de funciones Booleanas básicas (McCulloch y Pitts, '43).
- Esto es importante, porque entonces podemos usar estas unidades para construir una red que compute cualquier función Booleana.

Ejemplo de Aplicación

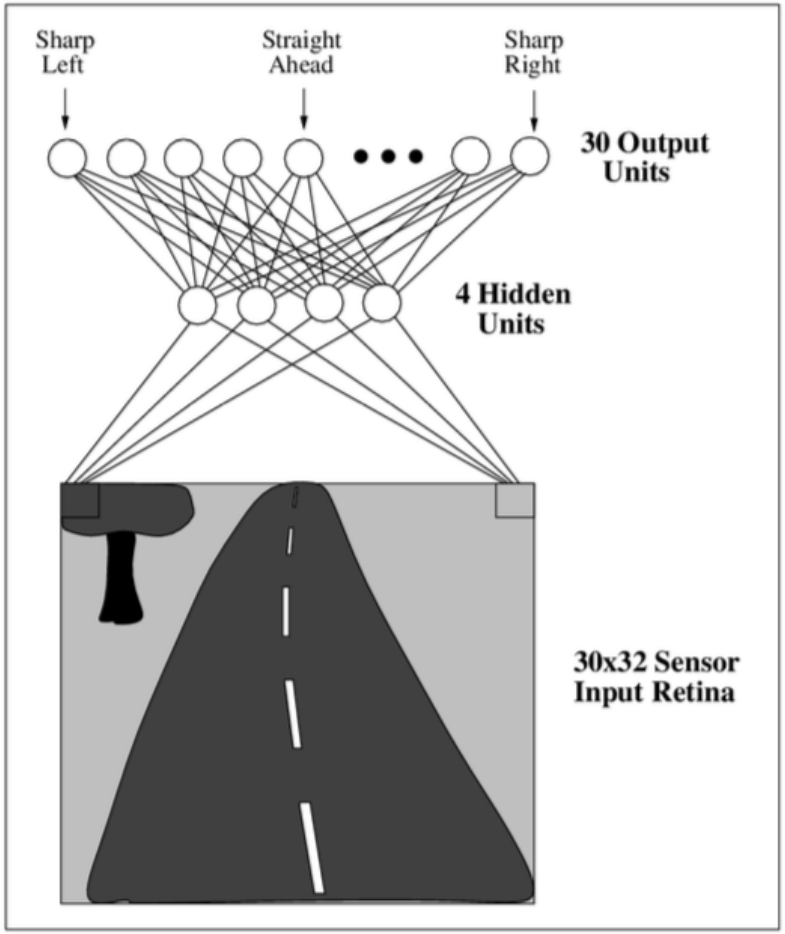


Figura: Arquitectura de Alvinn

Introducción

- En la práctica, casi todas las implementaciones de RN son en software y utilizan un control síncrono en su actualización.
- Para el diseño uno debe de decidir:
  1. número de unidades
  2. cómo se deben de conectar
  3. qué algoritmo de aprendizaje utilizar
  4. cómo codificar los ejemplos de entradas y salidas
- Cada unidad recibe señales de sus ligas de entradas y calcúla un nuevo nivel de activación que manda a través de sus ligas de salidas.

Introducción

Problemas apropiados para abordarse con RNAs

- Las instancias se representan por muchos pares atributo-valor
- La función objetivo de salida puede ser discreta, real, un vector de reales-categorías o una combinación de ambos
- Los ejemplos de entrenamiento pueden tener errores
- Se requiere uan evaluación rápida de la función aprendida
- No es importante interpretar la función aprendida

Estructuras de Redes

- Existen muchas estructuras que dan propiedades computacionales distintas.
- La distinción principal es entre:
  1. *feed-forward*: ligas unidireccionales, sin ciclos (DAGs). Normalmente estaremos hablando de redes que están arregladas en capas. Cada unidad está ligada solo con las unidades de la siguiente capa. No hay ligas inter-capas, ni ligas a capas anteriores, ni ligas saltandose capas.
  2. *recurrent*: las ligas pueden formar topologías arbitrarias.

Feed-Forward

- Una red *feed-forward* calcula una función de las entradas que depende de los pesos. Este es el modelo más usado y nos vamos a concentrar más en éste.
- Por un lado, están las unidades de entrada (su valor de activación depende del medio ambiente). Del otro, las unidades de salida. En medio (sin conección al medio ambiente) se tienen las unidades ocultas (ver figura 2).

Feed-Forward

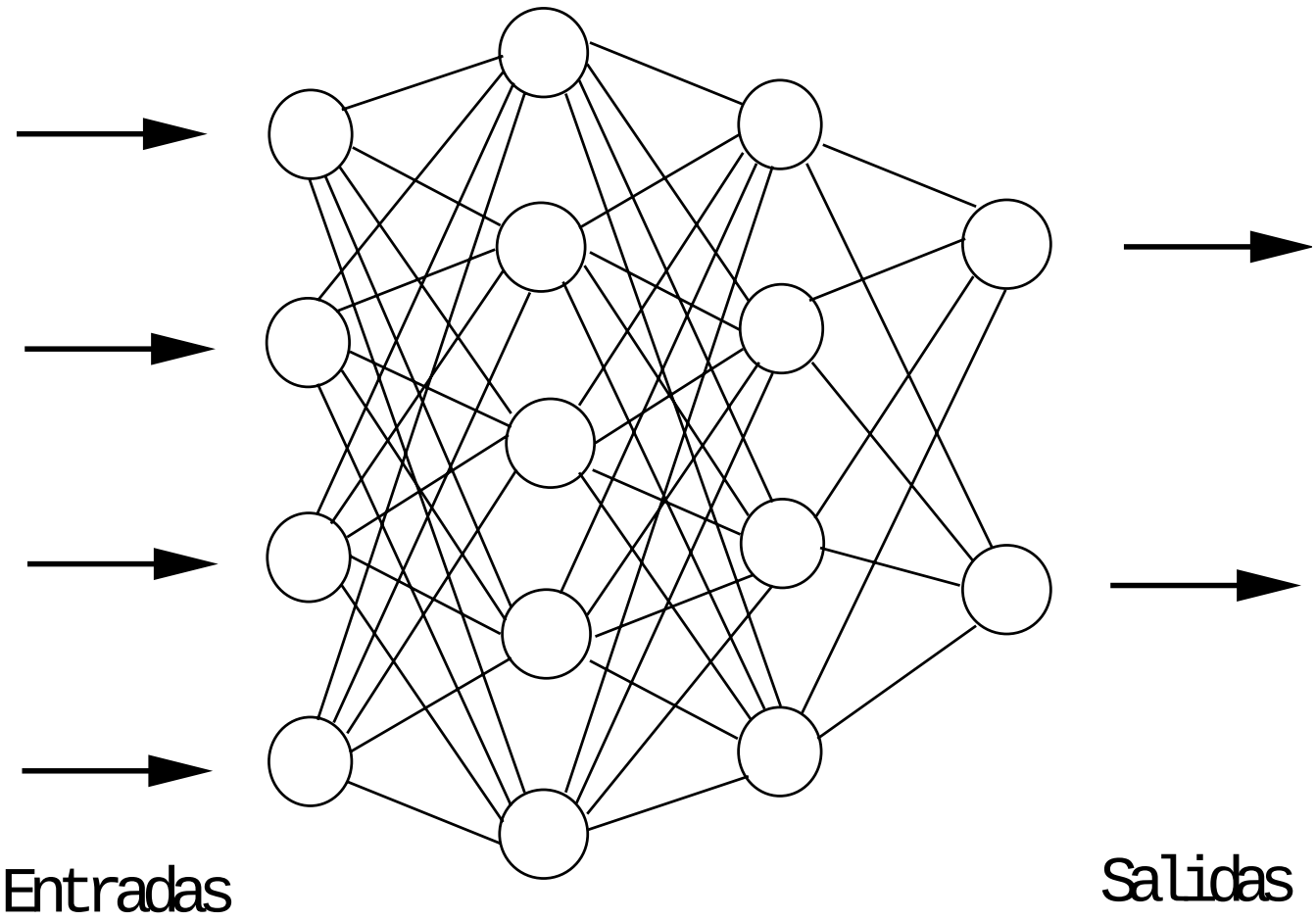


Figura: Arquitectura típica de una Red Neuronal *feedforward*.

Feed-Forward

- Algunas redes no tienen nodos o unidades ocultos (*perceptrones*)
- Esto hace el aprendizaje mucho más sencillo, pero limita lo que se puede aprender
- Redes con una o mas capas ocultas se llaman redes multicapas.

Feed-Forward

- Con una sola capa (suficientemente grande) de unidades ocultas, es posible representar cualquier función continua de las entradas. Con dos capas es posible representar hasta funciones discontinuas.
- Con una estructura fija y función de activación *g* fija, las funciones representables por una red *feed-forward* están restringidas por una estructura específica parametrizada.

Feed-Forward

- Los pesos escogidos para la red determinan cuáles de las funciones se representan.
- Por ejemplo, una red con 2 unidades de entrada, dos ocultas y una de salida, con todas las conecciones intercapas, calcula la siguiente función (ver figura 3):

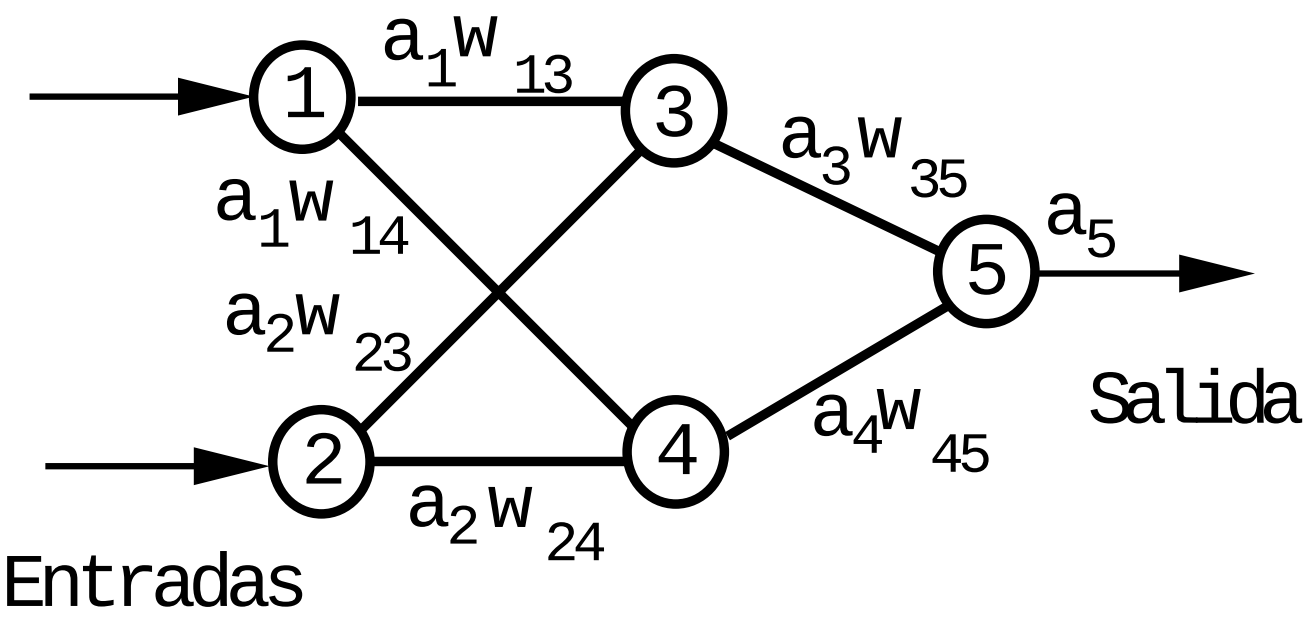


Figura: Arquitectura de una Red Neuronal simple.

Feed-Forward

- La salida es:

$$a_5 = g(w_{3,5}a_3 + w_{4,5}a_4)$$

$$= g(w_{3,5}g(w_{1,3}a_1 + w_{2,3}a_2) + w_{4,5}g(w_{1,4}a_1 + w_{2,4}a_2))$$

- Como g es una función no lineal, la red representa una función nolineal compleja.
- Si se piensa que los pesos son los parámetros o coeficientes de esta función, el aprendizaje es simplemente el proceso de "afinar" los parámetros para que concuerden con los datos en el conjunto de entrenamiento (es lo que en estadística se llama regresión nolineal).

Perceptrón

- *Feed-forward* se estudiaron desde los 50's llamandose perceptrones. A pesar de que se estudiaron varias configuraciones, la única con una regla efectiva de aprendizaje en aquel tiempo fué la de una sola capa.
- Cada salida es independiente de las otras, cada peso sólo afecta una de las salidas, por lo que se pueden estudiar independientemente (i.e., ver sólo una salida a la vez).
- La activación de salida es:

$$O = \text{escalón}_0(\sum_j W_j I_j) = \text{escalón}_0(W \cdot I)$$

Perceptrón

- Lo que se puede representar son funciones Booleanas sencillas como, AND, OR or NOT. También otras no tan simples como: función mayoritaria (lo cual requiere un árbol de decisión de $O(2^n)$ nodos).
- Sin embargo, están muy limitados en las funciones que pueden representar. El problema, es que cada entrada I_j sólo puede influir la salida final en una dirección, sin importar los otros posibles valores de la salida.
- Lo que quiere decir es que si tenemos una entrada a_j que vale 0 cuando la salida vale 0 y vale 1 cuando la salida vale 1, no podemos tener otra entrada b_j que valga 1 cuando la salida valga 0 y viceversa.
- Lo que podemos representar son funciones linealmente separables (i.e., AND, OR, pero no XOR).

Preceptrón

- Esto se obtiene directamente de la ecuación que representa. Para *n* entradas, se vuelve más difícil visualizar la separación lineal.
- Existe un algoritmo del perceptrón que puede aprender cualquier función linealmente separable, dado un conjunto adecuado de ejemplos.
- La mayoría de los algoritmos de las redes neuronales, hacen pequeños ajustes en los pesos para reducir la diferencia entre los observado y lo predicho.
- La diferencia, con otro sistema de aprendizaje, es que éste se realiza varias veces para cada ejemplo.

Algoritmo de Aprendizaje

Función aprendizaje-red-neuronal (ejemplos)
red $\leftarrow$ una red con pesos asignados aleatoriamente
repeat
para cada $e \in$ *ejemplos* do
    $o \leftarrow$ salida de la red neuronal(red,e)
    $t \leftarrow$ valor observado de e
    Actualiza los pesos en la red con base en e, o y t
    end
until todos los ejemplos sean predichos correctamente o se alcance un criterio de paro
regresa red

Peceptrón

- Para los perceptrones la regla de actualización de pesos es más o menos sencilla:
  - Si lo predicho es o y lo real es t, el error es: $err = t - o$.
  - Si el error es positivo, aumenta o, y si es negativo decrece o.

Gradiente descendiente y la regla Delta

- El gradiente descendiente trata de encontrar los pesos que mejor se ajustan a los ejemplos y es la base del algoritmo de retro–propagación (*backpropagation*).
- El error lo podemos expresar por diferencias de error al cuadrado de la siguiente forma:

$$E(W) = \frac{1}{2}\sum_i (t_i - o_i)^2$$

- Lo que queremos es determinar el vector de pesos que minimice el error E
- Esto se logra alterando los pesos en la dirección que produce el máximo descenso en la superficie del error

Gradiente

- La dirección de cambio se obtiene mediante el gradiente. El gradiente nos especifica la dirección que produce el máximo incremento, por lo que el mayor descenso es el negativo de la dirección.
- La regla de actualización de pesos es entonces:

$$W \leftarrow W + \Delta W$$

$$\Delta W = -\alpha \nabla E$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ &= \sum_{d \in D} (t_d - o_d)(-x_{i,d}) \end{aligned}$$

Gradiente

- Por lo que:

$$\Delta w_i = \alpha \sum_{d \in D} (t_d - o_d) x_{i,d}$$

- En la práctica, se tiende a usar un gradiente descendiente incremental. Esto es, en lugar de procesar el error sobre todos los datos, se hace sobre uno solo.
- En este caso, la regla de actualización es:

$$\Delta w_i = \alpha (t - o) x_i$$

- La cual es también conocida como la regla delta, LMS (*least-mean-square*), Adeline ó Widrow–Hoff.

Gradiente

- Rosenblatt la propuso en 1960 y probó que usando está regla, se convergía a los pesos correctos, mientras la función fuera linealmente separable.
- El teorema de convergencia, creó gran expectación, hasta que en 1969 Minsky y Papert, hicieron lo que quizas se debió haber hecho desde el principio. Analizar la clase de funciones representables (en su libro Perceptrons).
- El resultado no debería de ser tan sorprendente, ya que en efecto está haciendo una búsqueda de gradiente descendente en el espacio de pesos. Se puede ver que el espacio de pesos no tiene un mínimo local.

Redes Multicapas

- Rosenblatt y otros se concentraron en una sola capa, por no encontrar un método adecuado de actualizar los pesos entre las entradas y las unidades ocultas, cuando el error se calcula en las unidades de salida.
- Minsky y Papert dijeron que investigar multicapas era un problema de importancia, pero especularon que no había razón para suponer que alguna de las virtudes de los perceptrones (teorema de regla de aprendizaje) se mantuvieran con multicapas y que su extensión sería esteril.
- En parte tuvieron razón, pero definitivamente no ha sido esteril. Aprendizaje en multicapas no es eficiente ni garantiza converger al óptimo global. El aprender funciones generales a partir de ejemplos es un problema intratable en el peor de los casos.

Redes Multicapas

Se obtienen modelos diferentes cambiando g. Las opciones comunes son:

- Función escalón:

$$escalon_t(x) = \begin{cases} 1, & \text{si } x \geq t \\ 0, & \text{si } x < t \end{cases}$$

- Signo:

$$signo(x) = \begin{cases} +1, & \text{si } x \geq 0 \\ -1, & \text{si } x < 0 \end{cases}$$

- Sigmoide:

$$sigmoide(x) = \frac{1}{1 + e^{-x}}$$

Redes Multicapas

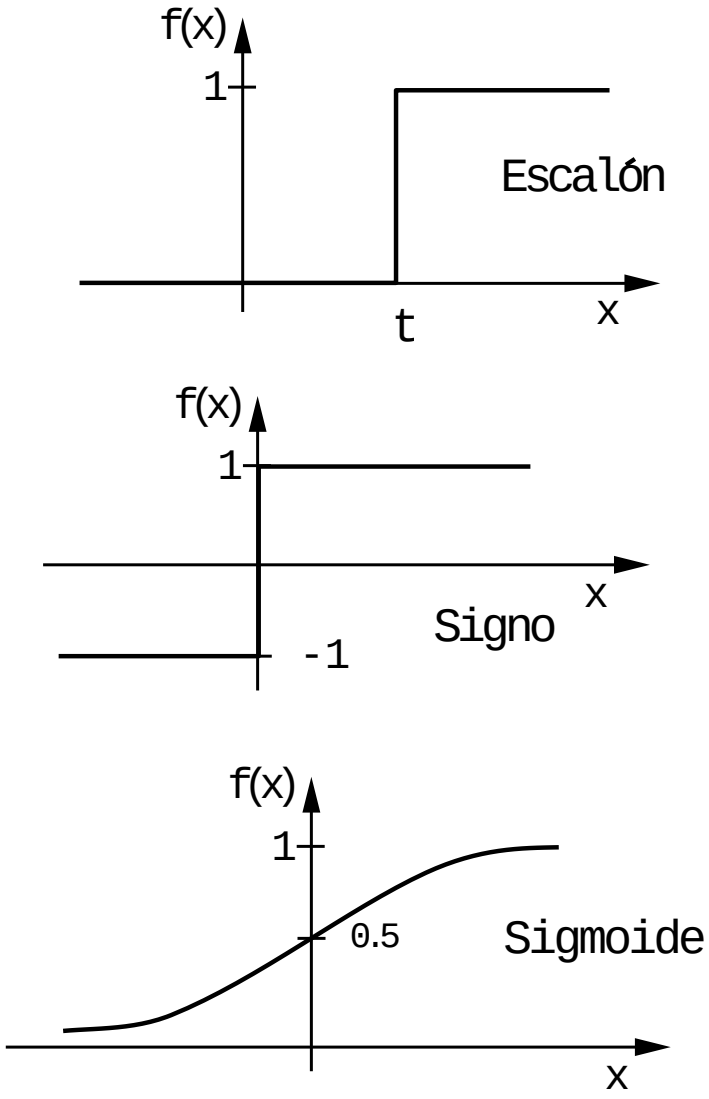


Figura: Funciones de activación comunes para Redes

Backpropagation

- El método más popular de multicapas es el de retro-propagación (*back-propagation*).
- Se publicó originalmente en 1969 por Bryson y Ho, pero fué ignorado hasta mediados de los 80's.
- Aprender en una red multicapas es muy parecido a un perceptrón. Si existe un error se ajustan los pesos para reducir el error.
- El truco es dividir la culpa del error entre los pesos contribuyentes. Como en el perceptrón se trata de minimizar el error (en este caso, el cuadrado del error).

Backpropagation

- En la capa de salida, la actualización es muy parecida a la del perceptrón. Las diferencias son:
  - se usa la activación de la unidad oculta a_i en lugar de la de entrada
  - la regla contiene un término para el gradiente de la función de activación

Notación

Notación:

- x_{ij} = la i-ésima entrada al nodo j
- w_{ij} = el peso asociado a la i-ésima entrada del nodo j
- $net_j = \sum_i w_{ij} x_{ij}$ (suma pesada de entradas al nodo j)
- o_j = la salida del nodo j
- t_j = la salida esperada del nodo j
- σ = función sigmoide
- sal = el conjunto de nodos de salida
- α = razón de aprendizaje.
- $sal(j)$ = conjunto de nodos cuyas entradas directas incluyen la salida del nodo j

Algoritmo de Retropropagación

(un solo paso un solo ejemplo)

1. Propaga las entradas a través de la red y calcula la salida
2. Propaga el error hacia atrás
   1. para cada unidad de salida k, calcula su error δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

   2. Para cada unidad oculta h, calcula su error δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in sal(h)} w_{hk}\delta_k$$

   3. Actualiza los pesos w_{ij}

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \qquad \text{donde} \qquad \Delta w_{ij} = \alpha \delta_j x_{ij}$$

Desarrollo

- Lo que queremos calcular es la actualizacón de los pesos w_{ij} sumandole Δw_{ij}

$$\Delta w_{ij} = \alpha \frac{\partial E_d}{\partial w_{ij}}$$

$$\frac{\partial E_d}{\partial w_{ij}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ij} = \delta_j x_{ij}$$

Capa de Salida

- Para la capa de salida:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in sal}(t_k - o_k)^2$$

- La derivada es cero en todos los casos, excepto cuando $k = j$, por lo que:

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= -(t_j - o_j)$$

Capa de Salida

- Como $o_j = \sigma(net_j)$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

- que es la derivada de la sigmoide:

$$= \sigma(net_j)(1 - \sigma(net_j)) = o_j(1 - o_j)$$

- Por lo que:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1 - o_j)$$

- y finalmente:

$$\Delta w_{ij} = -\alpha \frac{\partial E_d}{\partial w_{ij}} = \alpha(t_j - o_j)o_j(1 - o_j)x_{ij}$$

Capa Oculta

- Si j es un nodo oculto, ahora en la regla de actualización del peso w_{ij} se debe de considerar las formas indirectas en las que pudo contribuir al error (de alguna forma estamos distribuimos el error), por lo que consideramos todos los nodos a los cuales les llega la salida del nodo oculto j.
- Vamos a denotar: $\delta_i = -\frac{\partial E_d}{\partial net_i}$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in sal(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$\delta_j = \sum_{k \in sal(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$\delta_j = \sum_{k \in sal(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

Capa Oculta

- $\frac{\partial net_k}{\partial o_j}$ es diferente de cero, sólo cuando tenemos el término $w_{jk} \cdot x_{jk}$ (donde $x_{jk} = o_j$) en la sumatoria, por lo que:

$$\delta_j = \sum_{k \in sal(j)} -\delta_k w_{jk} \frac{\partial o_j}{\partial net_j}$$

$$\delta_j = \sum_{k \in sal(j)} -\delta_k w_{jk} o_j (1 - o_j)$$

$$\delta_j = o_j (1 - o_j) \sum_{k \in sal(j)} -\delta_k w_{jk}$$

- Lo que corresponde a la fórmula del inciso 2(b). Finalmente:

$$\Delta w_{ij} = \alpha \delta_j x_{ij}$$

Backpropagation

- La retro-propagación puede ser visto como búsqueda de gradiente descendente en la superficie del error.
- La retro-propagación nos da una forma de dividir el cálculo del gradiente entre las unidades, con lo que el cambio en cada peso puede calcularse por la unidad al cual el peso está ligado, usando sólo información local.
- Como cualquier gradiente descendiente tiene problemas de eficiencia y convergencia, sin embargo, es un paso para pensar en paralelizar.
- Tip: para calcular el error observado, se tiene que calcular una salida. Durante este cálculo es conveniente salvar algunos de los resultados intermedios (en particular el gradiente de activación $g'(in_i)$ en cada unidad), lo cual acelera la fase de retro-propagación.

Redes Recurrentes

- Las redes de Hopfield son probablemente las mejor entendidas de redes recurrentes
- Tienen conecciones bidireccionales con pesos simétricos (i.e., $W_{i,j} = W_{j,i}$)
- Todas las unidades son tanto unidades de entrada como de salida. La función de activación es la función signo, y los valores de activación pueden ser sólo ± 1.

Redes de Hopfield

- Una red de Hopfield funciona como una memoria asociativa.
- Despues de entrenarse con un conjunto de ejemplos, un nuevo estímulo causa la red a “asentarse” en un patrón de activación correspondiente al ejemplo de entrenamiento que se parece más al nuevo estímulo.
- Uno de los resultados teóricos interesantes es que una red de Hopfield puede almacenar en forma confiable hasta: $0{,}138N$ ejemplos de entrenamiento (donde N es el número de unidades de la red).

Máquinas de Boltzmann y Recocido Simulado

- Las Máquinas de Boltzmann también usan pesos simétricos, pero incluyen unidades que son ni de entrada ni de salida
- Usan una función de activación *estocástica*, tal que la probabilidad de que la salida sea 1 es una función de la suma total de los pesos.
- Las máquinas de Boltzmann siguen una transición de estados que se parece a la búsqueda de recocido simulado (*simulated annealing*), para encontrar la configurar que mejor se ajusta al conjunto de entrenamiento.

Algloritmo Recocido Simulado

Función: simulated-annealing (problema, agenda)
Entrada: problema, agenda (mapeo del tiempo a "temperatura")
Usa: nodo actual, nodo siguiente y T (temperatura)
(controla la probabilidad de pasos hacia abajo)

nodo actual $\leftarrow$ crea-nodo(estado-inicial[problema])
for $t \leftarrow 1$ a ∞ do
$T \leftarrow agenda(t)$
if $T = O$ regresa nodo actual
siguiente nodo $\leftarrow$ un sucesor (de nodo actual)
seleccionado aleatoriamente
$\Delta E \leftarrow$ valor(siguiente) $-$ valor(actual)
if $\Delta E > 0$ *then* actual $\leftarrow$ siguiente
else actual $\leftarrow$ siguiente, solo con probabilidad $e^{\frac{\Delta E}{T}}$

Recocido Simulado

- Como hill-climbing, pero el siguiente nodo se escoge en forma aleatoria.
- Si el movimiento mejora, lo toma, sino lo toma con cierta probabilidad.
- La probabilidad está determinada por la temperatura.
- La idea es ir reduciendo gradualmente la temperatura.
- Si se hace lo suficientemente lento, se llega a la configuración perfecta.

Mapas Autorganizados de Kohonen

- Es una alternativa para formar *clusters* o grupos, a partir de datos, cuando se conoce el número de grupos a formar.
- Se ajustan los pesos de un vector de entrada a nodos de salida organizados en una malla bidimensional (ver figura 5).

Mapas Autorganizados de Kohonen

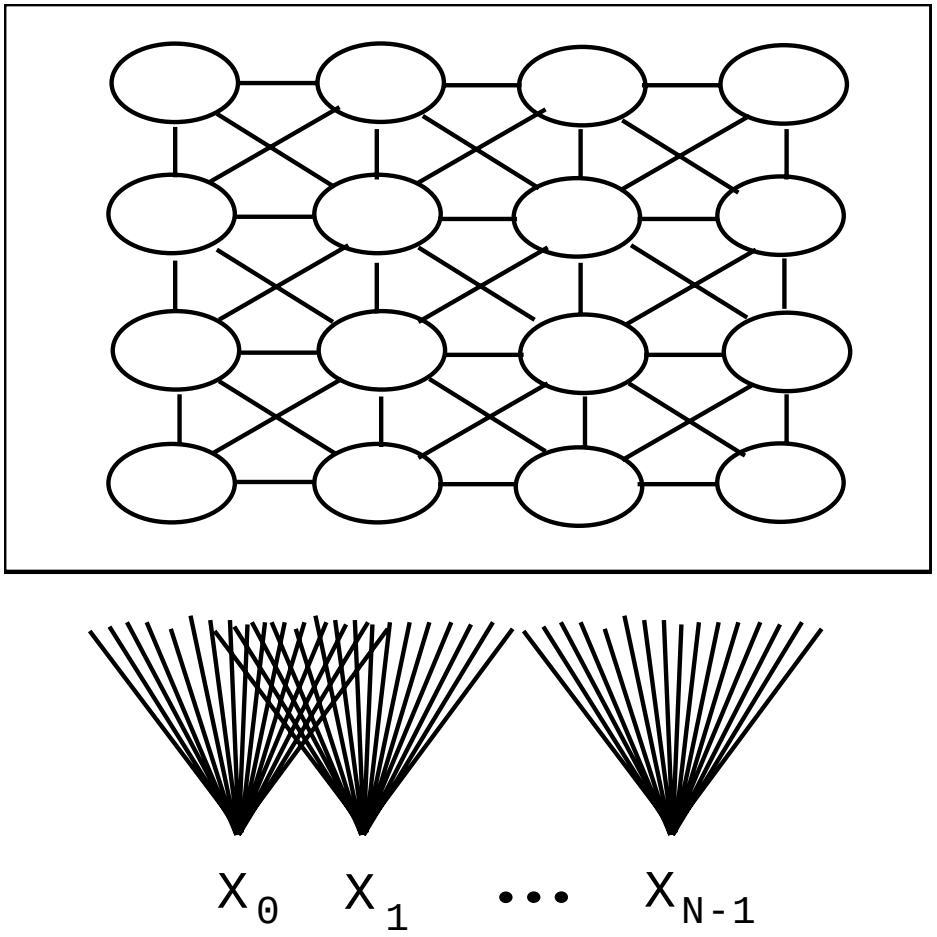


Figura: Red típica de Kohonen.

Introducción

- Se requiere definir una vecindad alrededor de cada nodo, la cual se va reduciendo con el tiempo (ver figura 6).

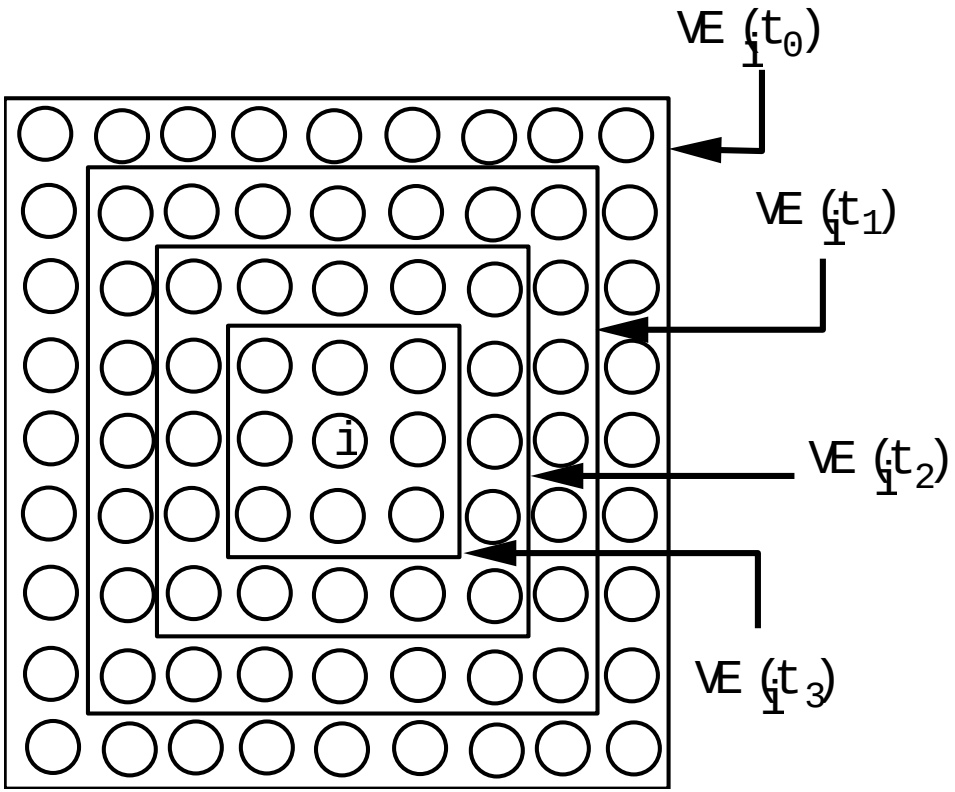


Figura: Vecindad en una red de Kohonen.

Algoritmo de Kohonen

Inicializa aleatoriamente los pesos con valores bajos de las N entradas a las M salidas.
Dada una nueva entrada:

1. Calcula la distancia entre la entrada y cada nodo de salida j:

$$d_j = \sum_{i=0}^{N-1} (X_i(t) - W_{ij}(t))^2$$

   donde $X_i(t)$ es la i-ésima entrada al tiempo t y W_{ij} es el peso del nodo i al nodo de salida j
2. Selecciona el nodo con la distancia menor (j^*)
3. Actualiza los pesos del nodo j^* y los de sus vecinos, de acuerdo al esquema de vecindad (ver figura 6):

$$W_{ij}(t+1) = W_{ij}(t) + \alpha(t)(X_i(t) - W_{ij}(t))$$

   donde $\alpha(t)$ $(0 < \alpha(t) < 1)$ decrese con el tiempo.

Estructuras Dinámicas

- Hasta ahora hemos visto, el desempeño con estructuras fijas. Esto es una debilidad, ya que una mala selección puede acarrear mal comportamiento.
- Una red muy chica, puede ser incapaz de representar lo que queremos. Una muy grande, puede memorizar todos los ejemplos, sin producir las generalizaciones que buscamos.
- Esto es, como en todos los modelos estadísticos, las redes neuronales son capaces de hacer “overfitting” cuando existen demasiados parámetros (i.e., pesos) en el modelo.

Estructuras Dinámicas

- Todavia no se tiene una teoría que nos caracterice el tipo de funciones que se pueden aproximar con un número reducido de unidades.
- Algo que se ha utilizado para encontrar una buena estructura es usando algoritmos genéticos (para que busquen estructuras). Pero es computacionalmente caro.
- Otro método es usar un tipo de hill-climbing. Posibilidades:
  - Empezar con una red grande e irla reduciendo
  - Empezar con una pequeña e irla aumentando

Estructuras Dinámicas

- Ejemplo de la primera opción: *optimal brain damage* eliminan pesos de una red completamente conectada inicialmente. Despues de un entrenamiento inicial, se sigue un método basado en teoría de la información para seleccionar de manera óptima las conecciones que se pueden eliminar (i.e., los pesos se ponen en 0).
- Entonces la red se re-entrena, y si mejora, se mantiene y el proceso se repite. Este proceso elimina hasta 3/4 partes de los pesos y mejora el desenpeño en los datos.
- Ejemplo de la segunda opción: *tiling algorithm* es parecido a la construcción de un árbol de decisión. Se van aumentando poco a poco los nodos. Se puede usar *cross-validation* para decidir si la red es del tamaño adecuado.

Discusión

- **Expresividad**: las redes neuronales son una representación atributo-valor y no tienen el poder expresivo de representaciones lógicas generales.
- Son adecuadas para entradas y salidas continuas (lo cual no es adecuado para árboldes de decisión).
- La clase de multicapas, puede representar cualquier función de un conjunto de atributos, pero una red particular puede tener muy pocas unidades.
- En teoría se requieren $2^n/n$ unidades ocultas para representar cualquier función Booleana de n entradas.
- En la práctica, se requieren mucho menos (el diseño de una buena topología sigue siendo “arte”).

Discusión

- **Eficiencia computacional** (entrenamiento): Si hay m ejemplos y $|W|$ pesos, cada paso se toma: $O(m \mid W \mid)$ tiempo.
- Sin embargo, en el peor de los casos, el número de pasos puede ser exponencial con el número de entradas.
- En la práctica es variable y se han sugerido varias técnicas usando parámetros a ajustar.
- El mínimo local es un problema.
- Con un costo computacional alto se puede usar recocido simulado.

Discusión

- **Generalización**: han tenido resultados buenos en problemas reales
- **Sensibles al ruido**: Como lo que se hace es básicamente una regresión no-lineal, son muy tolerantes al ruido (aunque no dan una noción de probabilidad)
- **Transparencia**: son escencialmente cajas negras, aunque recientemente se han obtenido reglas a partir de redes neuronales.
- Conocimiento del dominio: difícil de incorporar

Discusión

Se han sugerido otras heurísticas para evitar caer en mínimos locales:

- Decaimiento de pesos *weight decay*: decrementar todos los pesos por un cierto factor en cada interacción. La motivación es de mantener los pesos más o menos bajos.
- Entrenar varias redes con dos posibilidades: (i) quedarse con la mejor, (ii) hacer votación de todas
- Validar el error con respecto a un conjunto de prueba. Mantener una copia de una red con los mejores pesos hasta el momento. Cuando el error en el de prueba empieza a empeorar, deten el proceso y reportar la mejor copia.

Discusión

- Si existen pocos datos, hacer validación cruzada con k–subconjuntos. Una posibilidad es con cada conjunto de prueba estimar el número de interación que dió el mejor resultado. Promediar las k mejores y usarla para entrenar la red con todos los datos.
- Las unidades en las capas ocultas tienen la capacidad de crear nuevos conceptos intermedios, lo cual permite que se “introduzcan” nuevos atributos

Aplicaciones

- Pronunciación: NETtalk (87), aprende a pronunciaer texto escrito. 29 unidades de entrada (26 letras, más espacios, puntos, comas, ...). 80 unidades ocultas. 1024-palabras de entrenamiento y 95 % de éxito en el entrenamiento, pero 78 % en la prueba.

Aplicaciones

- Reconocimiento de caracteres: una de las aplicaciones más grandes de redes neuronales actuales (Le Cun et al. 89). Una red que lee códigos postales en cartas escritas a mano. El sistema tiene un prepocesador para localizar los números, y la red los descifra. 3 capas ocultas (768, 192 y 30 unidades cada una). No se conectaron todas las unidades contra todas, si no que se dió un efecto de *detectores de atributos*, dividiendo las unidades ocultas en grupos (un total de 9,760 conecciones). Logra un 99 % de éxito, adecuado para un sistema de correo automático y se ha implementado en un chip.

Aplicaciones

- Manejar: ALVINN (Autonomous Land Vehicle In a Neural Network) (Pomerleau 93) es una red neuronal que aprende a manejar un vehiculo viendo como maneja un humano. Maneja dos vehiculos equipados especialmente. Se utiliza una camara que alimenta una rejilla de 30×32 entradas a la red. La salida (30 unidades) controla la dirección del volante.
- La red tiene 5 capas ocultas totalmente conectadas. Después de que gente maneja el vehiculo y se entrena al sistema (con retro-propagación, por cerca de 10 min.) el sistema está listo para manejar. Puede manejar hasta a 70 mph por distancias de hasta 90 millas. Extensiones: MANIAC.