

PERCEPTRON SIMPLE

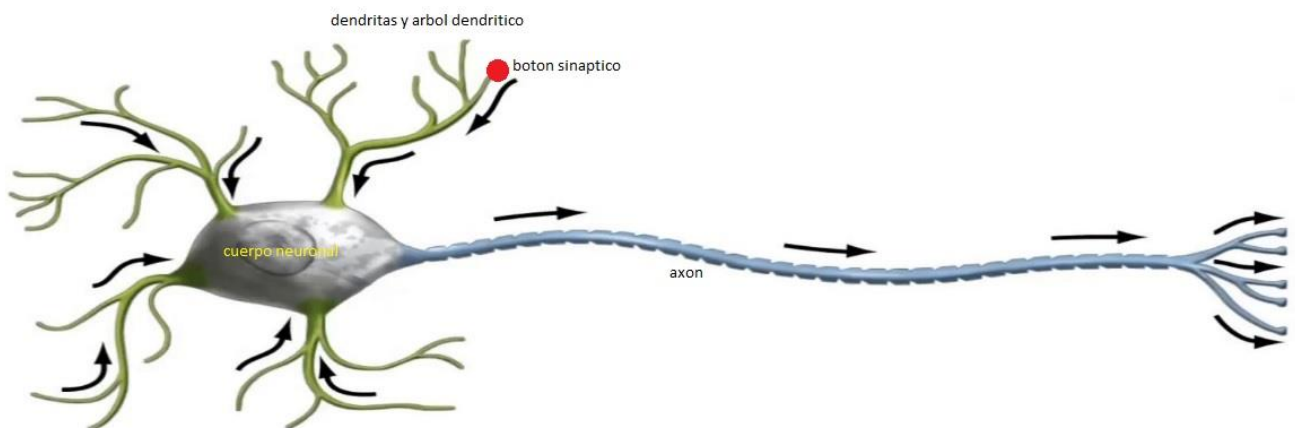
- Video 001

Características de la neurona para poder modelarla en una computadora:

- En el cerebro hay 10^{11} neurona, aunque actualmente hay modelos de esta cantidad no lo vamos a realizar en la catedra.
- No linealidad, un pequeño cambio en la entrada puede generar un gran cambio en la salida o si dos entradas tiene dos salidas la suma de las entradas no es la suma de las salidas.
- Paralelismo, hay muchas neuronas o unidades de proceso que convergen o concurren para resolver problemas complejos pero cada unidad es muy sencilla o elemental.
- Aprendizaje, pueden aprender a partir de los datos le damos ejemplos y ellas aprenden a resolver situaciones parecidas.
- Generalización, es la capacidad de resolver problemas nunca antes visto o sea no es exactamente el mismo problema ejemplo pero puede resolver una situación parecida.
- Adaptabilidad, queremos que sean capaces de adaptarse a nuevas situaciones o sea situaciones que no se vieron en el proceso de aprendizaje.
- Robustez, la capacidad de funcionar bien aun cuando los datos tienen ruido o tiene faltantes.

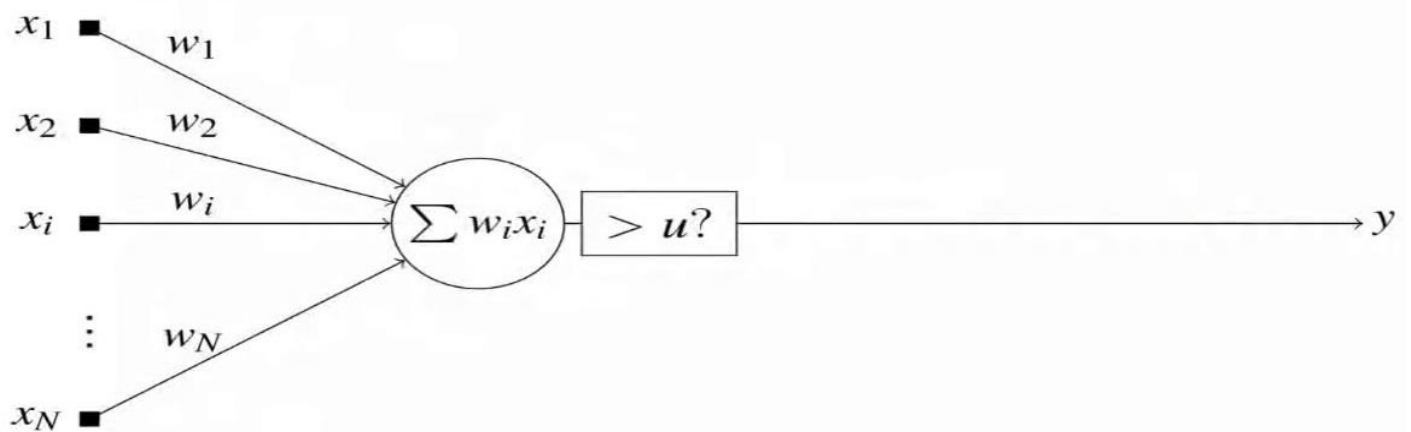
Partes y fisiología de la neurona biológica:

- Sinapsis y botón sináptico, en el botón es donde se produce la sinapsis que es la transición de los neurotransmisores, la finalidad de la misma es despolarizar la membrana de la neurona, cuando recibe una cierta cantidad de estímulos y emite un pulso por el axón. Estos estímulos no son todos iguales, algunos contribuyen más que otros o incluso que inhiban o resten a aumentar el estímulo.
- Comportamiento todo o nada, solamente se envía el pulso cuando se logra la suficiente excitación de la neurona.
- Propagación del impulso
- Refuerzo de la sinapsis o aprendizaje, cuando se va aprendiendo se refuerzan los impulsos y los que eran débiles pasan a ser más fuertes y se va modificando la sinapsis a veces hasta se desconectan.
- Dendrita o árbol dendrítico, es por donde ingresan los impulsos, recibe muchas conexiones
- Axón, es por donde se comunica la respuesta
- Cuerpo neuronal y núcleo, que es donde se procesa la respuesta



- **Video 002**

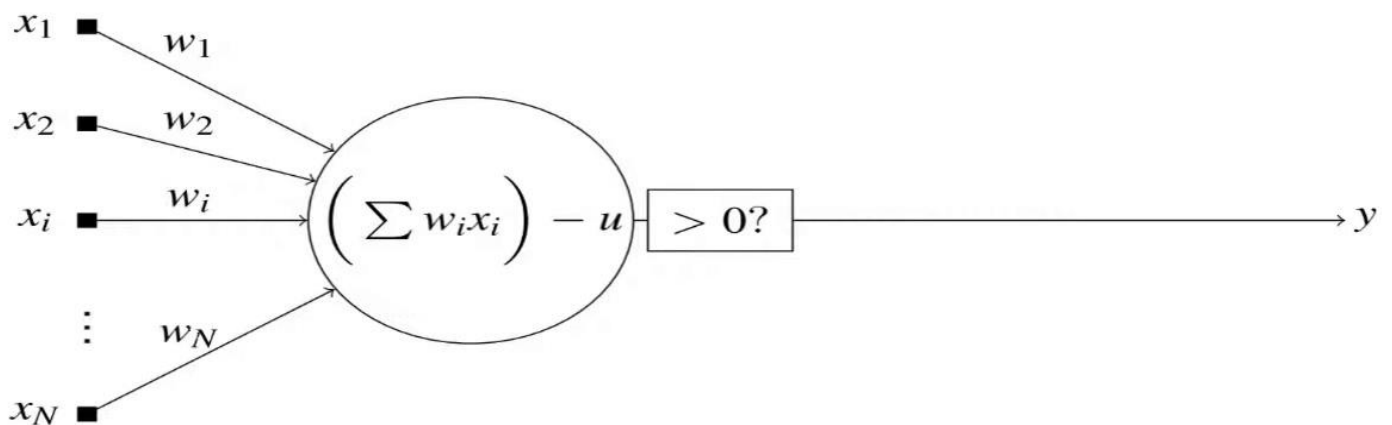
Modelo simplificado de la neurona o perceptron simple:



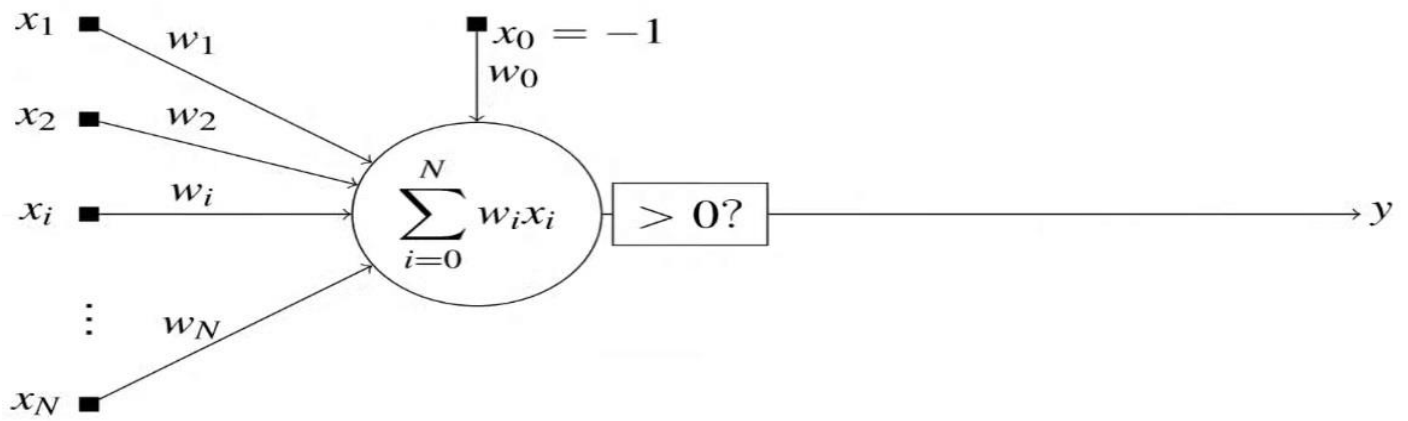
Se simplifica enormemente el funcionamiento de la neurona ya que consideramos n entradas $x_i = x_1 \dots x_n$ y cada entrada va a tener un peso sináptico $w_i = w_1 \dots w_n$ que es un número real sin ninguna limitante. Entonces a cada entrada x_i se la pondera con el peso w_i , se suman estas entradas ponderadas y si es mayor al umbral definido u se envía una respuesta o salida y .

Se realizan mayores simplificaciones asumiendo que las entradas x_i , llegan simultáneamente y que se evalúa de manera instantánea en el cuerpo neuronal si se supera el umbral u .

La siguiente simplificación es con la evaluación del umbral buscando una ecuación más sencilla, introduciendo el umbral como resta en el núcleo de la neurona, dando por resultado:



Se realiza una última simplificación en la cual al umbral u la consideramos una entrada x_0 que siempre es -1 , multiplicado por un peso w_0 dando por resultado:



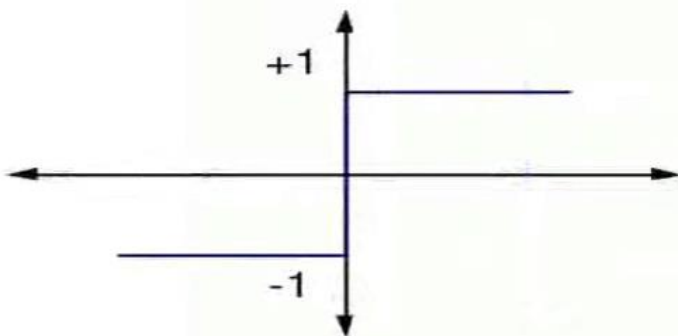
Que se llega o lo que es el perceptron simple, que es un producto interno entre las entradas y peso sináptico denominado activación lineal, y un umbral:

$$y = \phi(v u) = \phi(\sum_{i=1}^N w_i x_i u)$$

Que considerando $x_0 = -1$, $w_0 = u$, obtenemos la función de activación $\phi(z)$

$$\phi(z) = y - \phi\left(\sum_{i=0}^N w_i x_i\right) - \phi(< w, x >)$$

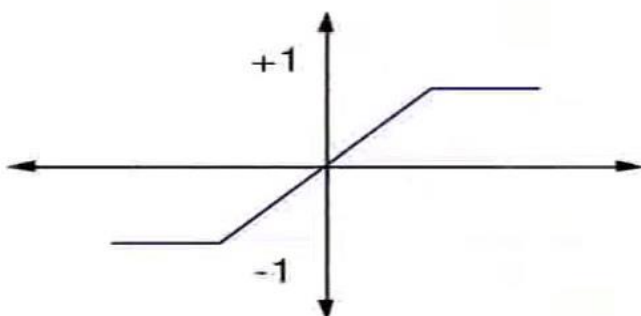
- Funciones de activación $\phi(z)$



$$\text{sgn}(z) = \begin{cases} -1 & \text{si } z < 0 \\ +1 & \text{si } z \geq 0 \end{cases}$$

Existen varias funciones de activación, en la siguiente se realiza una linealización en la sección de transición, salvando así la discontinuidad en el 0:

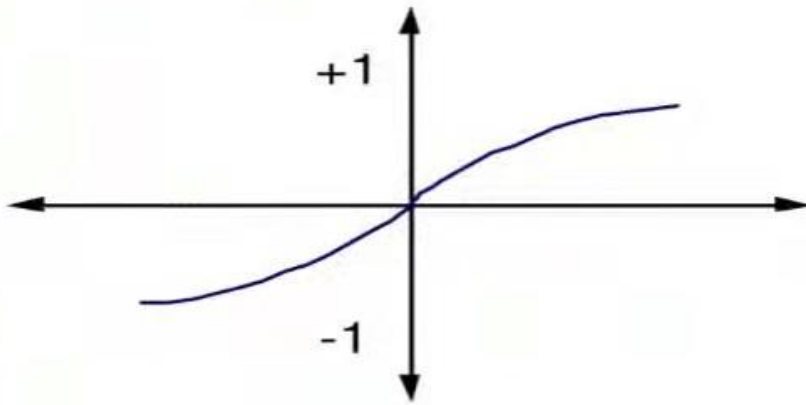
- Funciones de activación $\phi(z)$



$$\text{sln}(z) = \begin{cases} -1 & \text{si } z < -a \\ \alpha z & \text{si } -a < z < a \\ +1 & \text{si } z \geq a \end{cases}$$

También se tiene la función sigmoidea, no tiene ninguna discontinuidad y es no lineal, su principal ventaja es poder derivarla, entre mayor el alfa más parecida a la primera es.

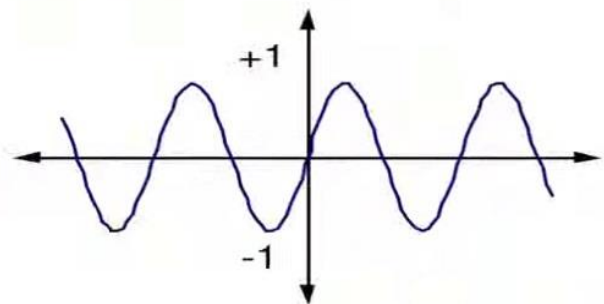
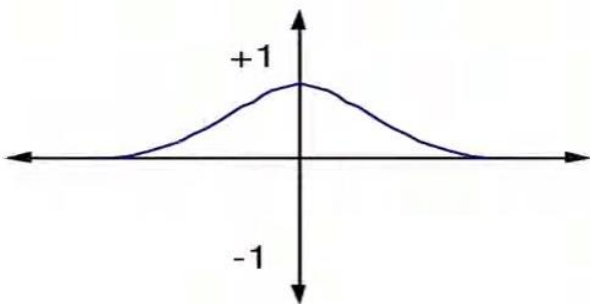
- Funciones de activación $\phi(z)$



$$\text{sig}(z) = \frac{1 - e^{-az}}{1 + e^{-az}}$$

Otras alternativas son la gaussiana y la sinusoidal.

- Funciones de activación $\phi(z)$

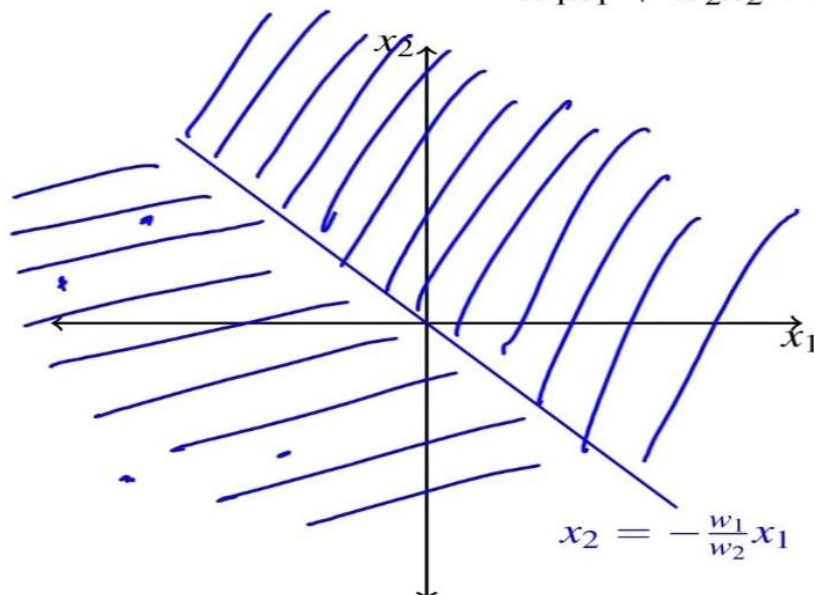


- Video 003

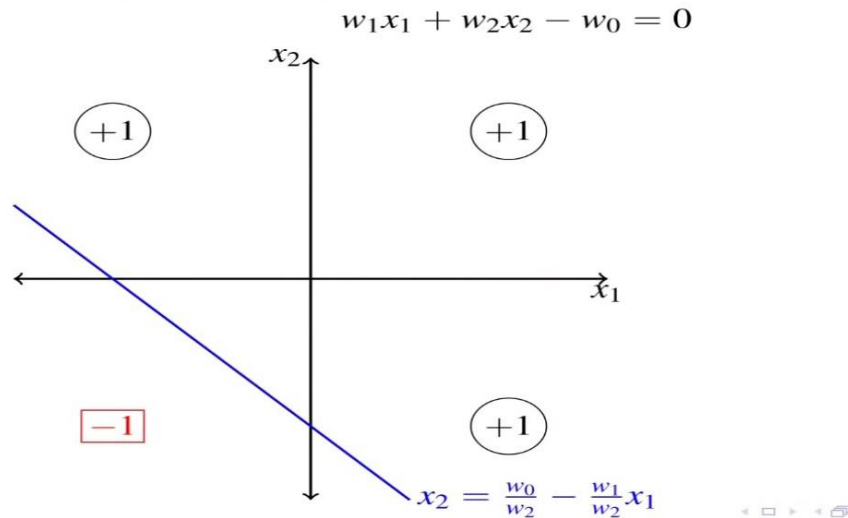
Brinda un ejemplo para 2 entradas con sus respectivos pesos, en donde obtiene una recta de pendiente w_1/w_2 y para el caso del or necesitas la condición u como ordenada al origen.

ejemplo con 2 entradas

$$w_1x_1 + w_2x_2 = 0$$



Perceptrón simple: necesidad del bias



- **Video 004**

Algoritmos de aprendizaje para el perceptron simple, alimentándolo con ejemplo y que solo valla encontrando los valores de los pesos sinápticos, que es lo que termina dando el comportamiento de esa neurona.

El primer algoritmo de entrenamiento es el de corrección por error:

1. Inicialización al azar, esto se realiza en todos los entrenamientos de cualquier algoritmo. Este valor debe ser pequeño ejemplo entre -0.5 y 0.5 la azar.
2. Se muestran muchos ejemplos con las salidas esperadas, estos pueden tener alguna variación y no ser perfectos. Y en cada caso:
 - Si la salida de la red es correcta, no se hacen cambios, principio de mínima perturbación.
 - Si la salida de la red es incorrecta, penalización, se actualizan las w_i en el sentido opuesto con el que contribuyen a la salida incorrecta.
3. Volver a 2 hasta satisfacer algún criterio de finalización.

Algoritmo del perceptrón simple:

1. Inicialización al azar: $\mathbf{w}(1) \in [-0.5 \ 0.5]$
2. Para cada ejemplo de entrenamiento $\mathbf{x}(n) | y_d(n)$:

- Se obtiene la salida:

$$y(n) = \phi(\langle \mathbf{w}(n), \mathbf{x}(n) \rangle)$$

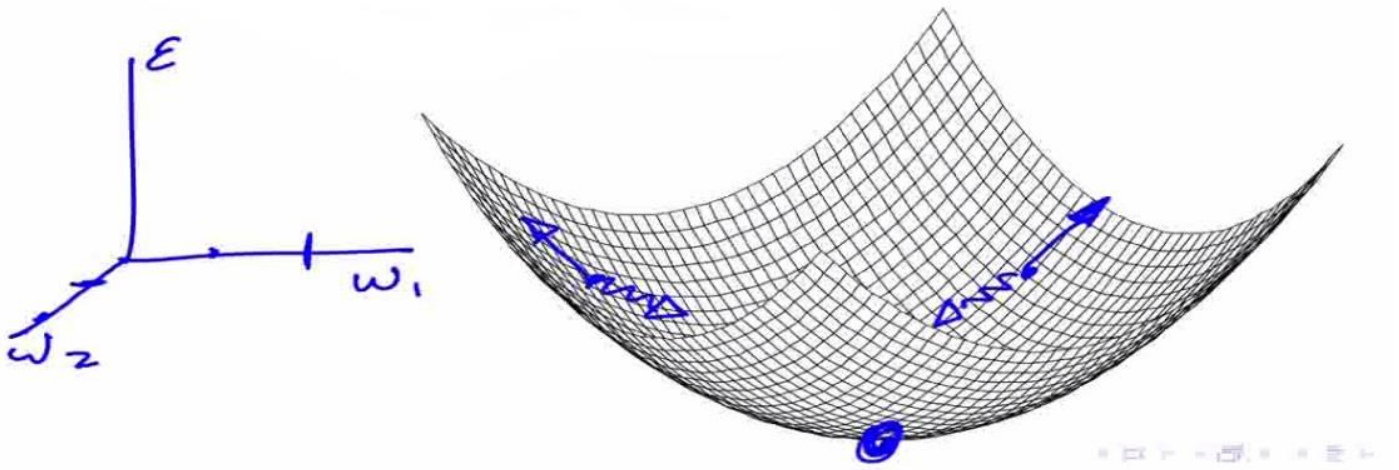
- Se adaptan los pesos:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\eta}{2} [y_d(n) - y(n)] \mathbf{x}(n)$$

3. Volver a 2 hasta satisfacer algún criterio de finalización.

- **Video 005**

Se busca una manera de poder hallar rápidamente los pesos sinápticos, el cual se realiza a través del método del gradiente. Se mueve los pesos en la dirección que se reduce el error que es la opuesta a su gradiente con respecto a los pesos.



Entonces la ecuación para la actualización de los pesos es:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \nabla_{\mathbf{w}} \xi(\mathbf{w}(n))$$

$\mathbf{w}(n+1)$ son los pesos de la interacción siguiente, $\mathbf{w}(n)$ es la iteración actual, es negativo porque hay que ir en la dirección opuesta, μ es el factor de aprendizaje, y después el gradiente del error.

Se puede aplicar para el caso sencillo del perceptron simple como el multicapa.

Para aplicar este método primero debemos obtener el error instantáneo, para un caso lineal ya que su función de activación no es la signo sino que la lineal.

$$e^2(n) = [d(n) - y(n)]^2$$

E es el error, $d(n)$ es la salida deseada, $y(n)$ es la salida actual. Reemplazando por la salida neuronal obtenemos:

$$e^2(n) = [d(n) - y(n)]^2 = [d(n) - \langle \mathbf{w}(n), \mathbf{x}(n) \rangle]^2$$

Derivando con respecto a los pesos obtenemos:

$$\nabla_{\mathbf{w}} e^2(n) = 2 [d(n) - \langle \mathbf{w}(n), \mathbf{x}(n) \rangle] (-\mathbf{x}(n))$$

$$\nabla_{\mathbf{w}} e^2(n) = 2e(n)(-\mathbf{x}(n))$$

reemplazando en: $\mathbf{w}(n+1) = \mathbf{w}(n) - \mu \nabla_{\mathbf{w}} \xi(\mathbf{w}(n))$

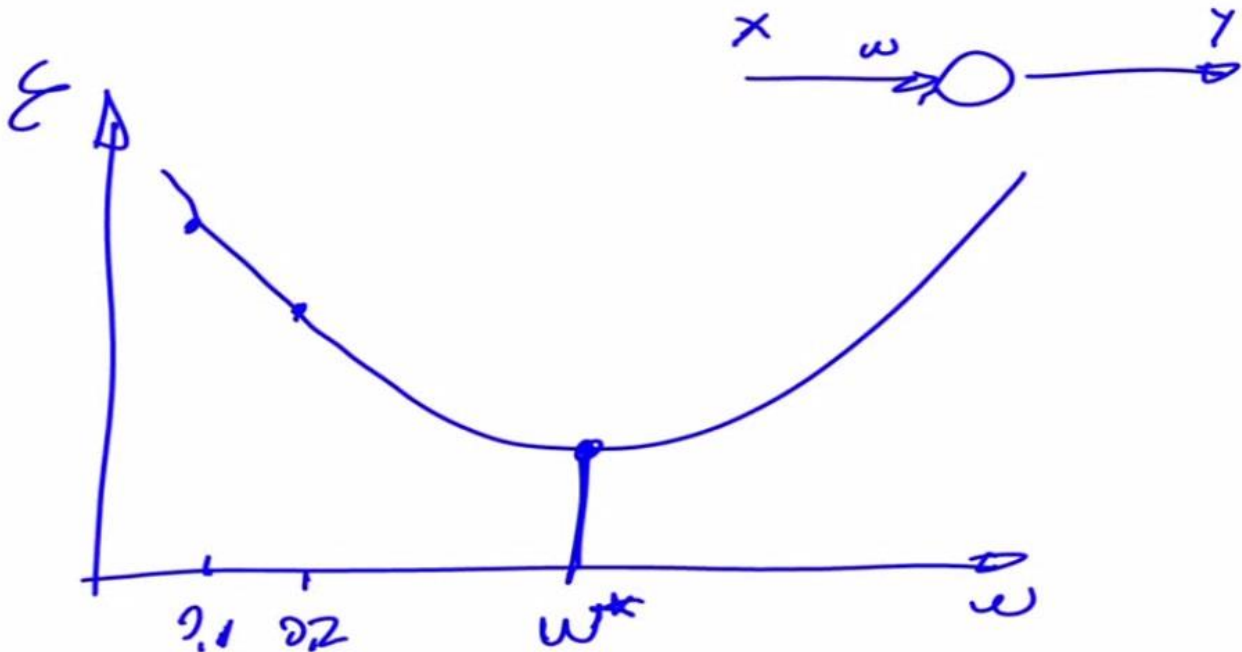
$$\boxed{\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{x}(n)}$$

MEDICIONES DE DESEMPEÑO Y CAPACIDAD DE GENERALIZACION

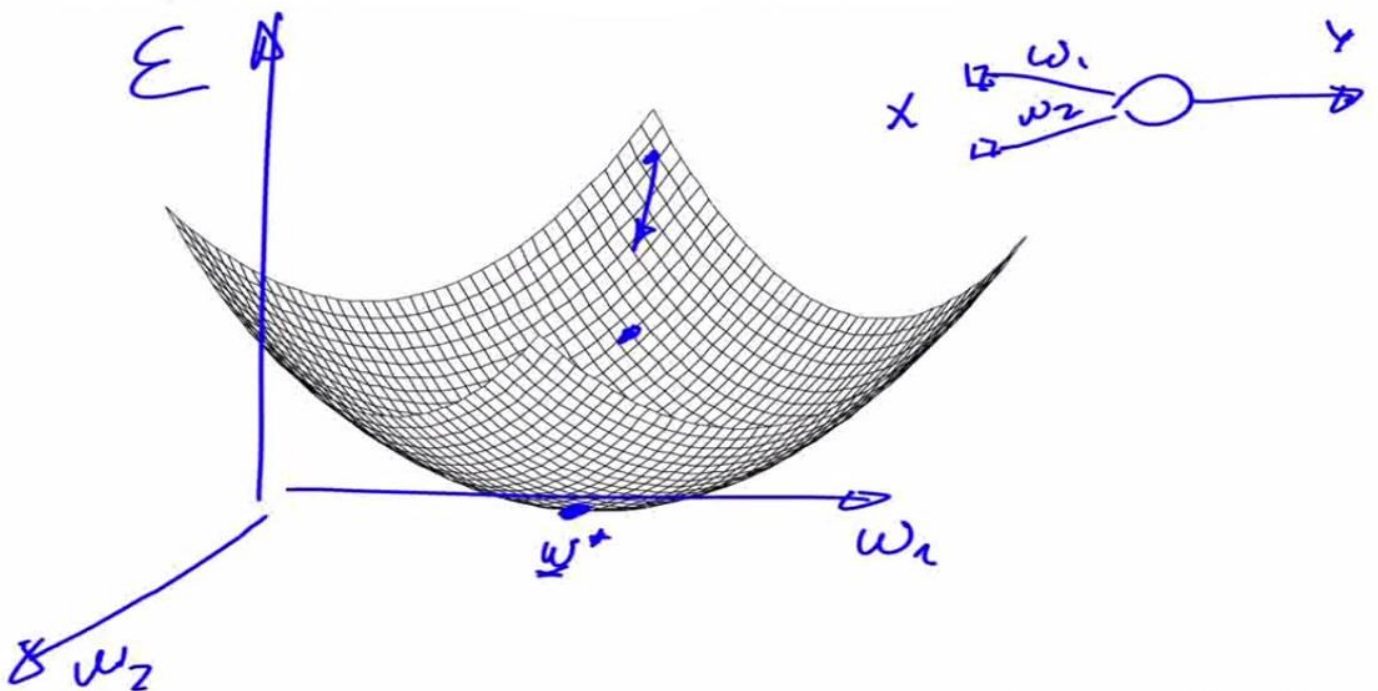
- Video a01

Se analiza la capacidad de generalización y conceptos relacionados de la capacidad que tienen los sistemas inteligentes para resolver situaciones que no se vieron durante el ajuste o entrenamiento.

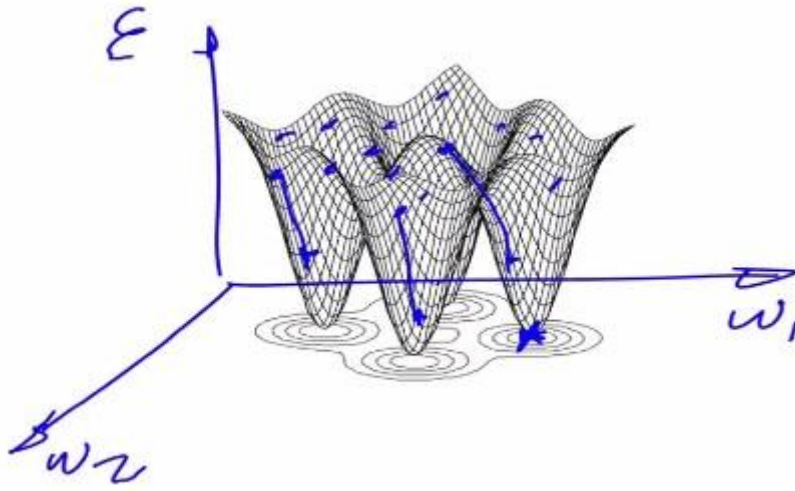
Lo primero que entra en consideración es la superficie de error suponiendo que hay un único parámetro que ajustar sería una curva.



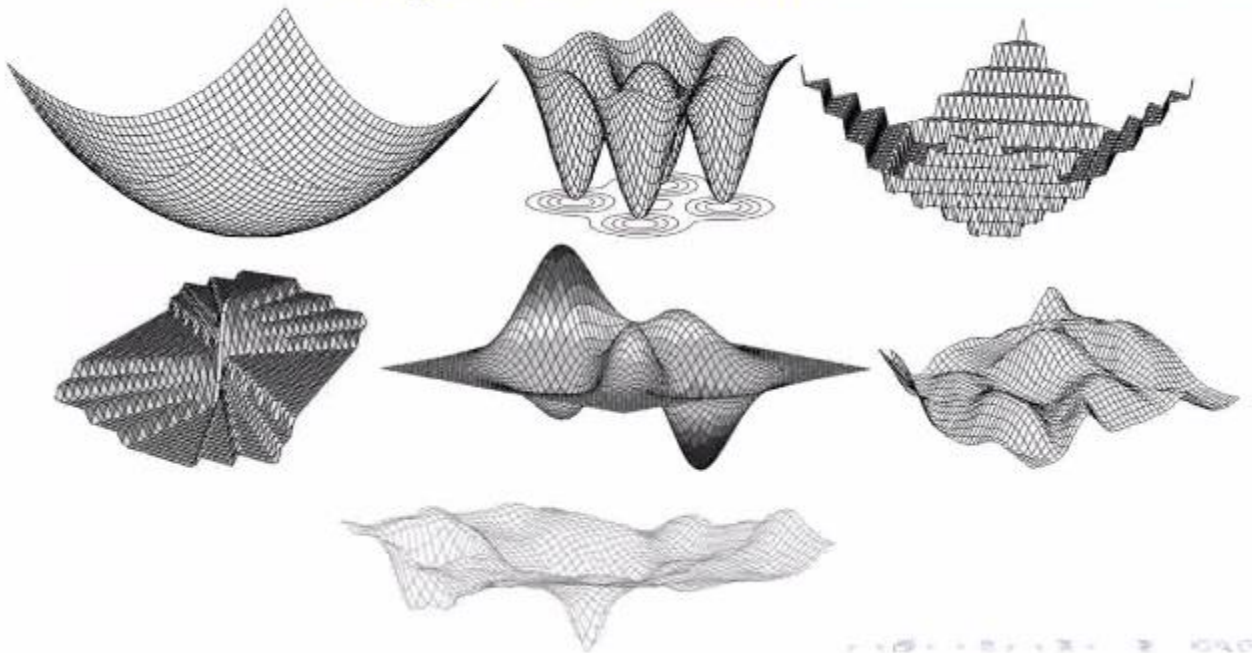
Pero qué pasa si considero dos dimensiones o pesos.



Pero en realidad hay muchísimas variables, los pesos, las capas, la estructura, la conexiones, así que no resulta tan trivial en caos reales. Además puede suceder que no hay únicamente un mínimo, sino varios y quisiéramos el menor de todos los mínimos



Superficies de error



Otro problema para el gradiente es si tengo mesetas o si tenemos muchos mínimos locales. Además que estas superficies son en 2D que no pasa en entornos reales.

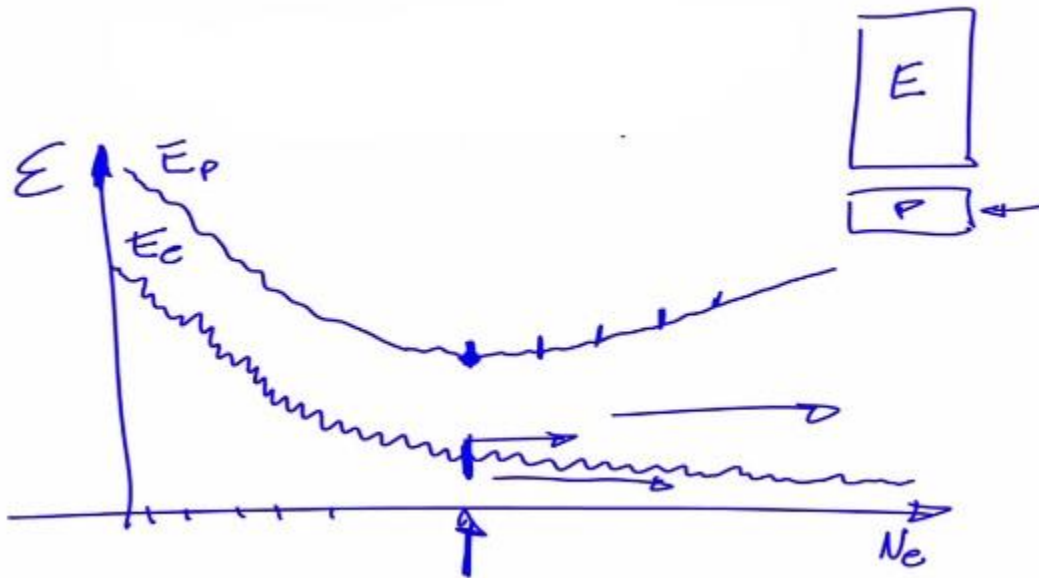
Estas superficies surge durante el entrenamiento y con los datos de entrenamiento, por lo tanto que va a pasar con los casos que el sistema no vio durante el entrenamiento, por lo tanto como aseguramos que va a funcionar bien, que es el concepto de capacidad de generalización que es que tan bien funciona el sistema en situaciones no vistas o previstas, que es la clave del entrenamiento.

Por lo tanto del conjunto de datos, tenemos un conjunto para entrenar y otro para probar para ver que tan bien funcione el entrenamiento ya que en la prueba le daremos casos que nunca vio.

- **Video a02**

Error durante el entrenamiento versus error en la prueba, entonces vimos que en el entrenamiento un bajo porcentaje de error no me asegura totalmente que se comportara adecuadamente durante la prueba con datos que nunca vio. Esto se conoce como sobre entrenamiento, hay que evitarlo ya que el sistema funcionara muy bien en el conjunto de entrenamiento pero no con los nuevos casos nunca vistos.

Entre más épocas se pasan el error va a tender a disminuir, pero a partir de cierto punto en las pruebas el error se va a disparar y aumentar, ya que el algoritmo aprende detalles muy finos de los datos de entrenamiento y que no son detalles generales que nos interesan más o pueden aprender los ruidos ingresados con los datos.



Otra variable que también contribuye al sobreentrenamiento es aumentar demasiado las neuronas para resolver el problema, ya que cada vez puede aprender más y aprende al detalle y no tanto las generalidades. Por ende también hay un punto donde se maximiza con cierta cantidad de neuronas y después funcionara cada vez peor.

- **Video a03**

Se muestran ejemplos de sobre entrenamiento y se da una opción de solución de cortar cuanto el error en las pruebas aumenta con respecto a la iteración anterior. Pero puede acarrear que este corte se pueda deber a que se ajusta mucho a los datos de prueba. El otro problema es que los datos de prueba no los tenemos o no hay que usarlos, entonces del conjunto de entrenamiento tomamos algunos para realizar las pruebas, que se los denomina de monitoreo, que el problema que también acarrea que a veces hay pocos datos de entrenamiento y encima le sacas para monitoreo.

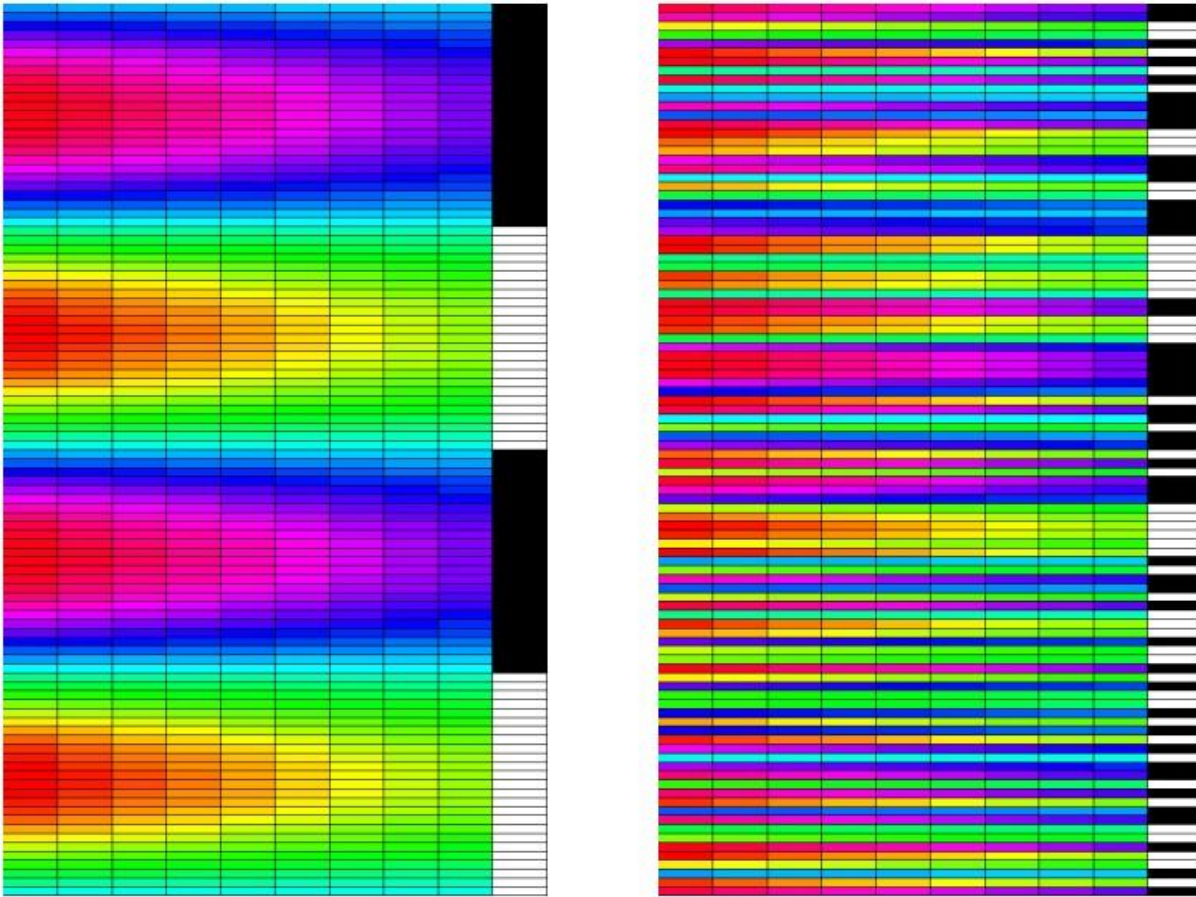
- **Video a04**

Como medimos la capacidad de generalización, que tan bien va a funcionar en la realidad o aunque sea poder tener una estimación. Que es la verdadera tasa de error o acierto que me interesa más allá del error en el entrenamiento.

A un conjunto de datos de entrada le asignamos colores quedando:

0.2485	0.8507	0.5274	0.4113	0.1740	0.7892	0.1007	0.6332	0.8224	1
0.4500	0.3887	0.8276					191	0.0017	0
0.4109	0.6664	0.9390					787	0.9631	1
0.2603	0.4436	0.5336					424	0.8403	1
0.8704	0.6089	0.5303					363	0.0992	0
0.1850	0.7844	0.4559					416	0.7883	1
0.0197	0.9680	0.4487					102	0.8131	1
0.9533	0.3733	0.1680					145	0.3894	0
0.6805	0.6727	0.3775					402	0.6952	1
0.4866	0.8077	0.9378					829	0.0893	1
0.9650	0.6975	0.5968					711	0.0639	1
0.3934	0.8061	0.5593					631	0.4700	1
0.0796	0.2670	0.9345					242	0.2121	1
0.3514	0.3185	0.5308					669	0.6160	1
0.1636	0.1600	0.7816					163	0.3861	1
0.9832	0.8161	0.6401					194	0.5153	0
0.8806	0.8728	0.0027					329	0.3616	0
0.4941	0.8259	0.5029					451	0.2095	0
0.4010	0.7552	0.6880					016	0.8960	1
0.4513	0.5171	0.0475					336	0.5675	0
0.7209	0.5527	0.2922					385	0.5174	0
0.2478	0.5581	0.6848					135	0.0727	0
0.6228	0.9937	0.8572					280	0.0589	1
0.1424	0.9865	0.0785					779	0.1814	0
0.2012	0.5102	0.4075					579	0.6625	1
0.0812	0.7557	0.9157					184	0.0799	0
0.9535	0.9366	0.3024					616	0.6250	1
.
.
.

Ordenándolo según cada salida:



Una desventaja es que si selecciono una parte para la prueba estaría tomado casos donde la salida es de una categoría. Por lo tanto es más conveniente ordenarlos al azar.

Entonces una técnica es la validación cruzada, en la cual se realizan particiones de los datos para ir teniendo distintos set de pruebas pero con los mismos datos, se entrena con los elementos restantes de la partición y se los prueba con la partición elegida para prueba y así sucesivamente, pero siempre iniciando los pesos en la siguiente prueba de manera aleatoria. Entonces guardo cada uno de estos errores y luego los promedio.

Una de las falencias de este método es que la diferencia entre los errores de cada partición puede ser grande, acarreado que a veces funciona muy bien y otras muy mal pero en el promedio se compensan, por eso calculando la varianza podemos ver si los promedios no son muy dispares.

- **Video a05**

Sigue hablando del método de validación cruzada, el visto anterior se llama leave-k-out. Otra manera similar es k-fold, es hacer k veces particiones pero no necesariamente dejar a k fuera. El caso extremo de leave-k-out, es leave-1-out, dentro de la partición dejo uno afuera nomás y entreno con el resto de la partición, prosigo con el siguiente y entreno con el resto de la partición y así sucesivamente, la desventaja es que tengo que entrenar al algoritmo muchas veces y tiene un costo elevado aunque es el que menor error nos da.

Otra técnica cuando tengo pocos datos, es realizar particiones no excluyentes dando un grado de solapamiento, se pierde la independencia estadística, pero se gana en tener muchas más particiones para promediar.

La última técnica que la comentamos, es tomar particiones y elegir patrones con repetición que pueda elegir un patrón una vez y volver a elegirlo otra vez, permite hacer muchas más particiones, incluso esa elección podría ser al azar.

- Video a06

Se habla de medir el error pero no se especifica como medimos ese error. Entonces las medidas de desempeño de la clasificación presentando la matriz de confusión. En este caso existirán 2 clases los positivos y negativos.

Matriz de confusión

		Predicciones		
		\oplus	\ominus	
Clases correctas	\oplus	t_{\oplus}	f_{\ominus}	$N_{\oplus} = t_{\oplus} + f_{\ominus}$
	\ominus	f_{\oplus}	t_{\ominus}	$N_{\ominus} = f_{\oplus} + t_{\ominus}$
				$N = N_{\oplus} + N_{\ominus}$

En el centro para la clase positiva, van los verdaderos positivos (el patrón era positivo y el clasificador también dijo positivo) y los falsos para los negativos (son aquellos casos que el patrón era positivo pero el clasificador lo dio como negativo), que sumando estos dos nos da los casos total positivos.

Se procede igual para la clase de los negativos, con los falsos positivos y los verdaderos negativos, que ambos dan el total de negativos.

Entonces el centro da la matriz de confusión, que si nos fijamos la diagonal si da buenos números o sea la diagonal es mayor que los de fuera, es un clasificador bastante bueno.

La matriz de confusión es la base para definir todos los índices de medida de desempeño.

Sensibilidad y especificidad:

Los nombres alternativos son, tasa de verdaderos positivos, tasa de éxito, recall o potencia. Es la división de los verdadero positivos por la suma de verdaderos positivos y los falsos negativos.

$$s^{+} = \frac{t_{\oplus}}{t_{\oplus} + f_{\ominus}} = \frac{t_{\oplus}}{N_{\oplus}} \rightarrow \left\{ \begin{array}{l} \text{**sensitivity**} \\ \text{true positive rate} \\ \text{hit rate} \\ \text{recall} \\ \text{power} \end{array} \right.$$

El problema que acarrea, es que podría estar haciéndolo muy bien con los positivos pero no con los negativos, por lo tanto hay que tener cuidado ya que analiza un único aspecto del sistema.

La especificidad o tasa de verdaderos negativos es similar a la anterior porque cuenta los mismo pero para los negativos.

$$s^{-} = \frac{t_{\ominus}}{t_{\ominus} + f_{\oplus}} = \frac{t_{\ominus}}{N_{\ominus}} \rightarrow \left\{ \begin{array}{l} \text{**specificity**} \\ \text{true negative rate} \end{array} \right.$$

Por lo tanto también acarrea el mismo problema que analiza nomas un aspecto.

Precisión:

También llamada valor predictivo positivo, en el numerador están los positivos, pero los divide por los verdaderos positivos más los falsos positivos, por ende sonde suban muchos los falsos positivos la precisión va bajar

$$p = \frac{t_{\oplus}}{t_{\oplus} + f_{\oplus}} \rightarrow \left\{ \begin{array}{l} \text{precision} \\ \text{positive predictive value} \end{array} \right.$$

Una medida relacionada, es el valor predictivo de negativos, que es igual a la anterior pero enfocada a los negativos.

$$n = \frac{t_{\ominus}}{t_{\ominus} + f_{\ominus}} \rightarrow \left\{ \begin{array}{l} \text{negative predictive value} \end{array} \right.$$

Exactitud:

Es la más intuitiva y la más utilizada, se suman los verdaderos positivos y los verdaderos negativos y luego lo dividimos por la cantidad de casos.

$$a = \frac{t_{\oplus} + t_{\ominus}}{N} \rightarrow \left\{ \begin{array}{l} \text{accuracy} \end{array} \right.$$

F1 score:

Está focalizada en los positivos, ya que es la que más interesa porque es cuando le estoy acertando, se combina con una media armónica la sensibilidad con la precisión.

$$F_1 = \frac{2t_{\oplus}}{2t_{\oplus} + f_{\oplus} + f_{\ominus}} = 2 \frac{s^+ p}{s^+ + p} \rightarrow \left\{ \begin{array}{l} F_1 \text{ score} \\ \text{F-score/measure} \\ \text{harmonic mean} \end{array} \right.$$

Media geométrica:

También relaciona la sensibilidad y precisión, que también nos da una idea de que también está andando el clasificador.

$$G = \sqrt{s^+ p} \rightarrow \left\{ \begin{array}{l} \text{G-measure} \\ \text{geometric mean} \end{array} \right.$$

- Video a07

Primeras ideas sobre errores relativos. Entonces podemos definir el error de exactitud $e=1-a$. Y el error relativo de referencia, que toma como referencia el clasificador menos o punto de comparación el error de exactitud, todo dividido el error de referencia. Analiza entre un error al siguiente entrenamiento o distintos clasificadores, cuanto se reducen porcentualmente los errores.

$$\delta_e = \frac{e_r - e}{e_r}$$

- **Video a08**

Medidas de predicción en series, es análogo a cuando realizábamos en estadísticas las regresiones lineales y cuanto mi modelo se ajusta la línea real, tomando el r y R^2. Que era la suma de las distancias entre mi valor y el valor real.

Las medidas utilizadas son el error medio absoluto:

$$\tilde{\epsilon}_A = \frac{1}{N} \sum_{i=1}^N |e_i|$$

Otra medida es el error cuadrático medio, se usa mucho más:

$$\tilde{\epsilon}_S = \frac{1}{N} \sum_{i=1}^N e_i^2$$

Por último la raíz cuadrada el error cuadrático medio:

$$\tilde{\epsilon}_R = \sqrt{\frac{1}{N} \sum_{i=1}^N e_i^2}$$

Introducción al reconocimiento estadístico de patrones

- **Video i01**

Se inicia con la base estadística del reconocimiento de patrones, que es tratar de aproximarnos como un objetivo de máxima imitar el proceso de razonamiento humano, cuando reconocemos objetos. Plantea unas imágenes donde hay que reconocer llaves y que se va complejizando, para hacer una analogía de como funcionamos.

Entonces podemos observar cual es el proceso que se intenta reproducir en un sistema informático que es enseñarle a la computadora que emule todas las tareas que se realizaron para la tarea de ejemplo, en particular detectar los objetos, percibir el entorno, identificarlos, darles una etiqueta para reconocerlos, contarlos, etc.

Entonces el aprendizaje maquina tiene como objetivo programar máquinas para:

- Aprender a realizar una tarea, tienen un objetivo específico a resolver.
- Mejorando su desempeño, siempre midiendo el desempeño
- Basado en la experiencia, cuantos más ejemplos les mostremos el sistema debe ser capaz de ir aprendiendo.

Tienen restricciones adicionales:

- Mínima intervención humana posible, como máxima sin intervención humana, que funcionen de manera autónoma.
- Con reducido conjunto de ejemplos, conseguir bases de datos para aprender una tarea puede ser muy costoso y complejo.

- **Video i02**

Se verán conceptos básicos de aprendizaje maquina para luego aproximarnos a los sistemas particulares de reconocimiento.

Hay dos grandes ramas de la inteligencia artificial:

- IA clásica, fue la primer aproximación:
 - Modelado del proceso de razonamiento humano mediante técnicas de la lógica
 - Aprendizaje deductivo
 - Es lo más clásico se toma decisiones en base a los resultados y un análisis de experto
- Reconocimiento de formas, es más reciente:
 - Modelado del proceso de percepción humano mediante técnicas de la teoría de la decisión estadística y de la teoría de los lenguajes formales
 - Aprendizaje deductivo es decir a través de ejemplos
 - Es como el ejemplo de las llaves

Algunas consideraciones del aprendizaje maquina desde el punto de vista de la inteligencia artificial:

- Adquisición y representación del conocimiento, conformación y almacenamiento de conjunto de patrones o prototipos, en un sistema de este tipo debemos ser capaces de tomar los patrones de la realidad y representarlos de alguna manera óptima para el sistema
- Aprendizaje, algoritmos de aprendizaje inductivo a partir de un conjunto de entrenamiento, a través de muestras poder inferir el comportamiento maquina
- Clasificación, etiquetado de patrones nuevos utilizando el conjunto de clases disponibles, asignarles identidad a patrones nuevos no vistos en el entrenamiento, es la capacidad de generalización

- Evaluación, mecanismos para evaluar la bondad, confianza o error del sistema, utilizar medidas de desempeño y error

Patrón:

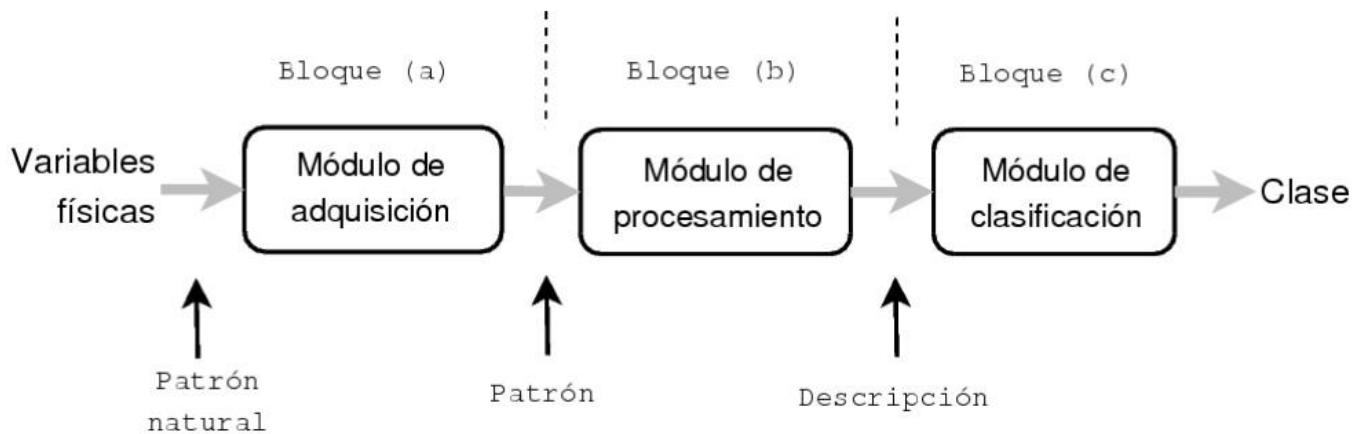
Son los objetos de interés o lo que queremos clasificar, que es identificable del resto, posiblemente difusos o con ruido.

Reconocimiento de patrones:

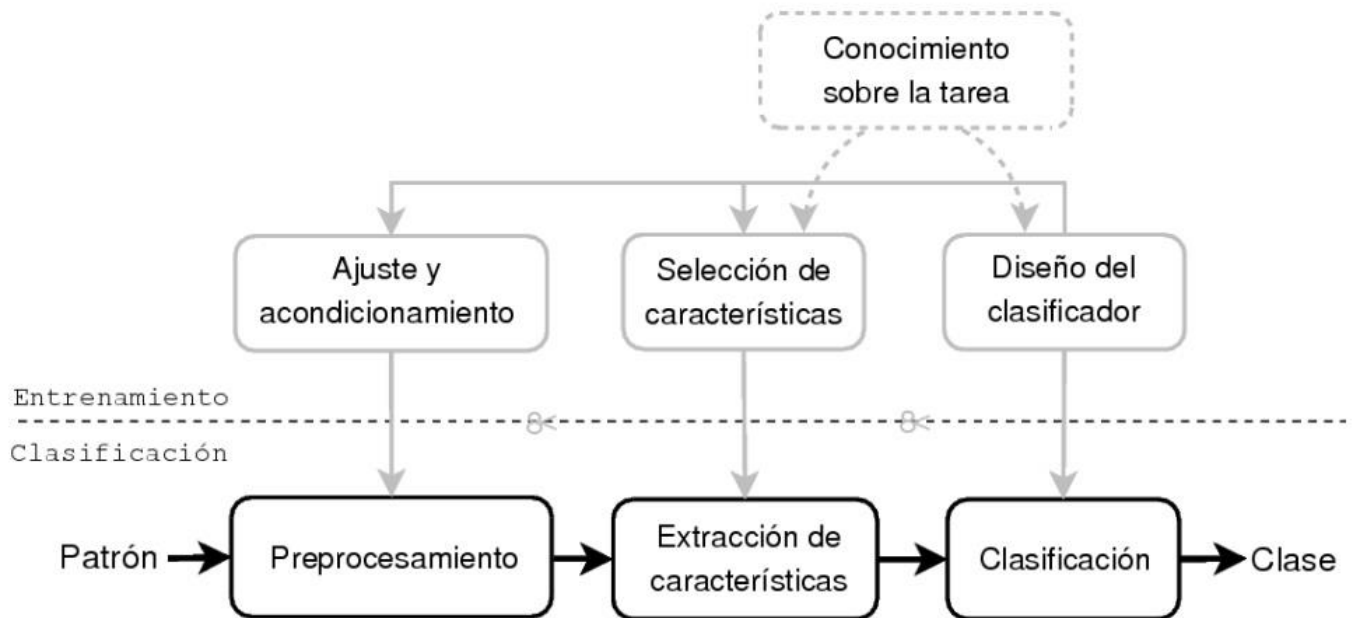
Estudia los procesos de percepción y razonamiento humano, siendo la capacidad de distinguir y aislar los patrones, reunirlos en grupos y asignarles etiquetas o nombres. Siendo el objetivo crear sistemas informáticos que realicen lo mismo.

El módulo de bloques de un sistema de reconocimiento lo podemos describir:

- (a) Adquisición: transducción del mundo real a la representación digital.
- (b) Procesamiento digital: acondicionamiento y representación alternativa.
- (c) Clasificación: decisión sobre la clase.



A veces también se le denomina patrón a la señal natural ya procesada que es la descripción en el diagrama.



- **Video i02**

Hay dos grandes aproximaciones para poder realizar el proceso de clasificación, numérico o sintáctico.

Aproximación geométrica o estadística:

Está basada en la teoría estadística de la decisión, se representan los patrones como vectores numéricos, las clases mediante patrones prototipos dando clasificadores Gaussiano, basados en la distancia, etc.

Aproximación estructural o sintáctica:

Basada en la teoría de lenguajes formales, representan los patrones como cadenas de símbolos, utilizan reglas sintácticas para especificar los patrones validos de una clase, siendo los tipos de clasificadores los autómatas, gramáticas, etc.

PERCEPTRON MULTICAPA

- Video 006

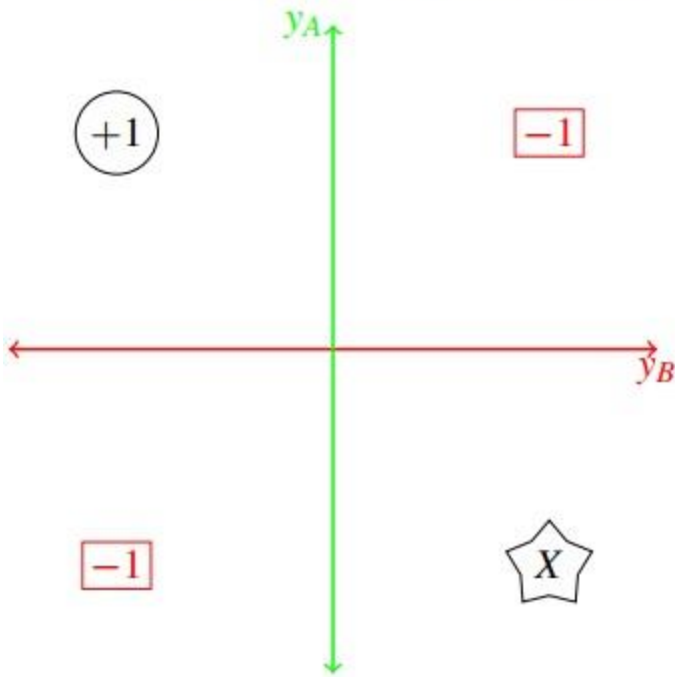
Enfoque inicial de como conectar 3 perceptron simples para poder resolver el xor, se conecta la salida de dos perceptron a un tercero.

Considerando la primer capa únicamente los perceptron a y b



Considerando la segunda capa formada por las salidas tanto de a, b conectadas al perceptron c

Tabla de verdad para el perceptron C



- **Video 007**

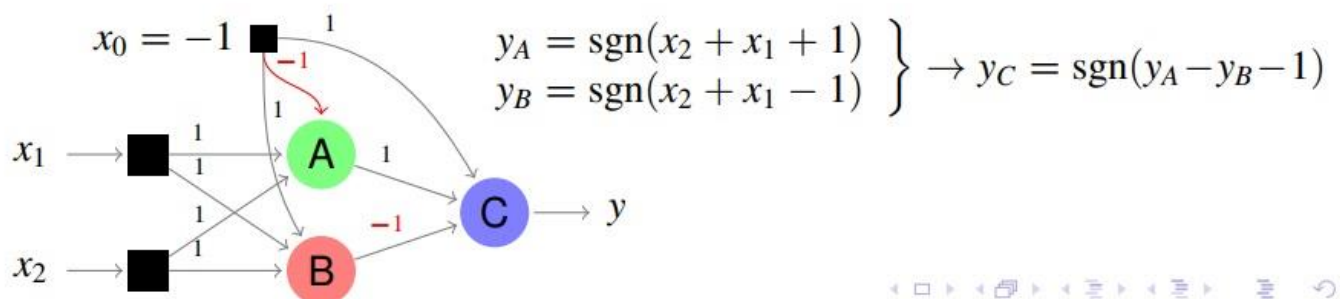
Para el ejemplo anterior de resolución de xor, empieza a buscar los pesos para las entradas del perceptron a, b, c dando por resultado.

$$\left\{ \begin{array}{l} w_{C0} = +1 \\ w_{C1} = -1 \\ w_{C2} = +1 \end{array} \right\}$$

$$\left\{ \begin{array}{l} w_{A0} = -1 \\ w_{A1} = +1 \\ w_{A2} = +1 \end{array} \right\} \left\{ \begin{array}{l} w_{B0} = +1 \\ w_{B1} = +1 \\ w_{B2} = +1 \end{array} \right\}$$

$$\left. \begin{array}{l} y_A = \text{sgn}(x_2 + x_1 + 1) \\ y_B = \text{sgn}(x_2 + x_1 - 1) \end{array} \right\} \rightarrow y_C = \text{sgn}(y_A - y_B - 1)$$

Analizando los pesos y conexiones de las entradas obtenemos:



- **Video 008**

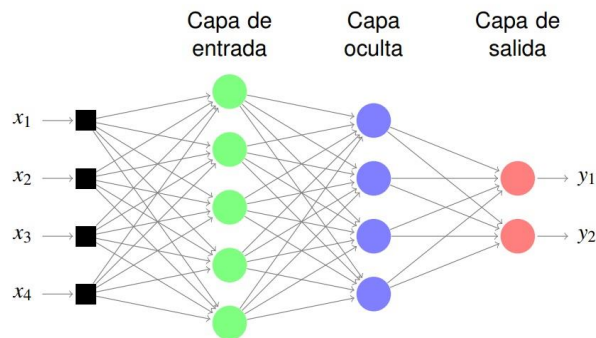
En base a los pesos y funciones asignadas a los perceptrones, se toman las entradas x_1 , x_2 seteandolas en las 4 posibles combinaciones de entradas con 1 y -1, luego analiza cada salida de la red y las compara con las del xor, que las resuelve.

- **Video 009**

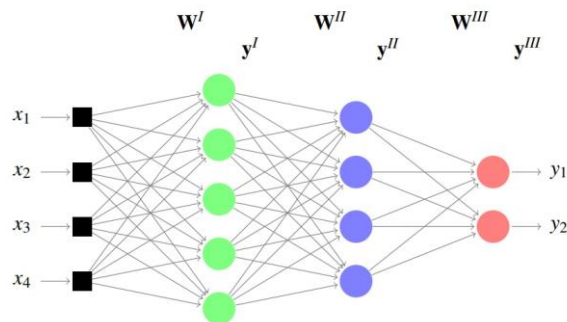
Con el enfoque visto para 3 perceptrones, se va a intentar generalizar como se solucionan problemas. Lo primero que analizamos son las regiones de decisión.

Estructura	Tipos de regiones de decisión	Problema XOR	Separación en clases	Formas regiones más generales
Una capa 	hemiplano limitado por hiperplano			
Dos capas 	Regiones convexas abiertas o cerradas			
Tres capas 	Arbitrarias (Complejidad limitada por N° de Nodos)			

Podemos clasificar las capas en 3, la capa de entrada que está en contacto con las entradas, la capa oculta que no está en contacto no con las entradas o salidas, y por último la capa de salida que es aquella que brinda las salidas finales.



Se empieza a analizar la notación para llamas a las entradas y salidas en las distintas capas.



Para calcular las salidas, se calcula el producto punto entre las entradas y salidas y también considerando el umbral como entrada pero utilizando la función sigmoidea para dar la salida.

- Capa I:

$$v_j^I = \langle \mathbf{w}_j^I, \mathbf{x} \rangle = \sum_{i=0}^N w_{ji}^I x_i \quad (\text{completo } \mathbf{v}^I = \mathbf{W}^I \mathbf{x})$$

$$y_j^I = \phi(v_j^I) = \frac{2}{1 + e^{-bv_j^I}} - 1 \quad (\text{simétrica } \pm 1)$$
- Capa II:

$$v_j^{II} = \langle \mathbf{w}_j^{II}, \mathbf{y}^I \rangle \rightarrow y_j^{II} = \phi(v_j^{II})$$
- Capa III:

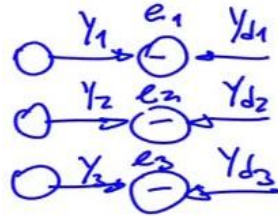
$$v_j^{III} = \langle \mathbf{w}_j^{III}, \mathbf{y}^{II} \rangle \rightarrow y_j^{III} = \phi(v_j^{III}) = y_j$$

• Video 010

Para poder actualizar los pesos se realiza una propagación hacia atrás, para lograr así que la red neuronal aprenda. Primero se define un criterio de error la suma del error cuadrático instantáneo.

$$\xi(n) = \frac{1}{2} \sum_{j=1}^M e_j^2(n)$$

Ya que ahora no tenemos únicamente una salida sino que tenemos varias y cada una tiene un error, que se lo eleva al cuadrado así en la suma son todos positivos, de lo contrario se suman y restan y podría llegar a dar 0 cosa que es incorrecta. El $\frac{1}{2}$ está por conveniencia ya que luego para derivar para hallar el gradiente se simplifica. Se denomina instantáneo ya que es en el instante n , no es un error para todos los patrones o ejemplos, sino que es para la iteración n -ésima.



Ahora entonces queremos hallar el gradiente del error que es la derivada del error con respecto a los pesos, pero es una concatenación de funciones se usa regla de la cadena tantas veces capas tenga.

$$\Delta w_{ji}(n) = -\mu \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

• Video 011

Se va a demostrar cómo aplicar el gradiente en los distintos perceptrones, así ir en la dirección opuesta donde aumenta el error.

Yi es la entrada a la neurona, Yj es la salida.

$$\Delta w_{ji}(n) = -\mu \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} = \Delta w_{ji}(n) = \mu \delta_j(n) y_i(n)$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \frac{\partial \sum_{i=0}^N w_{ji}(n) y_i(n)}{\partial w_{ji}(n)} = y_i(n)$$

$$\text{Gradiente de error local instantáneo: } \delta_j = \frac{\partial \xi(n)}{\partial y_i(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

Derivada de la función de activación

$$\frac{1}{2} (1 + y_j(n)) (1 - y_j(n))$$

• Video 012

Se analiza la retro propagación del error en la capa 3, el más sencillo. Lo más complicado es hallar las derivadas.

$$\delta_j^{III}(n) = -\frac{\partial \left\{ \frac{1}{2} \sum_j e_j^2(n) \right\}}{\partial e_j(n)} \cdot \frac{\partial \left\{ d_j^{III}(n) - y_j^{III}(n) \right\}}{\partial y_j^{III}(n)} \cdot \frac{1}{2} (1 + y_j^{III}(n))(1 - y_j^{III}(n))$$

$$\delta_j^{III}(n) = \frac{1}{2} e_j(n) (1 + y_j^{III}(n))(1 - y_j^{III}(n)) \star$$

$$\Delta w_{ji}^{III}(n) = \eta e_j(n) (1 + y_j^{III}(n))(1 - y_j^{III}(n)) y_i^{II}(n)$$

- **Video 013**

Se analiza cómo es la retro propagación de la capa oculta o capa 2.

$$\delta_j^{III}(n) = -\frac{\partial \left\{ \frac{1}{2} \sum_j e_j^2(n) \right\}}{\partial e_j(n)} \cdot \frac{\partial \left\{ d_j^{III}(n) - y_j^{III}(n) \right\}}{\partial y_j^{III}(n)} \cdot \frac{1}{2} (1 + y_j^{III}(n))(1 - y_j^{III}(n))$$

$$\delta_j^{III}(n) = \frac{1}{2} e_j(n) (1 + y_j^{III}(n))(1 - y_j^{III}(n)) \star$$

$$\Delta w_{ji}^{III}(n) = \eta e_j(n) (1 + y_j^{III}(n))(1 - y_j^{III}(n)) y_i^{II}(n)$$

Que se puede generalizar para cualquier capa oculta dando.

$$\Delta w_{ji}^{II}(n) = \eta \left[\sum_k \delta_k^{III} w_{kj}^{III}(n) \right] (1 + y_j^{II}(n))(1 - y_j^{II}(n)) y_i^I(n)$$

↓

$$\Delta w_{ji}^{(p)}(n) = \eta \left\langle \delta^{(p+1)}, \mathbf{w}_j^{(p+1)} \right\rangle (1 + y_j^{(p)}(n))(1 - y_j^{(p)}(n)) y_i^{(p-1)}(n)$$

- **Video 014**

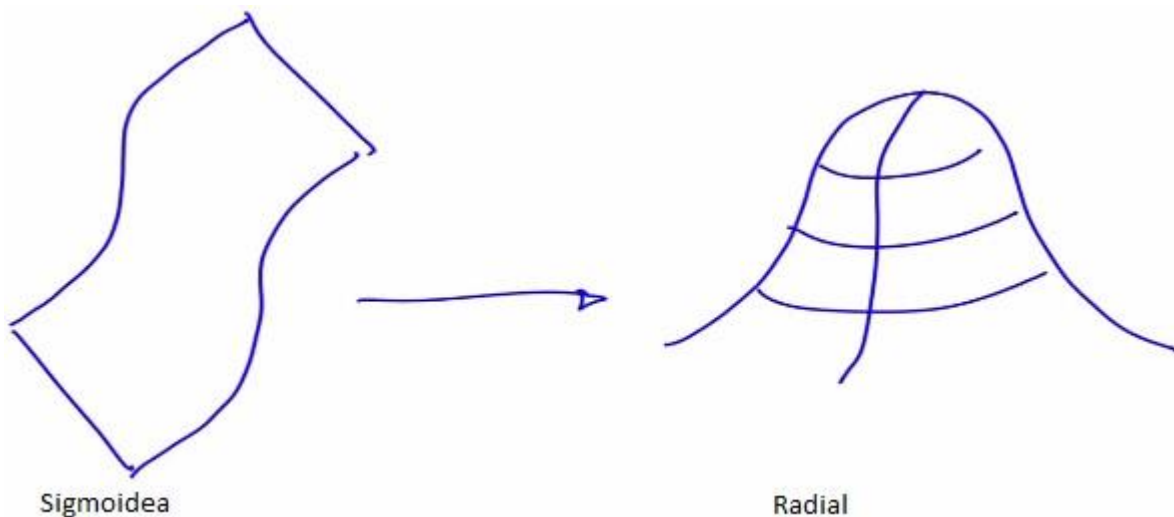
Se ve el algoritmo para perceptrones multicapa.

1. Inicialización aleatoria
2. Propagación hacia adelante
3. Propagación hacia atrás
4. Adaptación de los pesos
5. Iteración: vuelve a 2 hasta convergencia o finalización

REDES EN BASE RADIAL

- **Video 015**

Vuelve a plantear el problema del xor para multicapa con las 3 regiones de decisión. Entonces se puntualiza en que forma tiene la sigmoidea para resolver el problema, entonces se considera si podemos cambiar la forma siendo un círculo en vez de la s de la sigmoidea. Entonces el principal cambio es en la función de activación que se intenta sean un círculo.



Originalmente se usaban para aproximar funciones.

$$\phi : \mathbb{R}^N \rightarrow \mathbb{R}$$

$$d = \phi(\mathbf{x})$$

Aproximación:

$$h(\mathbf{x}) = \sum_j w_j \phi(\|\mathbf{x} - \mu_j\|)$$

frecuentemente se utiliza como función de base radial:

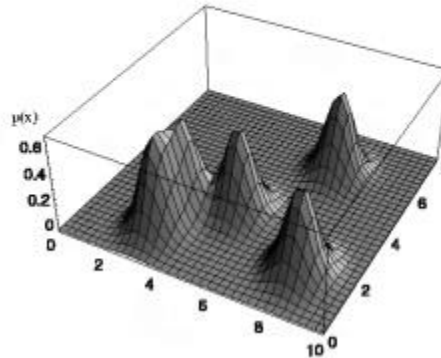
$$\phi(\kappa) = e^{-\frac{\kappa^2}{2\sigma^2}}$$

Como se observa phi tiene \mathbb{R}^n por lo tanto tiene n entradas y 1 salida, la función d toma la distancia desde las entradas x. El para realizar la aproximación de plantea la función h, compuesta por unos pesos w y alguna función de

base radial que mide una distancia entre los patrones de entrada y un vector de medias, y a esa distancia la pasa a alguna función no lineal, que la propuesta es la phi de k, gaussiana.

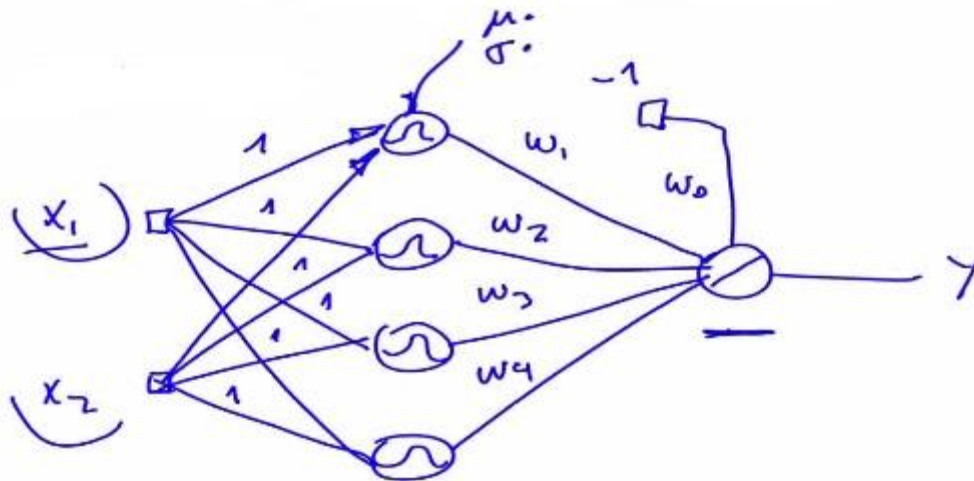
Una gráfica:

$$p(\mathbf{x}) = \sum_j w_j \phi(\|\mathbf{x} - \mu_j\|)$$



Las mus, o ese vector de medias corresponde a cada uno de los picos.

Para la arquitectura de estas neuronas, se conectan todas a un último perceptron con una función lineal, además las neuronas radiales no tienen el sesgo o vías. Además las entradas a las radiales los pesos se ponen todos en 1, pero se ajusta los parámetros de la gaussiana o la media, que corresponde cada uno a una entrada



Definición de fórmulas para base radial obtenemos:

$$y_k(\mathbf{x}_\ell) = \sum_{j=1}^M w_{kj} \phi_j(\mathbf{x}_\ell)$$

$$\phi_j(\mathbf{x}_\ell) = e^{-\frac{\|\mathbf{x}_\ell - \mu_j\|^2}{2\sigma_j^2}}$$

Yk son las salidas, wk los pesos sinápticos, ϕ_{ij} es la función de base radial la gaussiana, x_i son las entradas. En la gaussiana ϕ_{ij} , vemos que es la entrada menos el vector de medias. Para cada entrada j tenemos una μ distinta. Y por último el valor de sigma el mismo para todas las dimensiones.

- **Video 016**

Primer parte del entrenamiento de neuronas de base radial. Tienen dos etapas, es un entrenamiento mixto supervisado y no supervisado. Entonces hay dos métodos para realizar este entrenamiento. RBF parámetros de la base radial.

- Método 1:
 - Adaptación no supervisada de las RBF
 - Utilizando el método k-medias
 - Utilizando mapas autoorganizativos
 - Otros...
 - Adaptación supervisada de los w_{kj} (LMS)
- Método 2:
 - Inicialización por el Método 1
 - Adaptación supervisada de las RBF $\left(\frac{\partial \xi}{\partial \mu_j}, \frac{\partial \xi}{\partial \sigma_j} \right)$
- En general se adaptan RBF y w_{kj} por separado.

Primero vamos a ver el método no supervisado, el más sencillo y más utilizado, de k medias.

Objetivos:

- Encontrar k conjuntos C_j de forma que:
 - Cada conjunto C_j sea lo más diferente posible de los demás
 - Los patrones \mathbf{x}_ℓ dentro de cada C_j sean lo más parecidos posible entre ellos
- Encontrar el centroide μ_j de cada conjunto C_j

Ecuación de optimización:
$$\min \left\{ J = \sum_{j=1}^k \sum_{\ell \in C_j} \|\mathbf{x}_\ell - \mu_j\|^2 \right\}$$

El primer algoritmo es el k medias por lotes.

1. Inicialización: se forman los k conjuntos $C_j(0)$ con patrones \mathbf{x}_ℓ elegidos al aleatoriamente.
2. Se calculan los centroides:

$$\mu_j(n) = \frac{1}{|C_j(n)|} \sum_{\ell \in C_j(n)} \mathbf{x}_\ell$$

3. Se reasignan los \mathbf{x}_ℓ al C_j más cercano:

$$\ell \in C_j(n) \Leftrightarrow \|\mathbf{x}_\ell - \mu_j\|^2 < \|\mathbf{x}_\ell - \mu_i\|^2 \quad \forall i \neq j$$

4. Volver a 2 hasta que no se realicen reasignaciones.

Una alternativa al de lotes es realizar ajuste patrón a patrón o 1 a 1 k online. El cual es optimizado por el método del gradiente.

$$\nabla_{\mu} J = \nabla_{\mu} \left\{ \sum_{j=1}^k \sum_{\ell \in C_j} \|\mathbf{x}_{\ell} - \mu_j\|^2 \right\} = 0$$

$$\mu_j(n+1) = \mu_j(n) + \eta(\mathbf{x}_{\ell} - \mu_j(n))$$

1. Inicialización: se eligen k patrones aleatoriamente y se usan como centroides iniciales $\mu_j(0) = \mathbf{x}'_{\ell}$.

2. Selección:

$$j^* = \arg \min_j \{ \|\mathbf{x}_{\ell} - \mu_j(n)\| \}$$

3. Adaptación:

$$\mu_{j^*}(n+1) = \mu_{j^*}(n) + \eta(\mathbf{x}_{\ell} - \mu_{j^*}(n))$$

4. Volver a 2 hasta no encontrar mejoras significativas en J .

- **Video 017**

Segunda parte del entrenamiento. Ya se entrenó la capa radial, por lo tanto ahora hay que entrenar el perceptron lineal para la salida.

Se basa en una simplificación que es que se deja la capa radial fija y luego entrenamos el perceptron simple, que permite obtener un algoritmo bastante elemental. Por lo tanto las salidas de las neuronas radiales la podemos considerar entradas fijas al perceptron simple, pudiendo obviar todo el proceso de la capa radial.

Pero este entrenamiento podemos usar el ya visto por gradiente descendente o una nueva alternativa la pseudo inversa $\frac{y}{\phi(x_I)} = W$.

Detallando para este caso el método del gradiente descendente.

$$e_k(n) = y_k(n) - d_k(n)$$

$$\begin{aligned} \xi(n) &= \frac{1}{2} \sum_k e_k^2(n) = \frac{1}{2} \sum_k \left(\sum_j w_{kj}(n) \phi_j(n) - d_k(n) \right)^2 \\ \frac{\partial \xi(n)}{\partial w_{kj}(n)} &= (y_k(n) - d_k(n)) \frac{\partial}{\partial w_{kj}} \left(\sum_j w_{kj}(n) \phi_j(n) - d_k(n) \right) \\ \frac{\partial \xi(n)}{\partial w_{kj}(n)} &= e_k(n) \phi_j(n) \end{aligned}$$

Obteniendo la regla de aprendizaje.

$$w_{kj}(n+1) = w_{kj}(n) - \eta e_k(n) \phi_j(n)$$

$$w_{kj}(n+1) = w_{kj}(n) - \eta \left(\sum_i w_{ki}(n) \phi_i(n) - d_k(n) \right) \phi_j(n)$$

Ventajas y desventajas de las redes de función radial y perceptron multicapa.

RBF-NN

1 capa oculta

distancia a prototipos gaussianos

representaciones locales sumadas

convergencia más simple (linealidad)

entrenamiento más rápido

arquitectura más simple

combinación de diferentes paradigmas de aprendizaje

MLP

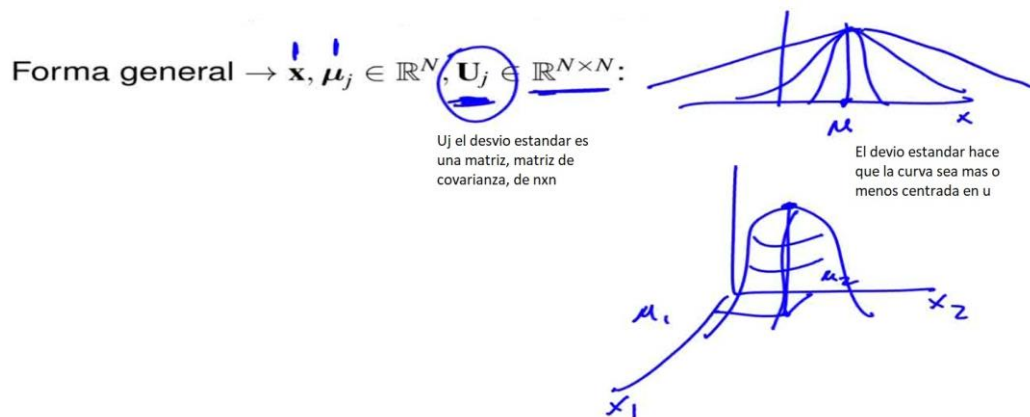
p capas ocultas

hiperplanos sigmoideos

representaciones distribuidas combinadas

• Video 018

Gaussianas n-dimensionales de la capa oculta, observaciones.



Entonces obtenemos la ecuación general.

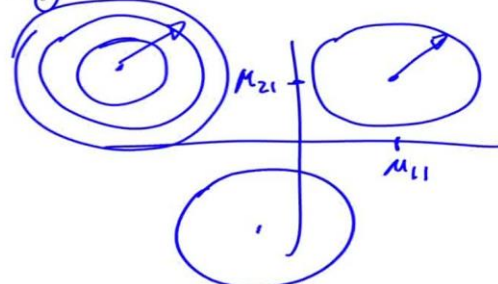
$$\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_j, \mathbf{U}_j) = \underbrace{\frac{1}{(2\pi)^{N/2} |\mathbf{U}_j|^{1/2}}}_{\text{Es una constante}} \cdot e^{-\frac{1}{2} [(\mathbf{x} - \boldsymbol{\mu}_j)^T \mathbf{U}_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j)]}$$

Handwritten notes: $\|\mathbf{x} - \boldsymbol{\mu}_j\|^2$ and $\frac{1}{|\mathbf{U}_j|}$ are written above the equation.

Entonces analizando distintos casos, el primero consideramos la matriz de covarianza como la identidad

- Caso simplificado 3 $\rightarrow \mathbf{U}_j = \mathbf{I}$:

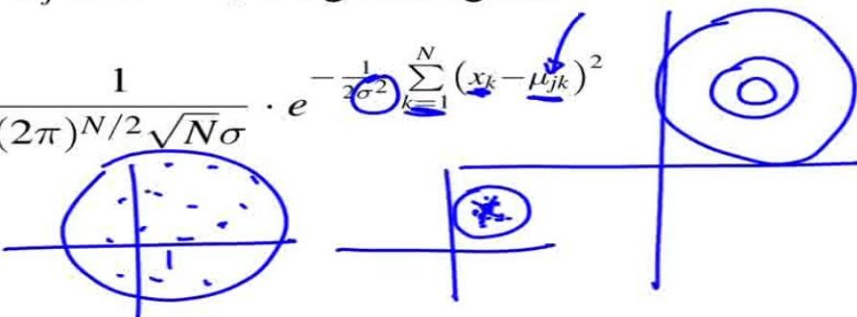
$$\mathcal{N}'(\mathbf{x}, \boldsymbol{\mu}_j) = e^{-\frac{1}{2} \sum_{k=1}^N (x_k - \mu_{jk})^2}$$



Como la varianza es 1 nos da un círculo, podemos tener varias neuronas pero cada una es un círculo, porque el desvío estándar al estar fijo no nos deja mover, pero luego con los pesos en el perceptron simple le podemos dar más o menos importancia, evitando así tener que calcular las desviaciones para modificar los círculos. Por lo general con este modelo se pueden resolver muchos problemas.

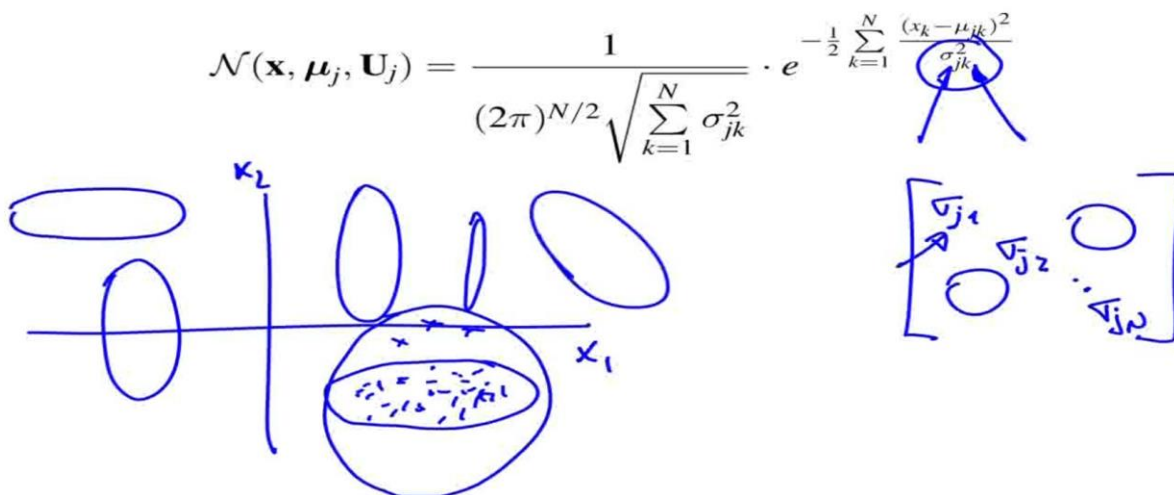
El segundo caso tenemos que la diagonal de U_j es igual, o sea son las misma pero distinto de 1. Una sigma grande permite un círculo grande y una sigma pequeña un círculo pequeño, simplemente calculando las desviaciones de los patrones desde el centroide, método k medias.

- Caso simplificado 2 $\rightarrow U_j \in \mathbb{R}^{N \times N}$, diagonal igual:

$$\mathcal{N}(\mathbf{x}, \mu_j, U_j) = \frac{1}{(2\pi)^{N/2} \sqrt{N}\sigma} \cdot e^{-\frac{1}{2\sigma^2} \sum_{k=1}^N (x_k - \mu_{jk})^2}$$


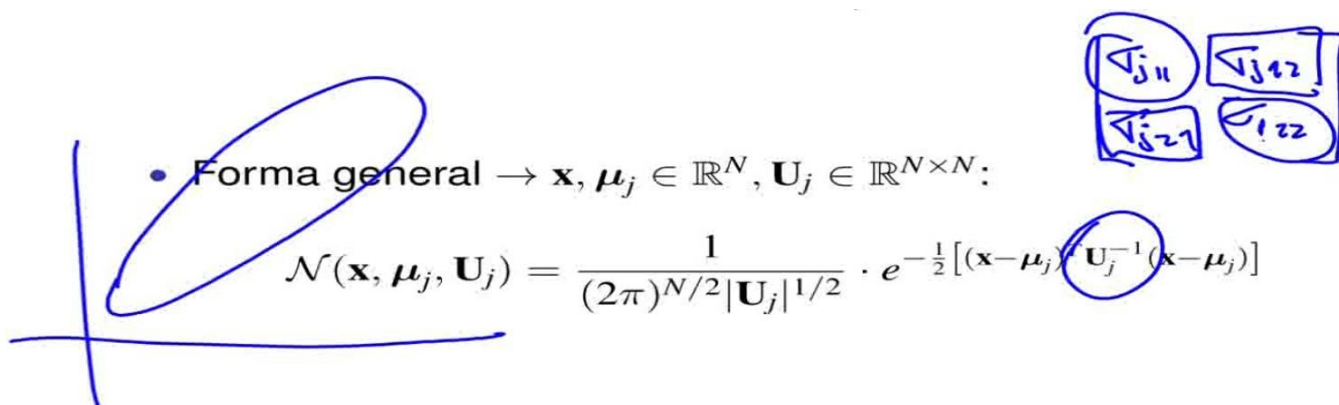
El último caso es la diagonal general, en el cual el sigma esta dentro de la sumatoria y depende de cada una de las neuronas y cada una de las dimensiones. Permitiendo que cada una de las dimensiones, de cada neurona tenga un varianza diferente, permitiendo que dejen de ser círculos perfectos, aunque únicamente pueden ser verticales u horizontales, no inclinadas.

- Caso simplificado 1 $\rightarrow U_j \in \mathbb{R}^{N \times N}$, diagonal general:

$$\mathcal{N}(\mathbf{x}, \mu_j, U_j) = \frac{1}{(2\pi)^{N/2} \sqrt{\sum_{k=1}^N \sigma_{jk}^2}} \cdot e^{-\frac{1}{2} \sum_{k=1}^N \frac{(x_k - \mu_{jk})^2}{\sigma_{jk}^2}}$$


Entonces el último caso es el más general, donde hay correlación entre las dimensiones, no son independientes unas de las otras y tenemos la matriz U_j completa.

- Forma general $\rightarrow \mathbf{x}, \mu_j \in \mathbb{R}^N, U_j \in \mathbb{R}^{N \times N}$:

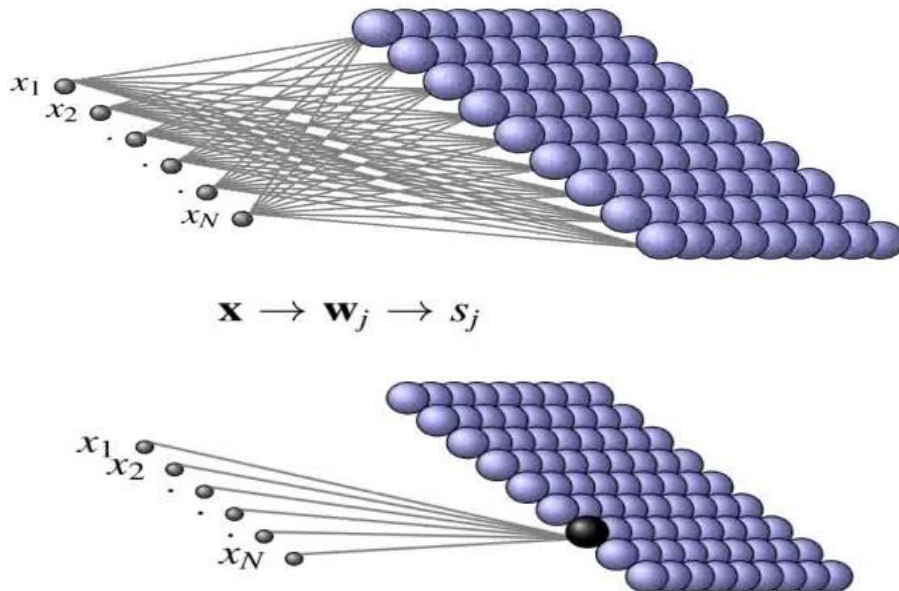
$$\mathcal{N}(\mathbf{x}, \mu_j, U_j) = \frac{1}{(2\pi)^{N/2} |U_j|^{1/2}} \cdot e^{-\frac{1}{2} (\mathbf{x} - \mu_j)^T U_j^{-1} (\mathbf{x} - \mu_j)}$$


MAPAS AUTOGENERADOS

- Video 019

Primero definimos la auto-organización, es el proceso en el cual, por medio de interacciones locales, se obtiene un ordenamiento global. La idea general es que hay algunas interacciones locales que hacen de un comportamiento global mucho más complejo, con la definición de algunos elementos de la estructura hace que mirando la estructura en general se vean comportamientos bastante complejos, que van más allá de la simple regla de cómo se comporta cada uno de los elementos. Ejemplo: una colonia de hormigas.

La arquitectura de estas redes es de una capa, o sea esta en un plano y cada neurona recibe entradas, por lo tanto se conectan todos contra todos. Es la más básica y hay muchas variantes. Y como ya vimos cada entrada tiene un peso para cada neurona, por lo tanto tenemos una matriz de pesos.

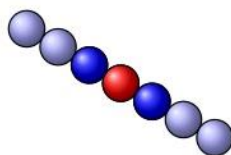


Como es la salida de este sistema, cuando entra un estímulo a la red hay una de las neuronas que se activa, por lo tanto son arquitecturas competitivas, es decir cuando llega una entrada se hace una comparación entre la entrada y todos los vectores de peso, y aquel que tiene la menor distancia o sea el vector de pesos que se parece más a la entrada define cual es la neurona ganadora.

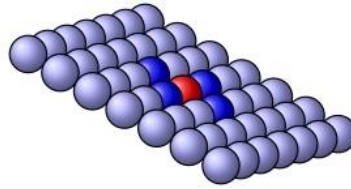
$$j^*(n) = G(\mathbf{x}(n)) = \arg \min_j \{ \|\mathbf{x}(n) - \mathbf{w}_j(n)\| \}$$

- Video 020

Vemos las formas básicas de cómo se pueden organizar las neuronas, como vimos es un plano de neuronas pero podemos hacer una simplificación a una línea de neuronas, que cuando una se activa puede afectar a su entorno.



La segunda manera de organización es una cuadrícula o plano, que también cuando se activa afecta al resto.



Una tercera forma es que sea hexagonal la división del plano que es bastante utilizada.



Entonces estos entornos que se definen cuando se activa una neurona, que es muy importante en el funcionamiento de estos mapas, a lo largo del entrenamiento estos entornos pueden tener alcances fijos o variables. Además cuando se activa el ganador el entono puede tener una excitación pura, inhibición pura o una función general excitación/inhibición.

Tomando un primer esquema de excitación/inhibición lateral.

- **Uniforme (entorno simple):** menor a 2 se activa

$$\Lambda_G(n) = 2 \Rightarrow \begin{cases} h_{G,i} = \beta(n) & \text{si } |G-i| < 2 \\ h_{G,i} = 0 & \text{si } |G-i| > 2 \end{cases}$$

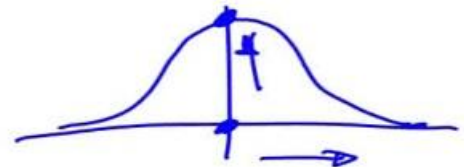
distancia 2 mas lejano nada

siendo $\beta(n)$ adaptable en función de las iteraciones n .

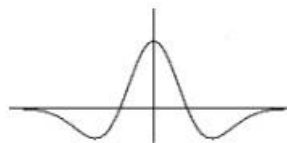
Una alternativa es usar una gaussiana, si excita o inhibe depende de lo que vale y será una función dependiente de la distancia, cuanto más lejos este la neurona de la que se activa menos la va a influenciar.

- **Gaussiana:**

$$h_{G,i} = \beta(n) e^{-\frac{|G-i|^2}{2\sigma^2(n)}}$$



Otra función que es muy utilizada es el sombrero mejicano, que se demostró que es la que se utiliza en la retina, tiene una pequeña zona de inhibición, una zona central y después se va atenuando.



- **Video 021**

Como se realiza el entrenamiento de estas redes. Es totalmente no supervisado, es competitivo ya que aprende la neurona ganadora un entorno cercano no todas.

Algoritmo de entrenamiento

1. Inicialización:

- pequeños valores aleatorios con $w_{ji} \in [-0,5, \dots, +0,5]$ o también
- eligiendo aleatoriamente $\mathbf{w}_j(0) = \mathbf{x}_\ell$ ($\ell \in [1, \dots, L]$)

2. Selección del ganador:

$$G(\mathbf{x}(n)) = \arg \min_{\forall j} \{ \|\mathbf{x}(n) - \mathbf{w}_j(n)\| \}$$

3. Adaptación de los pesos:

$$\mathbf{w}_j(n+1) = \begin{cases} \mathbf{w}_j(n) + \eta(n) (\mathbf{x}(n) - \mathbf{w}_j(n)) & \text{si } y_j \in \Lambda_G(n) \\ \mathbf{w}_j(n) & \text{si } y_j \notin \Lambda_G(n) \end{cases}$$

4. Volver a 2 hasta no observar cambios significativos.

Consideraciones del entrenamiento.

- $\Lambda_G(n)$ es generalmente cuadrado
- $h_{G,i}(n)$ uniforme en i , decreciente con n
- $0 < \eta(n) < 1$ decreciente con n

Como varia el entorno y la tasa de aprendizaje se definen varias etapas del entrenamiento. Anteriormente vimos el entrenamiento de la neurona pero no del entorno el cual se procede de la siguiente manera.

1. Ordenamiento global (o topológico)

- $\Lambda_G(n)$ grande (\approx medio mapa)
- $\eta(n)$ grande (entre 0.9 y 0.7)
- Duración 500 a 1000 épocas

2. Transición

- $\Lambda_G(n)$ se reduce linealmente hasta 1
- $\eta(n)$ se reduce lineal o exponencialmente hasta 0.1
- Duración ≈ 1000 épocas

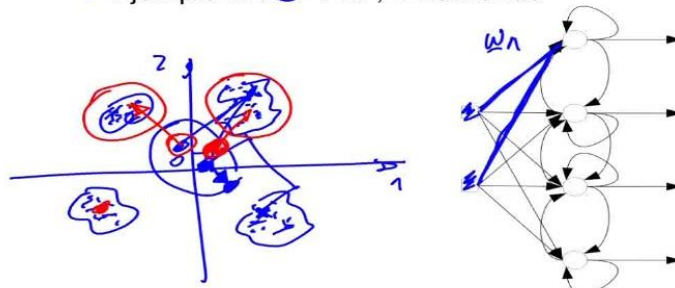
3. Ajuste fino (o convergencia)

- $\Lambda_G(n) = 0$ (sólo se actualiza la ganadora)
- $\eta(n) = cte$ (entre 0.1 y 0.01)
- Duración: hasta convergencia (≈ 3000 épocas)

• Video 021

Se observa un ejemplo práctico sencillo.

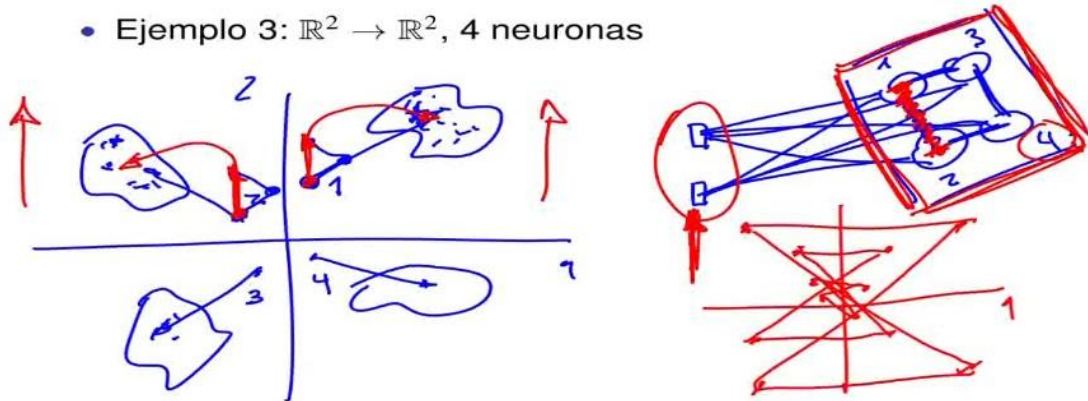
- Ejemplo 1: $\mathbb{R}^1 \rightarrow \mathbb{R}^1$, 2 neuronas
- Ejemplo 2: $\mathbb{R}^2 \rightarrow \mathbb{R}^1$, 4 neuronas



- **Video 022**

Más ejemplos prácticos.

- **Ejemplo 3: $\mathbb{R}^2 \rightarrow \mathbb{R}^2$, 4 neuronas**



- **Video 023**

Como utilizar un mapa autoorganizativo como un clasificador. Lo usábamos únicamente para entrenarlo de manera no supervisada y que agrupe cosas que se parecen entre sí. Hay varias formas de usarlos como clasificador pero vamos a ver la más sencilla.

- **Video 024**

Cuantización vectorial con aprendizaje.

Conceptos básicos:

- Cuantizador escalar: señales cuantizadas
- Cuantizador vectorial:
 - centroides o prototipos
 - diccionario o code-book
 - el proceso de cuantización: de vectores a números enteros
- Ideas de cómo entrenarlo:
 - algoritmo k -means etiquetado para clasificación
 - SOM etiquetado como clasificador...
 - algoritmo supervisado LVQ

Algoritmo de aprendizaje LVQ1

1. Inicialización aleatoria

2. Selección:

$$c(n) = \arg \min_i \{ \|\mathbf{x}(n) - \mathbf{m}_i(n)\| \}$$

3. Adaptación:

$$\mathbf{m}_c(n+1) = \mathbf{m}_c(n) + s(c, d, n) \alpha [\mathbf{x}(n) - \mathbf{m}_c(n)]$$

$$s(c, d, n) = \begin{cases} +1 & \text{si } c(n) = d(n) \\ -1 & \text{si } c(n) \neq d(n) \end{cases}$$

4. Volver a 2 hasta satisfacer error de clasificación

Observaciones:

- Interpretación gráfica
 - Caso de clasificación correcta
 - Caso de clasificación incorrecta
- No hay arquitectura neuronal
- Se puede ver al cuantizador como SOM lineal, sin entorno y supervisado
- Velocidad de aprendizaje
 - ¿Existe un α_c óptimo para cada centroide?
 - ¿Se debe considerar un $\alpha_c(n)$ óptimo para cada instante de tiempo?

Velocidad de aprendizaje:

$$\begin{aligned}\mathbf{m}_c(n+1) &= \mathbf{m}_c(n) + s(n)\alpha(n) [\mathbf{x}(n) - \mathbf{m}_c(n)] \\ \mathbf{m}_c(n+1) &= \mathbf{m}_c(n) + s(n)\alpha(n)\mathbf{x}(n) - s(n)\alpha(n)\mathbf{m}_c(n) \\ \mathbf{m}_c(n+1) &= [1 - s(n)\alpha(n)] \mathbf{m}_c(n) + s(n)\alpha(n)\mathbf{x}(n) \\ \mathbf{m}_c(n+1) &= \\ &= [1 - s(n)\alpha(n)] \\ &\quad \{ \mathbf{m}_c(n-1) + s(n-1)\alpha(n-1) [\mathbf{x}(n-1) - \mathbf{m}_c(n-1)] \} \\ &\quad + s(n)\alpha(n)\mathbf{x}(n)\end{aligned}$$

$\mathbf{x}(n-1)$ es afectado dos veces por α

Siendo $\alpha < 1$ la importancia relativa de los primeros patrones de entrenamiento siempre será menor que la de los últimos.

Si queremos que α afecte por igual a todos los patrones deberemos hacerlo decrecer con el tiempo.

Se debe cumplir que:

$$\alpha_c(n) = [1 - s(n)\alpha_c(n)] \alpha_c(n-1)$$

Demostrar que:

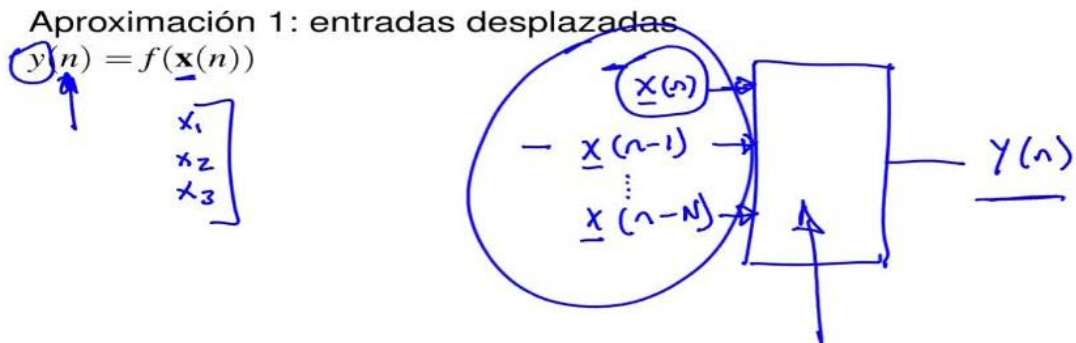
$$\alpha_c(n) = \frac{\alpha_c(n-1)}{1 + s(n)\alpha_c(n-1)}$$

(no sobrepasar $\alpha > 1$)

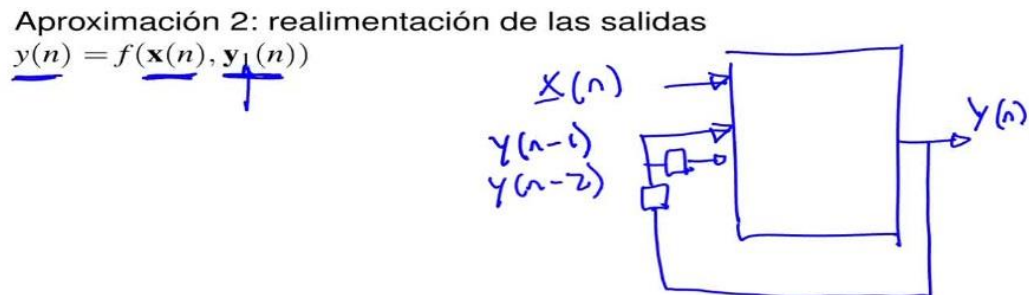
REDES NEURONALES DINAMICAS

- Video 026

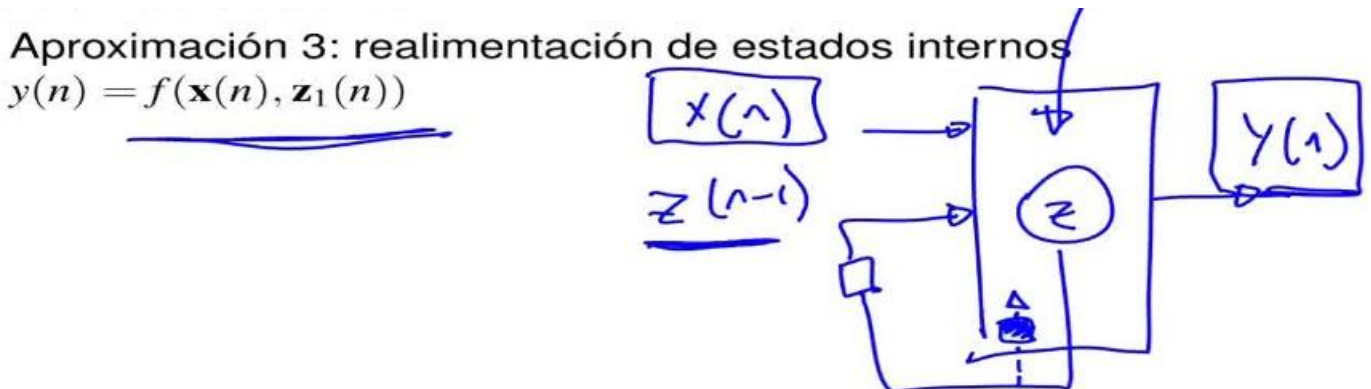
Son dinámicas ya que no solo consideran el patrón actual sino también lo que valieron en instantes anteriores, por lo tanto no considera únicamente las entradas actuales sino que tiene algún historial que también influye en la decisión que hay que tomar.



En esta primera aproximación solamente hay un desplazamiento o dinámica únicamente en el tiempo ya que considero un conjunto de entradas y las anteriores, pero la salida sigue siendo estática.



Esta segunda aproximación considera las salidas en instantes anteriores como entradas, aunque también sigue siendo una salida estática.



Esta tercera aproximación considera estados internos de la red neuronal de la capa oculta para alimentar a la red, empezando a tener un comportamiento más dinámico.

Caso general:

$$y(n) = f(\underline{x}(n), \underline{z}_1(n), \underline{y}_1(n))$$

En el caso más general podemos considerar los 3 tipos anteriores, retroalimentando con salidas, entradas e intermedios anteriores.

La clasificación de estas redes neuronales dinámicas puede ser:

- Redes con retardo en el tiempo, tienen un comportamiento dinámico aunque las podemos entrenar con los algoritmos que ya conocemos.
- Redes recurrentes, tienen realimentación hacia atrás.
 - Redes totalmente recurrentes.
 - Redes de Hopfield, memoria asociativa.
 - Redes de Boltzman, supervisadas.
 - Teoría de la resonancia adaptativa
 - Parcialmente recurrentes.
 - Retro propagación a través del tiempo
 - Redes de Elman.
 - Redes de Jordan.
- **Video 027**

Las redes de Hopfield, fueron las primeras recurrentes y tienen una importancia histórica. Se hace recurrencia desde la salida, pero no realimenta desde la que sale sino a las demás, además tienen la misma cantidad de entradas, salidas y neuronas. Después el comportamiento es bastante parecido al perceptron, se puede usar la función signo, los pesos se multiplican por la entrada y se suman, hay un umbral o sesgo, pero pueden tener 3 salidas, pero cuando es 0 la salida es la del instante anterior. Los pesos sinápticos son simétricos, y el de la misma neurona es 0.

$$y_j(n) = \text{sgn}(x) = \text{sgn} \left(\sum_{i=1}^N w_{ji} y_i(n-1) - \theta_j \right) \cdots \begin{cases} x > 0 & +1 \\ x = 0 & y_j(n-1) \\ x < 0 & -1 \end{cases}$$

$$\begin{aligned} w_{ji} &= w_{ij} \quad \forall i \neq j \\ w_{ii} &= 0 \quad \forall i \end{aligned}$$

Algunas generalidades de este tipo de redes:

- Cada neurona tiene un disparo probabilístico, hay una variable aleatoria de la cual le toca disparar en cada momento.
- Conexiones simétricas, $w_{ij}=w_{ji}$
- El entrenamiento es no supervisado, no hay una referencia o salida correcta contra la cual medir un error.
- Pueden utilizarse como memoria asociativa, en vez de ser una memoria clásica como el disco rígido que es por dirección sino que estas son accesibles por contenido devolviendo te lo más parecido a lo que le pases.

- **Video 028**

Utilizaremos las redes de Hopfield para realizar una memoria asociativa. El proceso de almacenamiento es el entrenamiento de la red y para retomar el contenido solamente se la usa o prueba la red.

El entrenamiento o almacenamiento, tenemos las memorias fundamentales, aquellos patrones o datos que queremos guardar en la red x_k^* .

$$X^* = \{ \mathbf{x}_k^* \in \mathbb{R}^N \}$$

El aprendizaje o almacenamiento se denomina Hebbiano, donde los pesos son de una neurona con otra promedia las entradas para todos los patrones haciendo un producto sobre todos los patrones k. Por lo tanto cuando dos neuronas se activan y ambas valgan positivos o negativos van a dar un valor grande, pero sino va a dar un valor menor, por lo tanto de cierta manera analiza que tanto se sincronizan las neuronas.

Es hebbiano ya que se aumentan los pesos cuando había dos neuronas que se activan juntas y se generan sinapsis entre esas dos neuronas, por lo tanto cuando las entradas están coordinadas las neuronas correspondientes tiene que activarse juntas y los pesos sinápticos se irían aumentado ya aprenden esa información.

$$w_{ji} = \frac{1}{N} \sum_{k=1}^P x_{kj}^* x_{ki}^*$$

Observaciones del entrenamiento:

- El proceso de entrenamiento no es iterativo, se le muestran una vez todos los patrones y se realizan los cálculos, por lo tanto conocemos cuanto demorara el proceso.
- w_{ji} es mayor cuando las neuronas i, j se tiene que activar juntas, regla de Hebb.
- La capacidad de almacenamiento tiene un error del 1% y está limitada $P_{max} = \frac{N}{2\ln(N)}$.

• Video 029

Como es el proceso de lectura o recuperación del contenido. Se le da la misma entrada o una parecida, como la cara de una persona desde otro Angulo y quiero recuperar la imagen limpia.

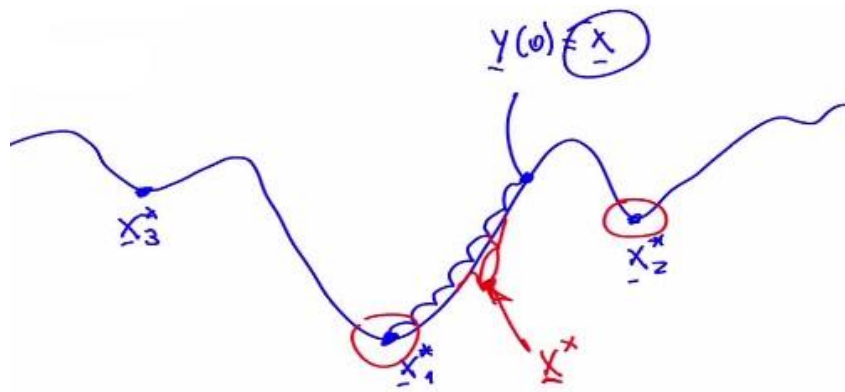
Dado un patrón x, inicializamos la salida en el instante cero con la entrada. Luego iteramos:

1. $j^* = rnd(N)$, entonces elegimos al azar entre 1 y N, seleccionando algunas de las neuronas.
2. $y_{j^*}(n) = sgn(\sum_{i=1}^N w_{ji} y_i(n-1))$, calculamos la salida para las seleccionadas en el paso anterior.
3. Volvemos al paso 1 hasta que no se observan cambios en las yj, finaliza cuando recorrimos todas las neuronas y ninguna cambia.

Algunas observaciones de este proceso:

- El proceso de recuperación es iterativo y dinámico pueden ser 3 iteraciones y se estabilizan las salidas o 500, por lo tanto no es estático no se calcula en una pasada y sale el resultado, sino que se va adaptando hasta obtener la salida.
- En general no se utilizan o no se necesitan los umbrales ϕ_{ij} .
- La salida final es y(M) cuando no hay más cambios al “recorrer” todas las salidas, Por lo tanto hay que controlar cuando se estabiliza o no hay cambios.
- Se pueden obtener estados espureos y oscilaciones, o sea nunca se estabiliza o converge la salida, sucede generalmente cuando queremos almacenar más elementos que los fundamentales o algunas que son muy parecidas

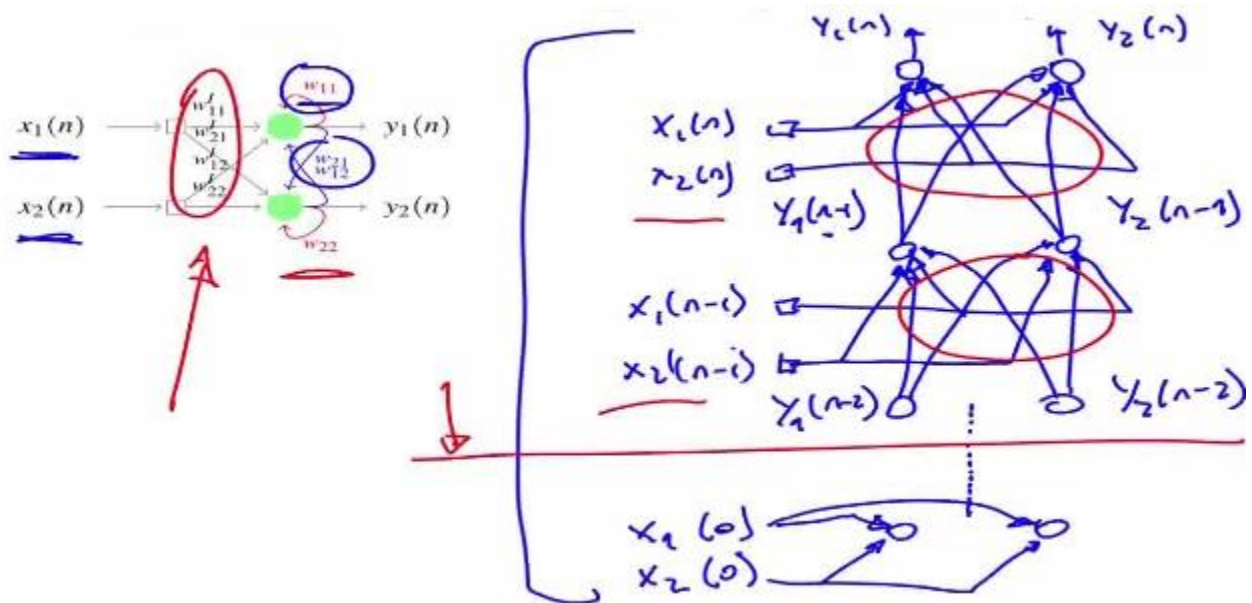
Campos energéticos de Hopfield, se estudia en detalle el caso en que una red queda oscilando porque son muy parecidos a dos fundamentales. Podemos observar en una dimensión que las fundamentales se almacenan en los valles, que esa superficie se genera durante el entrenamiento, y cuando se recupera se va acercando hacia una de las fundamentales.



- **Video 030**

Retro propagación a través del tiempo, es otra alternativa para tratar una red recurrente se definen tantas entradas como salidas y cada una de las conexiones asociadas con un peso, pero con el adicional que también se conectan o asocian las neuronas entre si incluso con si misma y también con sus propios pesos, recurrencia total.

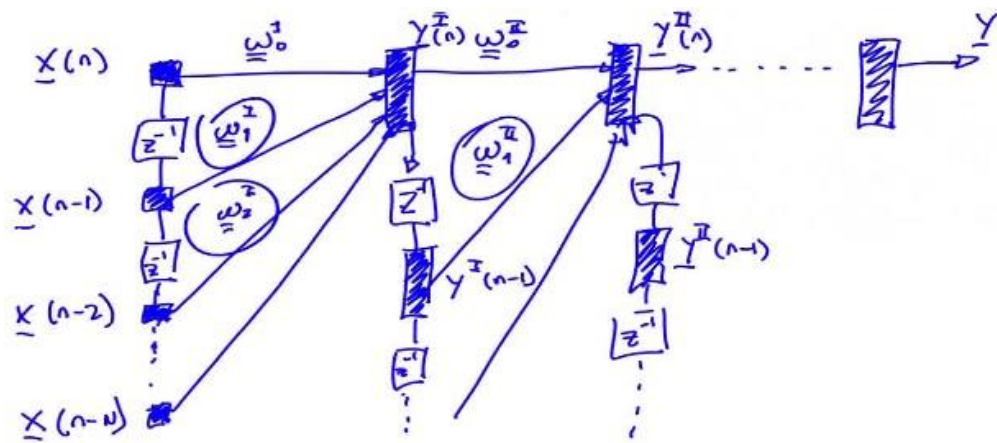
Para hacer que aprendan se utiliza el algoritmo mencionado, primero la expandimos, retropropagcion hacia adelante pura, donde además de las entradas necesitamos las salidas en el instante anterior y así sucesivamente y se lo puede entrenar con retro propagación de las multicapa.



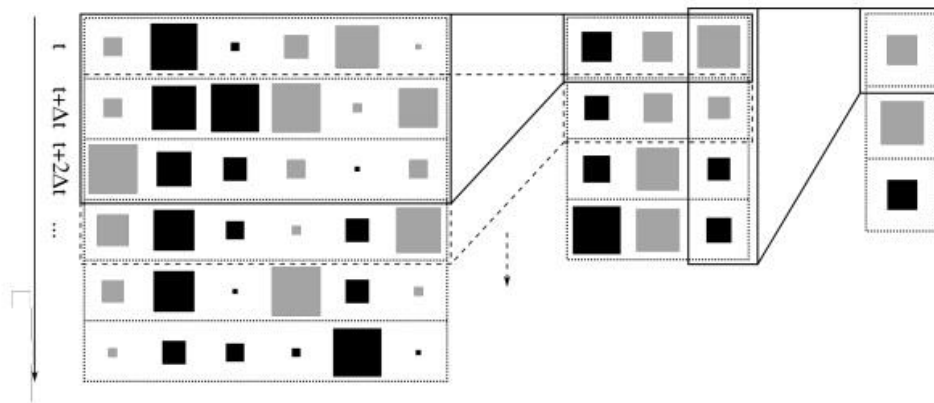
- **Video 031**

Redes neuronales con retardos en el tiempo, la arquitectura consta de un vector entrada en el instante n , una capa de neuronas y cada una asociadas a unos pesos como en el perceptron simple pero del instante 0 y así sucesivamente en las distintas capas de neuronas hasta la capa de salida. Pero además tenemos las entradas anteriores en un instante anterior, asociada a la primer capa con sus respectivos pesos y así sucesivamente tantos instantes anteriores. Hasta ahora solo estamos considerando las entradas retardadas, pero se puede aplicar también las salidas retardadas entre las capas.

Y también puede ser entrenada con propagación hacia atrás, como el perceptron multicapa, teniendo en cuenta todos los conjuntos de pesos.

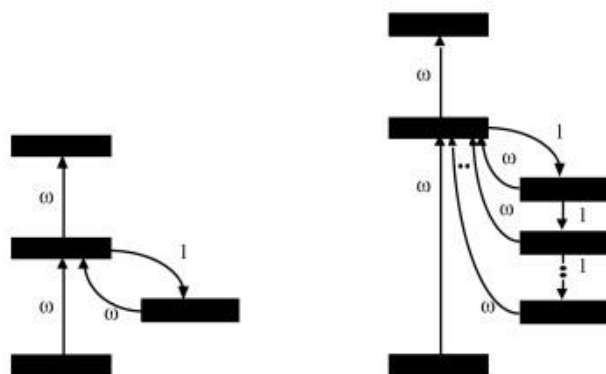


TDNN: clasificación espacio-temporal



Arquitecturas de Elman y Jordan, en Elman considera un retardo de la capa intermedia hacia sí misma la salida de la capa intermedia u oculta se guarda y el instante siguiente se vuelve a ingresar. Mientras que las de Jordan son más genéricas y considera todos los retardos anteriores.

Elman



Jordan

