uc3m | Universidad Carlos III de Madrid

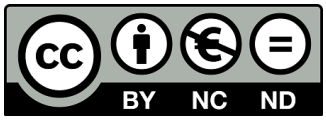
Grado en Ingeniería Informática

2021-2022

Apuntes

Redes de Neuronas Artificiales

Jorge Rodríguez Fraile[1]



[1]Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INFORMACIÓN

1.1. Profesores

Magistral: Isasi Viñuela, isasi@ia.uc3m.es

Prácticas: José María Valls, jvalls@inf.uc3m.es

Inés M. Galván, igalvan@inf.uc3m.es (de baja)

1.2. Objetivos

- "Transmitirnos su entusiasmo por las redes neuronales", darnos la posibilidad de acceder más fácilmente a este mundo.
- Entender que subyace sobre estas redes y el método científico.
- Estudiar los diferentes modelos de redes.
- Describir las diferentes áreas de aplicabilidad de las redes de neuronas.
- Resolver problemas con redes de neuronas.
- Analizar las ventajas e inconvenientes de neuronas.
- Analizar las ventajas e inconvenientes de cada uno de los modelos de redes desde una perspectiva aplicada.
- Diseñar un conjunto de experimentos para la resolución de problemas.

1.3. Sistema de evaluación

- 60 % Evaluación continua
  - 20 % Practica 1. Trata de resolver un problema de regresión, estimar un número real, y se divide en dos partes:
    - Modelo lineal (ADALINE) determinando los coeficientes, este caso tenemos programarlo nosotros en nuestro lenguaje de preferencia. Se nos proporcionan unos datos de prueba para cuando lo estemos programando salga un error muy pequeño, de esta manera sabemos si lo hacemos bien. Las de prueba son 3 atributos, pero los reales son de 8.

        - No lineal (Perceptrón multicapa), presentando una memoria de unas 8 páginas en forma de artículo. Para esta parte se nos dará un script en R, no hay que programarlo, pero hay que realizar muchas pruebas para ver los resultados (errores).
    - 20 % Practica 2. Habrá que modificar scripts de Python, también se usará Google Colab.
        - Parte 1: Problema de clasificación que resolveremos con Perceptrón multicapa.
        - Parte 2: Problema de clasificación con imágenes usando Deep Learning.
    - 20 % Prueba parcial, de las prácticas (a principios de noviembre).
- 40 % Examen final, se realizarán cuestiones teórico-prácticas, pudiendo incluir preguntas sobre las prácticas.
    - Se puede emplear material.
    - No hay nota mínima

Para la bibliografía se empleará en los primeros temas el libro escrito por los profesores (Redes de Neuronas Artificiales. Un enfoque práctico, 2004), pero para el resto de los temas sobre todo Deep Learning (Redes Neuronales & Deep Learning de Fernando Bernal, 2018).

2. PRACTICA 1

2.1. Preparación de Datos

2.1.1. Los datos

Variables, datos o patrones de entrada y de salida deseada de la red, que solo trabajan con atributos numéricos y cuando no son numéricos se discretizan para que pasen a ser numéricos.

Supervisado: Hay una salida que es lo que buscamos determinar.

No supervisado: Tratamos de agrupar o buscar una determinada estructura en los datos, no hay salida.

2.1.2. Transformación de los datos

Normalización de datos: Se emplea en los casos en los que los valores entre los atributos con muy dispares (ej. 0.1 y 10) lo que provocaría que al realizar operaciones entre ellos dieran valores que dependen mucho más de un atributo que otro. Los valores se ajustan al rango de 0 y 1.

$$ValorNormalizado_i = \frac{Valor_i - ValorMinimo_i}{ValorMaximo_i - ValorMinimo_i}$$

Aleatorización de los datos: Se altera el orden las instancias de manera que se evita cualquier sesgo a la hora de entrenar el modelo.

Eliminación de atributos irrelevantes: Nosotros no lo haremos, pero este nos permite eliminar la información que no aporta nada y entorpece el entrenamiento.

Reducción dimensionalidad: Se realizan las técnicas de reducción de dimensionalidad, que bien seleccionan un subconjunto de atributos, bien transforman los datos de entrada en otro conjunto de mayor dimensión.

2.1.3. Evaluación de una red de Neuronas

Separación del conjunto de entrenamiento y test: Se separa un conjunto de instancias que se emplearan solo para el entrenamiento y otro que se utilizara solo para evaluar el modelo, el conjunto de test. Esto permitirá detectar si el entrenamiento que estamos haciendo se está sobreajustando, haciendo que no generalizase adecuadamente.

Validación cruzada: Consiste en dividir los datos en una serie de conjuntos de igual tamaño. Con estos conjuntos se va entrenando, dejando uno fuera en todo momento para utilizarlo como test. Esto nos permite evaluar el error haciendo la media de todos los errores obtenidos con los modelos, siendo este valor error del modelo que se obtienen entrenando con todas las instancias.

Validación cruzada estratificada: Se dividen de igual las instancias en conjuntos de igual tamaño. En este caso lo que se hace es que haya en cada conjunto la misma proporción de instancias con cada clase con respecto al total. Todos tendrán la misma proporción de instancias de una clase que los otros del total. Lo que se hace es separar las instancias por la clase, de estos conjuntos generados se divide cada uno en el número de conjuntos de validación cruzada, de esta manera en cada uno de los conjuntos que vamos a generar vamos cogiendo uno conjunto de cada una de las clases.

Medida de evaluación

- **En regresión:** Error absoluto medio (Mean Absolute Error) es la media del valor absoluto de la diferencia de la salida y el deseado. Error cuadrático medio (Mean Square Error) es igual al anterior, pero en vez de valor absoluto la diferencia se pone al cuadrado.

  $$MAE = \frac{1}{N}\sum_{i=1}^{N}|d_i - s_i| \quad MSE = \frac{1}{N}\sum_{i=1}^{N}(d_i - s_i)^2$$

  Acaba cuando el error medio o el cuadrático es aceptable. También por una visualización gráfica de las salidas deseadas y las salidas de la red.

- **En clasificación:** Lo más habitual es evaluar la calidad de la red basándonos en su precisión predictiva (% de aciertos), la proporción de aciertos sobre el total de instancias.

  Acaba cuando el porcentaje de aciertos es alto, teniendo en cuenta que debe ser mayor que la probabilidad de que acierte aleatoriamente (como si tirase una moneda) y las clases están equilibradas (porque si siempre da una clase podría tener un buen porcentaje de acierto dando un valor constante).

  Matriz de confusión: Representa la cantidad de instancias según lo estimado y lo correcto, de manera que se pueda ver los que han sido estimados según TP, FN, FP o TN. Los datos correctamente clasificados están en la diagonal, los incorrectos fuera de ella. Se emplean otras medidas como son:

  - El porcentaje de aciertos total es $\frac{TP+TN}{TP+TN+FN+FP}$
  - El porcentaje de aciertos de + es: $\frac{TP}{TP+FN}$
  - El porcentaje de aciertos – es: $\frac{TN}{FP+TN}$

2.1.4. Anotaciones Practica 1

A la hora de normalizar hacerlo sobre el total de los datos, no solo sobre los de entrenamiento. Aleatorizarlos. Separar las instancias en 70 15 15.

En la parte de programar los vectores de pesos y umbrales, si se hace un vector con los pesos y el umbral, tener en cuenta que al hacer el producto cartesiano el de entrada debe tener un 1 para el umbral. $[x_1\ x_2\ x_3\ 1]$ con $[w_1\ w_2\ w_3\ \sigma]$

3. TEMA 1: INTRODUCCIÓN

El objetivo de la IA es la construcción de sistemas inteligentes.

Sistema Inteligente: Dispositivo físico o lógico con capacidades propias de los humanos como tomar decisiones, resolución de problemas o el aprendizaje.

Partiremos de la base, por lo que el primer paso serán organismos más pequeños, pero el objetivo es alcanzar la capacidad humana y llegar a sustituirnos.

Áreas

- **Simbólica (Top-down):** Diseñar componentes del sistema inteligente donde el comportamiento es programado por nosotros para que él sea capaz de razonar y aprender. Se emplea en búsqueda y optimización
- **Subsimbólica (Bottom-up):** Se programa lo básico y lo dejamos interactuar para ver si llega a un punto adecuado. No programamos el comportamiento, sino las partes base y miramos a ver como se relaciona y si lo resuelve.

Las **Redes de Neuronas** se encuentran en la IA subsimbólica, tratan de emular los sistemas neuronales biológicos. Se ha estado durante décadas detrás de la construcción de algoritmos capaces de procesar la información de la misma manera que lo hace el cerebro humano.

Se caracterizan por tener la **propiedad de aprender** a partir de ejemplos.

Tratan **información numérica**, si se puede discretizar bien, si no se emplean otros métodos.

En la actualidad existe un gran número de arquitecturas diferentes de redes de neuronas artificiales, además este progreso ha sido posible por la capacidad de cómputo, esos procesos son muy costosos.

Algunas **áreas de aplicación**:

- Reconocimiento de patrones (imagen, voz, caracteres)
- Compresión y análisis de datos.
- Robótica.
- Medicina.
- Predicción de series temporales.

Las redes de neuronas no permiten abordar problemas de: Aproximación, Predicción, Clasificación, Agrupación.

Otras técnicas de IA que se pueden emplear para estos problemas son:

- Máquinas de Vectores de Soporte (Vapnik, 1995)
- Gradient Boosting (Friedman, 2002)
- Random Forest (Breiman, 2001)
- K-medias (agrupación) (MacQueen, 1967)

Ventajas

- Son fáciles de utilizar y diseñar, la parte difícil es acertar.
- Capacidad de aprender a partir de ejemplos.
- Tolerancia al ruido en los ejemplos, es muy tolerante a fallos, incluso a la perdida de unas pocas neuronas.
- Diversidad de las áreas de aplicación, teniendo datos se puede aplicar en problemas de cualquier área aunque no seamos expertos en la misma.

Desde la aparición de las redes sociales y móviles las compañías han recogido muchísimos datos que han posibilitado el entrenamiento de redes en muchas áreas.

Problemas específicos de las RNA

- Tiempo de aprendizaje, requiere mucha capacidad de cómputo y son lentos.
- Determinación de los hiperparametros, como el número de neuronas, ciclos, etc.
- No se pueden explicar los resultados, como una caja negra por lo que se están diseñando redes cuyo propósito es explicar.

3.1. Historia

Comienza en **1940** con estudios en la neurocirugía, al descubrir la neurona, se empezó a ver el funcionamiento del cerebro y se vio que funcionaba por señales eléctricas. A raíz de esto se hicieron modelos computaciones basados en lo que se conocía, se crearon para puertas lógicas inicialmente (con pesos fijos). El primer modelo fue de S. McCulloch y W. Pitts, que permitía pesos que se ajustaban, pero no se aprendían.

F. Rosenblatt en **1957** diseñó el Perceptrón que tenía un proceso para calcular los pesos según las entradas, en esos momentos la entrada era binaria.

B. Widrow y M. Hoff en **1960** diseñaron el **ADALINE** que permitía entradas no binarias.

Faltaban neuronas por falta de hardware, pero se pensaba que ya estaba todo hecho.

M. Minsky y S. Papert con el **libro "Perceptrón"** en **1969** rompieron el pensamiento de que todo estaba hecho con el **XOR Problem**, en el que no existe un único plano que represente la logia de la puerta XOR. Este problema inició la Época oscura ("AI Winter").

D. Rumelhart, G. Hinton y R. William en **1986** solucionaron el problema XOR con más de 1 capa en el perceptrón, lo que da origen al **Perceptrón Multicapa y la Retropropagación.**

V. Vapnik y C. Cortes en **1995** desarrollaron las Máquinas de Vectores de Soporte (SVM) que están relacionados con problemas de clasificación y regresión.

En el año **2006** ya había capacidad de cómputo, se podía dar más profundidad con un mayor número de capas **Redes de Neuronas Profundas**, pero no se avanzó mucho en la teoría, se basaba en lo que había dicho Rumelhart.

3.2. Fundamentos biológicos

El **Sistema nervioso** es un conjunto de células especializadas en la conducción de señales eléctricas. Capta estímulos del entorno o internos, procesa la información y genera respuestas diferentes según la situación. No todos son igual de complejos, la estructura es lo que varía, pero todos están compuestos por los mismos componentes.

El **Cerebro** es el órgano principal del Sistema Nervioso. Se encarga de regular y mantener cada función vital del cuerpo, y es el órgano donde reside la mente y la conciencia del individuo. Existen distintas áreas, cada una más sofisticada para una serie de tareas (como puede ser la motora, sensora, etc.), pero todas son importantes para el individuo.

- Está formado por 10^{11} células, llamadas neuronas, interconectadas entre sí (cada una a otras 1000-10000) y forman una densa red con 10^{15} conexiones.

Las neuronas se comunican a través de **impulsos eléctricos de corta duración, que se transmiten a través de las sinapsis**.

El impulso se propaga a través del axón, y es recibido por la dendrita (ramificaciones de la neurona que buscan axones) de la célula receptora.

El **Axón** (una sola por neurona) se ramifica, y puede propagar una misma señal a cientos de dendritas de células diferentes. Contiene unas vesículas que contienen los neurotransmisores, que al liberarse se comunican con la neurona inmediata por medio de la sinapsis. La señal que se propaga es única y es recibida por todas las células conectadas.

Las **Dendritas** reciben multitud de señales diferentes, algunas excitatorias y otras inhibitorias.

Cuando el impulso llega a la neurona pre-sináptica, abre los canales de ion cálcico que liberan los neurotransmisores que son captados por los receptores de la neurona post-sináptica que abren o cierran los canales iónicos de la dendrita generando potenciales eléctricos que permiten la propagación de la señal hacia el soma de la célula. Se va despolarizando al recibir estímulos eléctricos, cuando supera un umbral se dispara un potencial de acción.

Son elementos no lineales: Si se supera un umbral, se produce una señal de activación

El sistema nervioso funciona como una enorme malla que propaga señales electroquímicas de unas células a otras, modificando sucesivamente la concentración de iones de las sinapsis.

Robustez y tolerancia a fallos

Adaptación y plasticidad

Manejar información: inconsistente, difusa y con ruido

Una **red artificial (ResNet152) puede llegar a los 60 millones de sinapsis** (diez millones de veces más pequeña), todavía estamos lejos.

3.3. Neurona artificial

Trata de imitar las características generales de las neuronas.

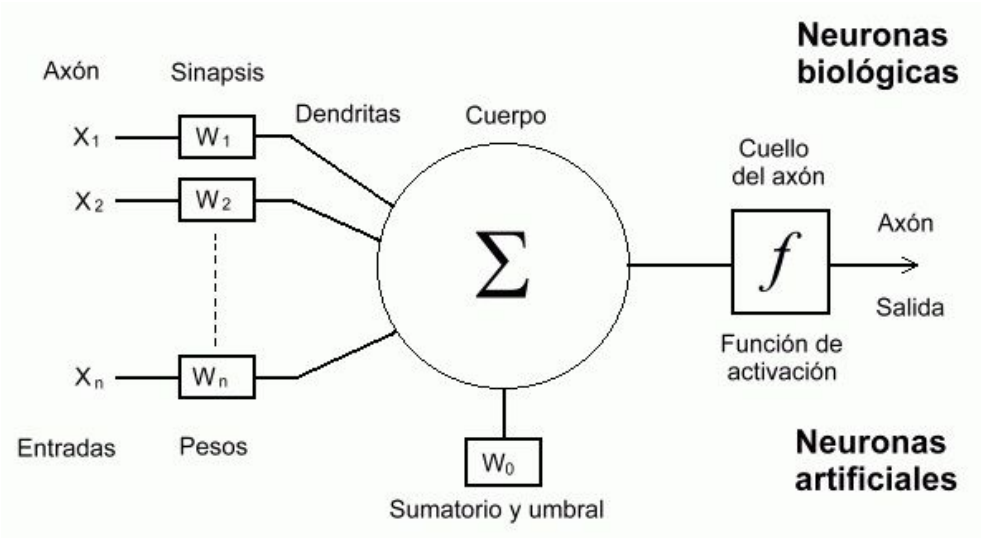


Fig. 3.1: Comparación neurona bilógica y neurona artificial

Las **dendritas funcionan como las entradas** en la neurona, el **axón como la salida** y la **densidad de la sinapsis para que pase mejor el impulso serían los pesos**.

En cuanto al **potencial de acción (cuando supera el umbral) se simula con una función de activación** no lineal. Se utiliza un **umbral para poder ajustar esta función**.

Cada neurona está caracterizada por un estado interno denominado nivel de activación:

- Discreto: Desactivado o Activado, S=0,1.
- Continuo: S=[0,1].

Función de activación: permite cambiar el nivel de activación a partir de las señales de entrada.

Las señales de entrada se combinan entre sí, generando la entrada total.

Tipos de funciones de activación:

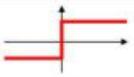
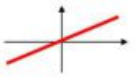
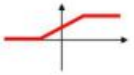
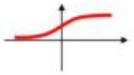
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Fig. 3.2: Tipos de funciones de activación

Las que nosotros emplearemos con la Sigmoide, Tangente hiperbólica y la ReLU (Rectified Linear Unit, es lineal a partir de 0).

Excepto la función lineal estas propagan la señal de manera no lineal.

3.3.1. Modelo computacional

Componentes

- Conjunto de unidades de proceso o neuronas artificiales.
- Conexiones entre las unidades.
- Regla de propagación, para propagar las entradas hacia la salida de la red.
- Regla de aprendizaje para modificar los pesos de las conexiones.

Tiene una arquitectura en forma de capas (entrada, ocultas y salida) que se propaga hacia delante (izq. a dch.).

Las capas ocultas son las que realizan el procesamiento, la entrada y salida solo se encargan de recibir y sacar los datos de la red.

Las conexiones van de una capa a la inmediata siguiente, no saltan capas o retroceden (todavía no). Todas las neuronas de una capa están conectadas con cada una de la capa anterior y posterior (full connected).

El aprendizaje consiste en ajustar los pesos de tal manera que pueda resolver el problema de manera eficaz.

3.3.2. Aprendizaje

Tratamos de que al pasar un patrón (serie de datos de entrada) nos dé la salida correspondiente, para esto cuando falla hay que ajustar los pesos y umbral.

Se realiza a partir de un conjunto de ejemplos o muestras conocidas, que llamaremos Conjunto de entrenamiento. Se debe disponer de una cantidad significativa y representativa de ejemplos para realizar un buen aprendizaje, suficientes ejemplos y que cubran la variedad de salidas.

Proceso:

1. Introducir progresivamente los ejemplos de aprendizaje.
2. Modificar los pesos siguiendo una ley o ecuación de aprendizaje.
3. Cuando se han introducido todos se ha completado una iteración. Ahora se comprueba si se cumple cierto criterio de convergencia y si no se cumple, se repite el proceso.

La modificación de los pesos puede ocurrir después de introducir cada patrón (lo habitual) o después de introducir todos los patrones (o un lote de estos, permite acelerar el proceso y se usa en Redes Convolucionales).

En cuanto a la finalización del aprendizaje se hace en función de un criterio de convergencia que puede ser que el error sea menor que un valor dado (puede no llegar a ocurrir) o que los pesos dejen de sufrir modificaciones y el error se estabilice.

Modelos de aprendizaje:

- **Aprendizaje supervisado:** Se conoce la salida de los ejemplos de aprendizaje. Utiliza un proceso externo para determinarlo. Conocer el resultado nos permite manejar el error cometido y guiar el aprendizaje
- **Aprendizaje no supervisado:** No se conoce la salida de los ejemplos. Permite detectar características de los datos, agruparlos o reducir la dimensionalidad. Se reajustan automáticamente los pesos y autoorganiza la información.
- **Aprendizaje por refuerzo:** Se aprende por prueba y error guiado por una recompensa si lo hace bien. No existe medida de error, solo si lo ha habido.

3.3.3. Validación

El modelo resultante debe ser capaz de responder de forma adecuada ante datos desconocidos. Meter datos no conocidos y usarlos solo para evaluar (no aprender) si el modelo es bueno.

El conjunto de datos disponibles se divide en subconjuntos de:

- **Entrenamiento:** Usado para ajustar el valor de los pesos.
- **Validación (no siempre se utiliza este conjunto):** Usado para determinar los hiperparámetros de la red y detener el aprendizaje. Permite detectar sobre ajuste.
- **Test:** Usado para medir la capacidad de generalización de la red. Se utiliza este segundo no conocido porque no es conocido y no se ha utilizado previamente (ni entrenar, ni para el entrenamiento) son verdaderamente nuevos para el modelo.

Los conjuntos de validación y test deben ser Independientes del aprendizaje, Significativos y representativos.

Sobreajuste (overfitting): La red se ha sobreajustado a los ejemplos de entrenamiento, deja de generalizar bien.

3.3.4. Taxonomía

Redes Feedforward supervisadas: Conexiones hacia delante y aprendizaje supervisado.

- Perceptrón simple, Adaline, Perceptrón Multicapa, Redes de Base Radialy Redes Convolucionales.

Redes no supervisadas

- Kohonen y ART.

Redes recurrentes: Conexiones en todas las direcciones

- Solo unas pocas conexiones recurrentes: Red de Jordan, Red de Elman
- Totalmente recurrentes: Backpropagation a través del tiempo, Long short-term memory.

3.3.5. Diferencia de las RNA y Biología

Estamos muy lejos todavía, el cerebro tiene 10 millones más de sinapsis y no necesita capas de neuronas. Siempre está aprendiendo (y olvidando).

El cerebro trabaja de manera asíncrona (se ajusta cuando lo necesita) y las RNA son síncronas (cuando se actualizan los pesos lo hacen todos a la vez).

Las RNA para aprender usan el descenso de gradiente, pero del cerebro no se conoce.

3.4. Primeros modelos

3.4.1. Células de McCulloch-Pitts

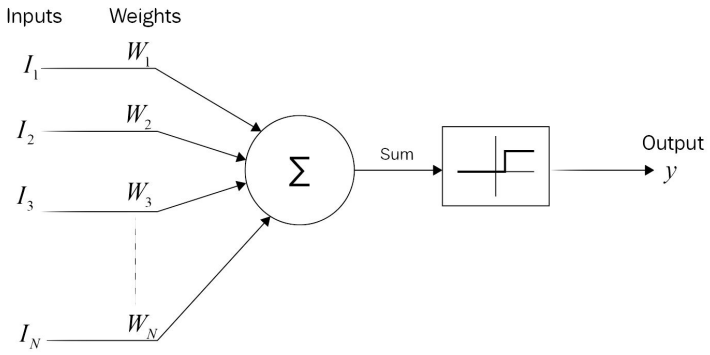


Fig. 3.3: Estructura de Célula de McCulloch-Pitts

Consiste en unas entradas que tienen asignado manualmente un peso, para cada una de las cuales se hace la suma de las entradas por su peso, este resultado se pasa por la función de activación no lineal y se obtiene la salida.

$$y = f\left(\sum_{i=1}^{n} I_i w_i\right) \quad f(x) = \begin{cases} 1 & si \quad x > 0 \\ 0 & en\ caso\ contrario \end{cases}$$

En este momento hablamos de células individuales, no de redes.

Limitaciones

Se puede simular cualquier puerta lógica. Uniendo las puertas lógicas, se puede representar cualquier función.

El problema es el diseño de la red, que requiere determinar la arquitectura y los pesos. Se necesita un mecanismo para que determine los pesos automáticamente, es muy pesado hacerlo a mano.

El perceptrón son un conjunto de células de McCulloch-Pitts, con una sola capa, y cuyos pesos pueden ser determinados a partir de los ejemplos de los que se disponga.

3.4.2. Perceptrón

Tiene una estructura similar a las células de McCulloch-Pitts, pero además de los pesos hay un umbral.

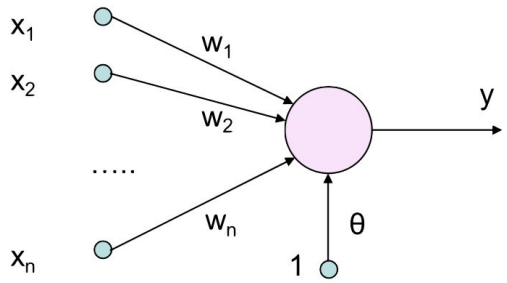


Fig. 3.4: Estructura del Perceptrón

La salida de cada célula se calcula mediante:

$$y = \mathcal{F}\left(\sum_{i=1}^{n} x_i w_i + \theta\right) \quad \mathcal{F}(s) = \begin{cases} 1 & si \quad x > 0 \\ -1 & en\ caso\ contrario \end{cases}$$

La función escalón lo hace discriminante, si solo hay una célula y da 1 pertenece a la categoría A y si es -1 a la categoría B.

El número de células de un Perceptrón se corresponde con la dimensión de la entrada, y el número de células de salida, con el número de clases a discriminar.

En el caso binario, dos entradas una salida, el perceptrón representa una recta que divide el plano en dos partes, cada una categoría.

Aprendizaje

El proceso de aprendizaje del perceptrón consiste en descubrir los parámetros de una recta que deje a todos los ejemplos de la clase A en un lado, y a los de la clase B en el otro.

Dado un conjunto de ejemplos de prueba distribuidos de los que se conoce su categoría, se trata de obtener la ecuación del hiperplano que deja a un lado los ejemplos de un tipo, y a otro los del otro.

$$A = (\vec{a}_1, ..., \vec{a}_n) \quad B = (\vec{b}_1, ..., \vec{b}_n)$$

$$\forall \vec{a} \in A, , w_1 a_1 + \cdot + w_n a_n + \theta > 0$$

$$\forall \vec{b} \in B, , w_1 b_1 + \cdot + w_n b_n + \theta < 0$$

Para todos los puntos de A deben dar la misma categoría y los de B deben dar la otra.

El proceso de aprendizaje necesita de un conjunto de ejemplos χ para los cuales tenemos una respuesta d(χ), y es un aprendizaje por refuerzo:

- Si la red da una respuesta correcta, no se modifican los pesos.
- Si da una respuesta incorrecta, se modifican los pesos una cantidad fija.

El incremento de los pesos se realiza mediante la expresión: $\Delta w_i = d(\vec{x})x_i, , x \in \chi$

Si la **salida obtenida es y($\vec{x}$) = 1, y la deseada es d($\vec{x}$) = -1, entonces** $w_i = -x_i$. La salida es mayor que la esperada, y los pesos se decrementan

Si la **salida obtenida es -1, y la deseada es 1, entonces** $w_i = x_i$**.** La salida es menor que la esperada, y los pesos se incrementan.

Validación

Se realiza de forma iterativa.

1. Inicialización aleatoria de los pesos y el umbral de la red.
2. Se toma un ejemplo entrada-salida $\vec{x} = (x_1, x_2, ..., x_n), d(\vec{x})$
3. Se calcula la salida de la red: $y(\vec{x}) = f(x_1w_1 + x_2w_2 + ... + x_nw_n + \theta)$
4. Si $y(\vec{x}) \neq d(\vec{x})$ (clasificación incorrecta) se modifican los pesos y el umbral:
   Si $\vec{x} \in A, d(x) = 1 \rightarrow w_i(t+1) = w_i(t) + x_i, \theta(t+1) = \theta(t) + 1$
   Si $\vec{x} \in B, d(x) = -1 \rightarrow w_i(t+1) = w_i(t) - x_i, \theta(t+1) = \theta(t) - 1$
5. Se vuelve al paso 2 hasta completar el conjunto de patrones de entrenamiento.
6. Se repiten los pasos 2, 3, 4 y 5 hasta alcanzar el criterio de parada.

Cada una de las capas representa una recta por eso en el problema XOR hacían falta más capas.

Tasa de aprendizaje

Es el ritmo con el que aprende, es un valor entre 0 y 1 que regula la variación de los pesos en cada iteración, para que no se vayan mucho y oscilen.

$$w_i(t+1) = w_i(t) + \gamma d(\chi)x_i$$

Se suelen empezar con valores mayores y se va reduciendo una vez se va aproximando (balance exploración-explotación)

3.5. ADALINE

El Perceptron es de naturaleza binaria, lo que restringe su dominio de aplicación a la clasificación, pero en muchos casos se dispone de datos en el dominio de los reales. El perceptron es para clasificación lineal y ADALINE para regresión lineal.

En este caso no hay que extrapolar a qué clase pertenecen las entradas desconocidas, sino que habrá que asignar a cada entrada su valor real correspondiente.

Se trata de aproximar una función F de forma que:

$$P = \{(\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)\} \quad F(\vec{x}_1) = y_1, \forall p \in P$$

No se puede utilizar aprendizaje por refuerzo, **no hay salidas correctas o incorrectas, todas son incorrectas, algunas más que otras**

En 1960, Widrow y Hoff propusieron un modelo que sí tiene en cuenta el error producido, ADAptive LInear NEuron (ADALINE)

Estructura identica al Perceptron.

La diferencia con el perceptron es la manera de utilizar la salida en la regla de aprendizaje, en el **perceptron se utiliza la salida de la funcion umbral (binaria)** para el aprendizaje (si se ha equivocado o no) y en **Adaline se utiliza directamente la salida de la red (real)** teniendo en cuenta cuánto se ha equivocado.

Se utiliza la diferencia entre el valor real esperado y la salida producida de la red ($d^p - y^p$).

3.5.1. Regla Delta

Se define el error cometido por un modelo (un conjunto de pesos) mediante la desviación entre las salidas producidas por el mismo y las que debería haber producido, para todos los ejemplos de entrenamiento:

$$E = \frac{1}{2} \sum_{p=1}^{m} (d^p - y^p)^2$$

El objetivo es encontrar un conjunto de pesos que haga que ese error sea el menor posible. Se aplica un proceso iterativo en el que después de propagar cada entrada, se modifican los pesos de acuerdo con la regla delta:

$$\Delta_p w_i = \gamma (d^p - y^p) x_i$$

El error es una función de los pesos por lo que tiene que haber un valor de los pesos para los que dicho error sea mínimo y para localizarlo se utiliza el método de descenso de gradiente.

El descenso de gradiente consiste en calcular la derivada de la función error respecto de los pesos en el punto determinado por el valor actual del peso, y utilizar dicho valor para variar los pesos

- Si el valor es grande, es que la pendiente es grande y podemos desplazarnos mucho.
- Si el valor es pequeño, nos acercamos al mínimo, y variamos poco los pesos.
- Si el valor es cero, estamos en el mínimo y no se varían los pesos.

$$\Delta_p w_i = -\gamma \frac{\partial E^p}{\partial w_i}$$

Utilizando la regla de la cadena:

$$\frac{\partial E^p}{\partial w_i} = \frac{\partial E^p}{\partial y^p}\frac{\partial y^p}{\partial w_i} = \frac{\partial(\frac{1}{2}(d^p - y^p)^2)}{\partial y^p}\frac{\partial(w_i x_i)}{\partial w_i} = -(d^p - y^p)x_i$$

De esta manera obtenemos la Regla Delta:

$$\Delta_p w_i = \gamma(d^p - y^p)x_i$$

Si aplicamos Adaline a problemas en los que las salidas son binarias la Reglas Delta quedaría:

$$\Delta_p w_i = \{\gamma x_i \quad si \quad d^p > y^p$$

La Regla Delta es una extensión de la regla del Perceptrón para valores de salida reales.

3.5.2. Procedimiento

1. Inicializar los pesos y umbral de forma aleatoria.
2. Presentar un patrón de entrada.
3. Calcular la salida, compararla con la deseada y obtener la diferencia: $(d^p - y^p)$
4. Para todos los pesos y para el umbral, calcular:
   $\Delta_p w_i = \gamma(d^p - y^p)x_i, , \Delta_p \theta = \gamma(d^p - y^p)$
5. Modificar los pesos y el umbral del siguiente modo:
   $w_j(t+1) = w_j(t) + \Delta_p w_j, , \theta(t+1) = \theta(t) + \Delta_p \theta$
6. Repetir los pasos 2, 3, 4 y 5 para todos los patrones de entrenamiento (1 ciclo).
7. Repetir los pasos 2,3,4,5 y 6 tantos ciclos hasta cumplir el criterio de parada.

4. TEMA 2: PERCEPTRÓN MULTICAPA

4.1. Problema XOR

Mediante ADALINE o un Perceptrón que son aproximadores de un solo plano no es posible crear una red que represente la lógica de la puerta XOR.

XOR		
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

Tabla 4.1: Puerta XOR

Se llega a un punto al aplicar estos métodos que no sabe mejorar más la solución, se estabilizan los pesos, pero es una solución muy alejada de la óptima.

Como se ha dicho solo se pueden resolver problemas en los que las salidas dependen linealmente de las entradas.

La manera de resolver este problema fácilmente es introducir una capa adicional, que es como emplear dos rectas para discriminar.

En el caso de más de 1 capa para ajustar los pesos no podemos hacerlo como anteriormente, ya que en las capas ocultas no conocemos la salida, solo nos serviría para la capa de salida.

4.2. No linealidad

En la mayoría de los casos los datos se distribuyen de manera no lineal, por lo que en general las rectas no son útiles para solucionar los problemas reales. Se necesitan curvas complejas.

4.3. Función de activación

Es necesario incluir una función de activación no lineal en la salida de las células.

Es importante que sean no lineales, derivables y acotadas.

Nosotros emplearemos:

- Función Sigmoide (0 - 1)

$$g(x) = \frac{1}{1 + e^{-x}} \quad g'(x) = g(x)(1 - g(x))$$

- Tangente Hiperbólica (-1 - 1)

$$g(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad g'(x) = 1 - g(x)^2$$

- Unidad de Rectificado Lineal - ReLU

$$g(x) = \text{máx}(0, x) \quad g'(x) = \begin{cases} 1 & x > 0 \\ 0 & \textit{en otro caso} \end{cases}$$

4.4. Redes alimentadas hacia delante

En cada capa C se calcula la salida de las células que la componen a_j^C con:

$$y_j^C = f\left(\sum_{i=1}^{C-1} w_{ij}^{C-1} y_i^{C-1} + \theta_j^C\right) = f(s_j^C)$$

Tal que w_{ij}^C es el peso de la capa C entre la célula origen i y la célula destino j y y_j^{C-1} es la salida de la célula i de la capa anterior. El sumatorio se hace para todas las células de la capa anterior.

4.5. Aprendizaje de pesos

En este caso no podemos hacer como en ADALINE, ya que solo se conoce la salida esperada para la última capa, pero para las intermedias no podemos. Solo podríamos ajustar los pesos de esta.

Es por esto por lo que en 1986 Rumelhart et al. presentaron una generalización de la Regla Delta que solucionaba este problema: **Regla Delta Generalizada**.

4.6. Función de error

El objetivo es que la red produzca resultados los más parecidos a los deseados, depende del tipo de problema.

En los problemas de **Clasificación** se utiliza la **entropía cruzada**: $E = -\sum_{i=1}^{C} d_i \log(y_i)$ La base del logaritmo depende del número de posibles valores.

- No siempre es conveniente ahorrar bits cuando se hace clasificación, ya que el conocimiento depende de las conexiones.

En **Regresión** se utiliza el **error cuadrático medio**: $E = \frac{1}{N}\sum_{i=1}^{N}(d_i - y_i)^2$

Se puede hacer una representación gráfica del error según los pesos que nos permite ver como se distribuyen y buscar los valles, pero esto no es posible de dibujar en los problemas por el gran tamaño de los espacios y coste. Lo que se hace es como no conocemos el error en todos los puntos se utiliza la derivada para ver la pendiente y reducirla en busca de un mínimo local (Problema del mínimo local, es difícil salir).

Como el objetivo es minimizar la función de error se calcula el gradiente de la función en un punto W para ver la pendiente y como queremos minimizar iremos en dirección opuesta al gradiente hasta alcanzar convergencia.

4.7. Retropropagación del gradiente

Hay que calcular la derivada del error respecto a cada uno de los pesos, dicha derivada se utilizará para modificar el peso correspondiente.

Se va aplicando la regla de la cadena, en las capas intermedias hay que calcular los valores hacia atrás.

La idea es desplazar el vector de pesos siguiendo la dirección negativa del gradiente del error en dicho punto, de esta manera nos aproximamos al mínimo error.

4.8. Procedimiento BackPropagation

1. Se inicializan los pesos y umbrales de la red (valores aleatorios próximos a 0)
2. Se presenta un patrón n de entrenamiento, $(x(n), s(n))$, y se propaga hacia la salida, obteniéndose la respuesta de la red $y(n)$
3. Se evalúa el error, $e(n)$, cometido por la red para el patrón n
4. Se aplica la regla delta generalizada para modificar los pesos y umbrales de la red:
   a) Se calculan los valores δ para todas las neuronas de la capa de salida
   b) Se calculan los valores δ para el resto de las neuronas de la red, empezando desde la última capa oculta y retropropagando dichos valores hacia la capa de entrada
   c) Se modifican pesos y umbrales

5. Se repiten los pasos 2, 3 y 4 para todos los patrones de entrenamiento, completando así un ciclo de aprendizaje

6. Se evalúa el error total cometido por la red (error de entrenamiento)

7. Se presentan los patrones de validación, calculando únicamente la salida de la red (no se modifican los pesos) y se evalúa el error total en el conjunto de validación

8. Se repiten los pasos 2, 3, 4, 5, 6 y 7 hasta que:

   a) El error de entrenamiento permanece estable

   b) El error de validación permanece estable

   c) El error de validación empieza a aumentar

4.9. Retropropagación del gradiente

$$s_k^P = \sum_j w_{jk}^{P-1} y_j^{P-1} + \theta_k^P \quad y_k^P = f(s_k^P)$$

Necesitamos para cada célula un valor delta (menos para la entrada). Se deriva en función de la entrada, ya que es lo que cambia el error. δ **Estimación del error en esa célula**.

$$\delta_k^P = -\frac{\partial E}{\partial s_k^P} = -\frac{\partial E}{\partial y_k^P}\frac{\partial y_k^P}{\partial s_k^P} = -\frac{\partial E}{\partial y_k^P} f'(s_k^P)$$

Para la salida (o): Siendo el error $E = \frac{1}{2}\sum(d_k^o - y_k^o)^2$

$$\delta_k^o = -\frac{\partial E}{\partial y_k^o} f'(s_k^o) = -[-(d_k^o - y_k^o)] f'(s_k^P) = (d_k^o - y_k^o) f'(s_k^o)$$

Para la oculta (h): Tras calcular la capa de salida, va hacia atrás estimando la salida deseada de las capas ocultas.

$$\delta_k^h = -f'(s_k^h)\frac{\partial E}{\partial y_k^h} = -f'(s_k^h)\sum_{l=1}^{m}\frac{\partial E}{\partial s_l^{h+1}}\frac{\partial s_l^{h+1}}{\partial y_k^h} = -f'(s_k^h)\sum_{l=1}^{m}[-\delta_l^{h+1}]\frac{\partial(\sum_j^n w_{jl} y_j^{h-1})}{\partial y_k^h} =$$

$$= f'(s_k^h)\sum_{l=1}^{m}\delta_l^{h+1} w_{kl}$$

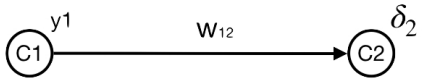
4.10. Retropropagación del error

Se calcula un valor, llamado δ, para todas las células de la red, empezando por las de la última capa y retrocediendo una capa en cada iteración.

El valor delta es proporcional a la derivada del error estimado por la célula.

- En la última capa se conoce el error producido por las células, y el delta se calcula teniendo en cuenta ese error.
- En las capas ocultas, se propaga hacia atrás el valor δ ponderado por los pesos.

Una vez calculados los deltas, todas las conexiones se actualizarán de la misma manera:



$$w_{12}^{+} = w_{12} - \gamma \delta_2 y_1$$

Fig. 4.1: Retropropagación del error

$$w_{ij}^{+} = w_{ij} - \gamma \delta_j y_i \quad b_i^{+} = b_i - \gamma \delta_j$$

4.11. Tasa de aprendizaje γ

Valor empleado en la regla delta por el que se multiplica la Δw_i.

Cuanto más grande más rápido nos aceptamos a la solución, pero cuidado porque si es muy alto oscilaremos y no nos acercaremos.

Si es muy pequeño la convergencia será muy lenta y puede ser inviable al ralentizar el proceso, estancándose en mínimos locales.

Encontrar la tasa de aprendizaje óptima no es trivial, se trata de asignar el mayor valor posible sin que se produzca oscilación.

Una manera de evitar la oscilación es hacer que el cambio de los pesos dependa de la intensidad del cambio anterior, velocidad con la que varía, una especie de inercia llamada momento. Evita que puedan diferir en exceso.

4.12. Momento

Término que recoge el cambio producido por los pesos en la iteración precedente:

$$m(t) = w(t) - w(t-1) = \Delta w(t)$$

$$w_{jk}(t+1) = w_{jk}(t) + \gamma \delta_k^p y_j^p + \alpha \Delta w_{jk}(t)$$

α es la intensidad que le queremos dar a la inercia.

Contrarresta las oscilaciones restando cuando cambia de sentido, se pueden utilizar tasas de aprendizaje relativamente altas, lo que permite una convergencia rápida. Permite navegar más eficazmente por la función de error.

4.13. Generalización

La red adapta sus pesos a partir de los ejemplos de entrenamiento hasta formar un modelo final, esta capacidad permite comportarse correctamente más allá de estos. Aunque existe un riesgo de que solo conociendo los de entrenamiento no sepa generalizar.

El conjunto de ejemplos sea suficientemente grande y característico.

El modelo resultante debe ser válido para cualquier situación, no solo para los datos de entrenamiento.

4.14. Sobre-aprendizaje

El módulo se ajusta a los datos de entrenamiento de una manera demasiado precisa. Dejando de generalizar, cosa no deseable.

Si se entrena poco, la aproximación que genera el modelo es muy burda.

Si se entrena mucho, puede adaptarse solo a los ejemplos de entrenamiento, como si lo memorizara.

Cuando el error de validación deja de disminuir, la red empieza a sobreentrenar, y debe pararse el entrenamiento

No solo depende de los ciclos de aprendizaje, también de:

- Excesivo número de pesos o neuronas ocultas, se ajusta con mucha exactitud.
- Excesiva complejidad del modelo en relación con el número de patrones de entrenamiento. Hacer las redes más sencillas.
- Escasez de patrones de entrenamiento, por lo que se adapta rápido a estos patrones y pierde la capacidad de generalización.

4.15. Conjunto de validación

Conjunto de datos diferentes a los de entrenamiento para comprobar la capacidad de generalización del modelo, no se utiliza para entrenar, solo para verificar la generalización y decidir cuándo parar. El conjunto de entrenamiento se divide en dos, uno para entrenar y otro para validar.

Se utiliza para saber cuándo empieza a sobreentrenar.

Los datos de validación deben tener una estructura similar a los de entrenamiento, para esto se aleatorizan los datos y de esta manera estarán lo menos sesgados posibles.

Si hay correlacion entre entrenamiento y validación, las dos curvas se comportaran demasiado parecido y tampoco nos servirá para ver la generalización.

4.16. Regularización

A veces se necesitan muchas neuronas para determinados problemas y para evitar sobreajuste lo que se hace es eliminar de manera aleatoria un porcentaje de las neuronas (10-25 %) en cada iteración, así no puede sobreaprenderlo.

Esto reduce la capacidad de memorizar, haciendo que tenga que aprender patrones y no casos concretos.

Se ha probado experimentalmente la eficacia de redes en las que las señales solo recorren parte de la red.

4.17. Actualización por lotes

La actualización de los pesos se puede realizar:

- Después de introducir un dato. $E = \frac{1}{2}(x - y)^2$
- Después de introducir todos o una serie de datos. $E = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$

Lo recomendado es tras una serie de datos, ya que tras cada patrón es muy costoso computacionalmente o si lo hacemos tras todo el conjunto de entrenamiento dificulta que la red pueda generalizar bien.

Hacer estas series de datos muy grandes hace que sea muy rápido computacionalmente y si son muy pequeños mejora la generalización.

Hacerlo por lotes permite una mejor estimación del gradiente, produce una convergencia más suave y utilizar tasas de aprendizaje más grandes, por lo que aprende más rápido.

Es paralelizable, lo que permite incrementar más aún la velocidad del aprendizaje.

4.18. Validación cruzada

Es un método para la evaluación de modelo que lo que permite es entrenar la red con diferentes valores de parámetros para quedarnos con el mejor.

Aunque, que se obtenga mejor evaluación, no implica que se comporte mejor ante ejemplos desconocidos, por esto que para generar modelos es necesario un conjunto de ejemplos que no hayan sido utilizados en el entrenamiento, ni para modificar los pesos y tampoco para establecer el criterio de parada.

Los modelos deben ejecutarse el suficiente número de veces con diferentes inicializaciones y asignar la media de los resultados.

Un modelo es mejor que otro si la media de los errores de test es estadísticamente significativa menor en una que en otro.

Se divide el conjunto de entrenamiento en x particiones, para hacer x modelos (x+1 en total) que se entrenaran con todas las particiones menos 1 y se hará test con la otra restante.

Para cada modelo se hace una serie de veces para evitar el sesgo del inicio aleatorio. Cuando se ha completado se hace la media del error y ese será el error del modelo que se genera si se entrena con el conjunto completo de datos y esos parámetros.

En todos los modelos una parte será para validación.

4.19. Debilidades

4.19.1. Mínimos locales

Si existen valores de la función de error con valores bajos rodeados extensamente por valores más altos, la red puede quedarse estancada y no acceder a valores mejores del error.

Soluciones:

- Añadir ruido al método de descenso del gradiente
- Partir de diferentes inicializaciones aleatorias
- Aumentar el número de neuronas ocultas

4.19.2. Saturación

Cuando la entrada total de una célula toma valores muy altos, la salida permanece constante por saturación, debido a la naturaleza de la función de activación.

El problema está en la función de activación.

Si la entrada sigue aumentando, esto no produce variaciones en la salida

La modificación de los pesos en una célula depende de la salida: $y_k(1 - y_k)$

Si la salida está cercana a 1, lo cual ocurre en la zona de saturación, los pesos dejan de modificarse y se produce estancamiento.

Soluciones:

- Normalizar datos de entrada
- Comenzar con valores de pesos cercanos a cero
- Utilizar otras funciones de activación: ReLU

  Impide la saturación de las salidas a un valor de 1 que detiene el aprendizaje

  Permite generar muchos valores cero como salida de neuronas, lo cual las desactiva y asemeja más a modelos biológicos.

5. TEMA 3: MODELOS DE REDES DE NEURONAS NO SUPERVISADOS

Tenemos un conjunto de datos y ya, sin clases o etiquetas, queremos saber que se puede decir de ellos.

La tarea más habitual es el agrupamiento o clustering en función de características comunes de los datos.

5.1. No supervisado

La red descubre por sí sola características, regularidades, correlaciones, categorías, ..., de los datos de entrada, a esto se le llama auto-organización.

Solo es relevante si los datos muestran algún tipo de redundancia $\nRightarrow$ aleatoriedad. Si tienen algún tipo de estructura.

Tareas principales:

- **Análisis de componentes principales:** Determinar qué entradas caracterizan principalmente a los datos. Si se eliminan, los datos tendrían una estructura similar.
- **Agrupamiento:** Dividir el espacio de entrada en regiones.
- **Prototipado:** Definir las regiones mediante prototipos.
- **Codificación:** La salida es una versión reducida de la entrada.
- **Extracción de características:** La red produce respuestas similares ante entradas similares.

5.2. Modelo biológico

Alta dimensionalidad 10^{11}-10^{14} células, con 1000-10000 conexiones cada una y una velocidad de 1 ms.

Aprendizaje continuo. No hay fase de aprendizaje y fase de funcionamiento. Se aprende en todo momento.

Focalización en estímulos relevantes. Aprendizaje por repetición de estímulos. Plasticidad.

5.2.1. Reglas de Hebb

Cuando el axón de una célula A está lo suficientemente cerca como para excitar a una célula B y repetidamente toma parte en la activación, ocurren procesos de crecimiento o cambios metabólicos en una o ambas células de manera que tanto la eficiencia de la célula A, como la capacidad de excitación de la célula B se ven aumentadas.

En la regla de Hebb la modificación de los pesos no obedece a ningún factor externo.

La conexión entre dos células se verá aumentada si ocurre que las dos se encuentran activas a la vez de manera persistente. Aquellas que no se repiten con frecuencia, se irán debilitando y se acabarán olvidando.

$$\Delta w_{ij} = a_i \cdot a_j \qquad \Delta w_{ij} = \mu(a_i - \overline{a_i}) \cdot (a_J - \overline{a_j})$$

$$\Delta w_{ij} = \Delta a_i \cdot \delta a_j \qquad \Delta w_{ij} = \mu a_i w_{ij} \Delta a_j$$

$$\frac{dw_{ij}}{t} = -w_{ij} + a_i \cdot a_j \qquad \frac{dw_{ij}}{t} = -w_{ij} + \sigma(a_i) \cdot \sigma(a_j)$$

Este tipo de aprendizaje se utiliza de forma generalizada en casi todos los modelos no supervisados, es la regla que se utiliza de manera normal en este tipo de procesos.

5.3. Modelo de Interacción lateral

Una de las características del cerebro es que está estructurado, este modelo trata de emular este tipo de conexiones estructuradas, de manera que hay partes que están más especializadas en una determinada tarea.

Estudios sobre el neocórtex muestran una estructura bidimensional por capas conectadas mediante conexiones laterales.

Cada célula se conecta a las más cercanas mediante conexiones excitatorias y a las más lejanas con inhibitorias. Las conexiones (-) (+) pierden intensidad con la distancia entre células. Al propagarse las señales mediante este esquema, se producen zonas más activadas y menos.

Un modelo artificial de célula (c) podría ser: $y_c^{t+1} = \sum_{i=1}^{n} x_i w_{ic} - \sum_{k=1}^{n} y_k w_{kc} + y_C^T w_{cc}$.

Primer parte las entradas normales, w_{kc} células colindantes, w_{cc} conexiones recurrentes excitatorias con si mismo. La y será $y_i = f\left(\sum_{j=-l}^{l} w_{ij} y_j\right)$

Donde los pesos w_{ij} tendrían valores siguiendo la función $f(x) = (1 - x^2)e^{-\frac{x^2}{2}}$, la función sombrero mejicano.

El eje horizontal (x) representa la distancia entre las células, el vertical (y) el valor del peso.

Si la distancia es corta, el peso es positivo y grande.

Disminuye con la distancia hasta hacerse negativo, para acabar siendo cero.

No es un modelo de red neuronal, porque no tiene aprendizaje, pero es utilizado, por su característica espacial embebida en otros modelos neuronales.

5.4. Aprendizaje competitivo - CL

Se trata de asignar, a una misma categoría, aquellos datos de entrada que compartan ciertas características.

Compuesto por dos capas, una de entrada, y una de salida con tantas células como categorías (capa competitiva) y cada célula de la capa competitiva se corresponde a una categoría.

Cuando llega un dato, lo deseable es que se active, en la capa competitiva, solo aquella célula que corresponda con la categoría a la que pertenece dicha entrada. Los datos de entrada no están etiquetados. La red debe asignar la misma categoría a datos similares.

En la capa competitiva:

- Todas las células están conectadas con las demás.
- Los valores de los pesos son fijos y siguen el esquema de interacción lateral.
- Las conexiones de las células consigo mismas toman valores positivos, iguales para todas ellas.
- Las conexiones con las demás células toman valores negativos, los mismos para todas.

Entre la capa de entrada y la competitiva.

- Las conexiones son excitatorias.
- Varían a medida que avanza el aprendizaje.

5.4.1. Funcionamiento

F1 es la capa de entrada y F2 es la competitiva o salida.

1. Las entradas se propagan de F1 a F2.
2. A cada célula de F2 llega la misma entrada, pero ponderada por sus conexiones que son diferentes.
3. Cuando la señal llega a F2, se desactivan las conexiones F1 $\rightarrow$ F2.

4. La señal es propagada de forma síncrona por F2.
5. F2 se estabiliza cuando todas las células producen salida cero, excepto una.
6. La célula que reciba más señal al principio, se queda con toda la señal y es la célula ganadora.
7. Se asignará a la entrada la categoría relacionada con la célula ganadora.
8. Cuando se introduce un nuevo dato, se restablecen las conexiones F1 → F2.

Lo ideal es que se vaya inhibiendo unas a otras de tal manera que todas excepto 1 queden muy cercanas al 0, la que toma el 1 es la que aprenderá.

5.4.2. Aprendizaje

En F2 (salida) no hay aprendizaje, los pesos entre sus células son fijos: $\forall i \quad \sum_j^N w_{ij} = 1$ y $\forall i, j \quad w_{ij} = w_{ji}$.

Los valores que van de una célula a otra en F2 tienen el mismo que la conexión contraria. Además los pesos se normalizan, de manera que todas en un sentido sumen 1.

Entre F1 y F2 aprende "el que gana se lleva todo": Para las células no ganadoras será 0, para las ganadoras será $\frac{dw_{ij}}{dt} = [-w_{ij} + \theta_i]$. $\theta_i = \frac{I_i}{\sum_{j=1}^M I_j}$

5.4.3. Características y limitaciones

Se trata de un aprendizaje Hebbiano. El aprendizaje es proporcional a las salidas producidas por las células que une.

Se refuerzan más las conexiones por las que pasa más señal (las que tienen un mayor I_i).

No produce codificación estable ante entradas arbitrarias. La introducción de entradas aleatorias o con ruido, produce salidas arbitrarias, y puede afectar a lo ya aprendido.

Capacidad limitada de codificación (con muchas entradas y pocas categorías se puede colapsar y dejar de aprender).

Se necesita establecer a priori el número de categorías. Lo normal es tener un número intuitivo de salida e ir probando, se evaluará con la dispersión de clústeres, se busca tener pocos clústeres y bajo dispersión.

5.5. Mapas auto-organizativos de Kohonen - KSOM

El modelo de aprendizaje competitivo se puede extender para realizar tareas de agrupamiento no supervisado.

En ese caso cada célula de la CC se corresponde con el centroide de un agrupamiento.

Un punto del espacio de entrada pertenece al agrupamiento representado por el centroide más cercano a él.

De esta manera se divide el espacio en regiones de Voronoi.

5.5.1. Modelo

Cada célula de la capa competitiva (CC) está conectada con todas las entradas. Si la entrada tiene dimensión "n", cada célula de la CC tendrá "n" conexiones.

Se puede representar cada célula de la CC como un punto en un espacio n-dimensional.

La señal se propaga desde la entrada de manera no habitual.

Para cada célula de la CC, se calcula la distancia entre sus pesos, y los valores de la entrada. Se calcula la distancia desde cada entrada hasta cada prototipo.

La activación de la célula coincide con dicho valor de distancia.

Se puede elegir entre varias medidas de distancia:

- Producto escalar: $\tau_j = d(e, w_j) = \sum_i e_i w_{ij}$. e es la entrada, w es el peso de la entrada-célula, y $d(x, y)$ es la distancia.
- La más utilizada es la euclídea: $\tau_j = \sqrt{\sum_i (e_i - w_{ij})^2}$.

5.5.2. Funcionamiento

1. Se recibe el vector de entrada y se propaga por las conexiones hasta llegar a la capa de competición mediante alguna de las fórmulas anteriores.
2. Cada neurona de la CC produce una salida al comparar la entrada con sus pesos, la competición sigue el modelo de interacción lateral. Calcula las distancias.
3. Se selecciona aquella que produzca una salida más pequeña (célula ganadora, la más cercana).

Cuando ya tienes las distancias de CC y entrada se cortan las conexiones con la entrada y se deja funcionar la interacción lateral hasta que quede un 1 y el resto 0. Aunque lo normal es coger la que tiene menor distancia y no hacer interacción lateral.

5.5.3. Aprendizaje

El aprendizaje se produce en los pesos entre la entrada y la competición, siguiendo un esquema Hebbiano.

Emplea la regla "el que gana se lleva todo": 0 para las no ganadoras y para la ganadora $\frac{dw_{ij}}{dt} = \alpha(t)\tau_j(e_i(t) - w_{ij}(t))$.

La tasa de aprendizaje varía en función del tiempo, va decrementando con el tiempo una cantidad constante. Se utiliza para que los valores más antiguos tengan mayor intensidad de aprendizaje. $\alpha(t+1) = \alpha(t) - \beta$.

Mediante el valor de β se determina el número de ciclos totales de aprendizaje, $Iteraciones = \frac{\alpha(0)}{\beta}$, que serán las veces que se puede decrementar el valor hasta llegar a 0.

Los pasos son:

1. Se introduce un patrón de entrenamiento.
2. Se calcula su distancia a todos los prototipos de la CC.
3. Se selecciona el de menor distancia C_j (célula ganadora).
4. Las conexiones de las células ganadoras se modifican, acercando sus coordenadas a las de la entrada: $w_{ij} = \alpha(e_i - w_{ij})$.

5.5.4. Vecindario

Al final del aprendizaje cada célula se situará en el centro geográfico de los patrones a los que representa.

Cada célula es independiente y no aporta información sobre las demás, por lo que sería interesante incluir información de cómo se relacionan las células entre sí (como se relacionan las categorías entre sí).

En SOM esto se puede hacer introduciendo el concepto de vecindario, cada célula tiene un conjunto de vecinos, que influirán en su aprendizaje. Se forma una rejilla que puede ser unidimensional o bidimensional.

La distancia entre dos células es el número de células por las que hay que pasar para llegar de una a otra y la distancia de una célula consigo misma es 1.

Este añadido se utiliza en el aprendizaje, cuando una célula gana, también se modificarán las vecinas según la distancia. La regla de aprendizaje pasa a ser:

$$\frac{dw_{ij}}{dt} = \frac{\alpha(t)}{d(c_i, c_j)}(e_i(t) - w_{ij}(t))$$

De manera que c_i es la ganadora y $d(c_i, c_j) < \theta$ es la distancia entre la ganadora y la vecina j, que debe estar a menos de una distancia umbral, para el resto de las células será 0.

En este caso el aprendizaje se produce en la célula ganadora y en sus vecinas, la intensidad del aprendizaje decrece con la distancia a la célula ganadora hasta que más allá de un determinado umbral θ no se produce aprendizaje.

Para cada patrón de entrenamiento se elige una célula que se desplazará en dirección a dicho patrón y en su movimiento arrastra a las que están conectadas con ella, hasta un determinado rango. Esto hace que células próximas en la topología de la red representen agrupamientos similares.

Al final, células cercanas en el vecindario ocuparán posiciones cercanas en el espacio y esto permite no solo dividir las entradas en grupos, sino relacionar los grupos entre sí.

5.5.5. Procedimiento

1. Inicialización pesos a valores aleatorios pequeños.
2. Presentar una nueva entrada. Actualizar α
3. Propagar la entrada hasta la capa de competición donde se selecciona una célula ganadora.
4. Actualizar los pesos de la célula ganadora y sus vecinas.
5. Si aún quedan entradas, ir a 2.
6. Actualizar α
7. Ordenar el conjunto de aprendizaje.
8. Si α está por encima de un cierto umbral, ir al 2.

5.5.6. Ejemplo de problema del viajante - TSP

El método que se utiliza es el de Kohonen de la goma elástica, dado que se va estirando y adaptando al dibujo de las ciudades, según se estira se aumentan las células.

El vecindario aporta información adicional sobre la estructura de los agrupamientos, en este caso el vecindario es fundamental, es lo que realmente nos interesa no la ubicación de las células.

Se realizan las siguientes modificaciones:

- Se crean un vecindario unidimensional circular en forma de anillo, como si fuera una goma elástica, tendrá solo dos conexiones con otras células, una en el lado derecho y otra en el lado izquierdo.
- Se comenzará con pocas células, como son 3, buscando que se vayan moldeando y acercando los puntos. Tras realizar un ciclo/iteración completa, haber pasado por las entradas las ciudades que componen el problema, en este momento se dividen las células en otras dos.

- En este caso cada célula tiene varias ciudades asignadas, por esto es por lo que se particiona, dado el caso de que no tenga asignada ninguna ciudad esta célula se elimina.

El punto final es que tengamos tantas células como ciudades, de manera que cada una corresponda con una ciudad y las conexiones con las otras dos el camino que se debe recorrer, es decir el vecindario es la solución.

5.6. K-medias

Fue propuesto por J. MacQueen en 1967.

Algoritmo de agrupamiento no supervisado, el espacio de patrones de entrada se divide en K clústeres o regiones. Es necesario establecer el número de clústeres (K).

Dado un conjunto de patrones: $X(n) = (x_1(n), x_2(n), ...x_p(n))\ n = 1, ...N$ se pretende encontrar K centros $C_i = (c_{i1}, c_{i2}, ...c_{ip})\ i = 1, ..., K$ con el objetivo de minimizar las distancias euclídeas entre los patrones de entrada y el centro más cercano.

$$D = \sum_{i=1}^{k} \sum_{n=1}^{N} Min||X(n) - C_i||$$

- N es el número de patrones.
- || || representa la distancia euclídea
- X(n) el patrón de entrada n
- Min es la función de pertenencia tal que:

$$Min = \begin{cases} 1 & si\ ||X(n) - C_i|| < ||X(n) - C_s||\ \forall \neq i \\ 0 & en\ caso\ contrario \end{cases}$$

Se trata de encontrar los K puntos de la región que se genera con todas las entradas, que mejor representa a todos los datos, cada uno de los puntos generará un clúster que estará compuesto por aquellos puntos que se encuentran más cerca de ese clúster que de ningún otro.

Los puntos C se generarán inicialmente aleatoriamente, y según que instancias estén más cerca de este que de otro, se calculara la media de sus entradas de manera que se centre para todas estas, esto repetido para todos los puntos y múltiples veces hace que se vayan centrando poco a poco y distribuyendo.

5.6.1. Procedimiento

1. Se inicializan aleatoriamente los centros de los K clústeres.
2. Se asignan N_i patrones de entrada a cada clúster C_i. El patrón X(n) pertenece al clúster C_i si: $\|X(n) - C_i\| < \|X(n) - C_s\|\ \forall s \neq i$.
3. Se calcula la nueva posición de los centroides como la media de todos los patrones que pertenecen al clúster, es decir: $c_{ij} = \frac{1}{N_i} \sum_{n=1}^{N} Min\ x_j(n)$
4. Se repiten los pasos 2 y 3 hasta que las nuevas posiciones de los centroides no se modifican respecto a su posición anterior: $\|C_i^{nuevo} - C_i^{anterior}\| < \epsilon, , \forall i$

5.6.2. Resumen

Clasificación por vecindario no supervisada. Se parte de un número determinado de prototipos, y de un conjunto de ejemplos sin etiquetar.

Los prototipos se sitúan en el espacio de forma que recojan características similares. Una vez situados, al llegar un nuevo dato, solo se compara con los prototipos, y se clasifica con aquel que sea el más próximo.

Tiene una fase de entrenamiento, que puede ser lenta, pero la clasificación de nuevos datos es muy rápida.

Se calcula para cada prototipo el ejemplo más próximo y se les relaciona.

Para cada prototipo A_k se crea un conjunto con los ejemplos que contiene y se desplaza el prototipo hacia el centro de masas de su conjunto de ejemplos, se repite el procedimiento hasta que ya no se desplazan los prototipos.

5.7. Learning Vector Quantization - LVQ

Cuando Kohonen creo su modelo vio que podía ser aplicable en casos de aprendizaje supervisado, como clasificación, no solo para clustering (no supervisado). Emplearlo conociendo las salidas nos permite dividir las regiones de manera más fácil.

Se combina el aprendizaje competitivo con el supervisado. El modelo de red neuronal es idéntico, cambia la regla de aprendizaje.

Se mantiene la tarea de dividir el espacio en regiones, pero estas regiones representarán las etiquetas de la clase de manera que dependiendo de la región en la que caiga una nueva instancia pertenecerá a la clase correspondiente.

Según las regiones de Voronoi que se generen.

Lo que se busca es reducir el número de errores de clasificación cuando se introducen los patrones de entrada.

En la Capa Competitiva al menos tantas células como categorías, que es conocido previamente al estar etiquetados los datos.

La misma categoría puede estar dispersa en el espacio por lo que hay que incluir más categorías en la red que las que hay en los datos.

Las células estarán etiquetadas e inicialmente se elige el número de células que tendrá el modelo y cuáles son sus etiquetas (no solo hay una neurona por etiqueta), debe haber al menos más del doble que el número de etiquetas.

Las entradas son del tipo: $E_i = [(e_1^i, C), (e_2^i, C), ..., (e_n^i, C),], , i = 1...m$

- m es el número de ejemplos del conjunto de entrenamiento.
- n la dimensión de la entrada.
- C la categoría a la que pertenece cada dato.

5.7.1. Reglas de aprendizaje

1. Se inicializan los pesos a valores pequeños aleatorios.
2. Para cada uno de los patrones de entrada, se propaga por la red y se obtiene la célula ganadora.
3. Si la célula ganadora pertenece a la misma categoría que la entrada.
4. Se modifican los pesos de la célula ganadora siguiendo las ecuaciones de KSOM: $\Delta w_{ij} = \alpha(e_i - w_{ij})$
5. Si la célula ganadora no pertenece a la misma categoría que la entrada.
6. Se modifican los pesos de forma inversa a los de KSOM: $\Delta w_{ij} = -\alpha(e_i - w_{ij})$
7. Repetir los pasos 2-7 hasta que se cumpla el criterio de parada.

Si la red produce una respuesta correcta, acercamos el prototipo al dato de entrada.

Si la red produce una respuesta incorrecta, alejamos el prototipo al dato de entrada.

Al final del proceso los prototipos se colocarán en el centro de masas de puntos etiquetados con su misma categoría.

5.8. LVQ2

Se calculan dos células ganadoras, las dos más próximas.

Si las dos pertenecen a la misma clase, solo se aprende la más cercana, siguiendo las reglas del LVQ:

- $\Delta w_{ij} = \alpha(e_i - w_{ij})$ Si misma clase que ejemplo
- $\Delta w_{ij} = -\alpha(e_i - w_{ij})$ Si distinta clase que ejemplo

Si cada una pertenece a una clase, la de la misma clase de la entrada se acerca, la otra se aleja.

Variante LVQ2.1: Solo se aprenden los dos prototipos más cercanos si:

- Pertenecen a clases diferentes.
- Están dentro de una ventana (θ) del plano medio de los dos vectores.

$$\text{mín}\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > \frac{1-\theta}{1+\theta}$$

Se divide el más grande sobre el más pequeño.

6. TEMA 4: REDES NEURONALES CONVOLUCIONALES - DEEP LEARNING

Surgieron con el aumento de la capacidad de cómputo.

Se emplea en problemas con muchas entradas (como son las imágenes o señales), se llama profundo porque tiene muchas neuronas.

Con el tiempo se han ido mejorando y actualmente los resultados son impresionantes.

6.1. Autoencoders

6.1.1. Redundancias

Los autoencoders se basan en la redundancia para poder comprimir y reducir la entrada a un menor número de células, dado que si fueran completamente aleatorio no se podrían sacar patrones.

Compresión: Para todo vector $\vec{x} = \{x_1, x_2, \cdot, x_m\}$, obtener otro $\vec{z} = \{z_1, z_2, \cdot, z_n\}$ con $n \ll m$, de forma que $\vec{x}$ pueda ser construido a partir de $\vec{z}$.

6.1.2. Formulación

La función de compresión y descompresión tendrá la forma: $\vec{z} = W_1\vec{x} + b_1$, $\overline{x} = W_2\vec{z} + b_2$.

Esto se puede representar mediante una red de neuronas en la que la entrada es $\vec{x}$, los pesos de compresión W_1, las células ocultas $\vec{z}$ capaces de recomponer la información tras pasar su salida por los W_2.

Se puede reconstruir la entrada, $\vec{z}$, con la salida, $\overline{x}$ al ser una aproximación.

6.1.3. Arquitectura

Se trata de una arquitectura de red neuronal con dos capas ocultas.

- Los pesos de la primera capa oculta se utilizan para la compresión.
- Los pesos de la segunda capa oculta se utilizan para la descompresión.

Se trata de una red no supervisada, no se necesita información adicional sobre los datos que guíe el aprendizaje (no hay etiquetas asociadas para clasificar, predecir o ajustar). Sin embargo, se utiliza un proceso supervisado.

Para ello se asigna como salida de cada dato $\vec{x}^{(i)}$ el propio dato, se aprende a obtener como salida la propia entrada $f(\vec{x}^{(i)}) = \vec{x}^{(i)}$.

El vector $\vec{z}^{(i)}$ es la entrada comprimida, porque a partir de dicho vector, y de los pesos W_2 que son siempre los mismos, se puede recomponer la entrada original $\vec{x}^{(i)}$.

6.1.4. Propiedades

Función identidad: La función identidad parece trivial, pero introduciendo restricciones en la red se pueden descubrir propiedades interesantes de los datos.

Si las neuronas ocultas son muchas menos que las de entrada → Compresión.

Si las entradas son aleatorias, no se pueden comprimir.

Si las entradas tienen estructura la red es capaz de descubrir determinadas correlaciones entre las entradas.

Incluso si las células ocultas son más numerosas se pueden descubrir estructuras en los datos imponiendo otras restricciones (sparsity).

6.1.5. Sparsity

Restricción de carencia: Consiste en forzar a la red a que, para cada patrón de entrada, solo se activen un número reducido de células, el resto deben producir salida cero.

Si calculamos la media de las activaciones (salida con pesos y todo) de las células de la capa oculta (según las entradas): $\overline{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} a_j(x^{(i)})$

- m entradas.
- $a_j(x^{(i)})$ activación célula j con entrada $x^{(i)}$.

Dicha media debe mantenerse en un valor bajo $\overline{\rho}_j \leq \rho$, donde ρ es el valor de carencia (sparsity) habitualmente cercano a 0.

Para conseguirlo es necesario que la mayoría de los valores de activación de las células que no se activan para un patrón determinado, estén cercanos a 0.

6.1.6. Divergencia KL

Para conseguirlo se introduce un término de penalización en la función de error, dicho término será proporcional a cuanto se desvíe $\overline{\rho}$ de ρ: $KL = \sum_{j=1}^{s} \rho \log \frac{\rho}{\overline{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\overline{\rho}_j}$

- s es el número de neuronas de la capa oculta.

Se trata del término de divergencia de Kullback-Leibler que mide la divergencia entre una variable aleatoria de Bernoulli de media ρ con otra de media $\overline{\rho}$.

- $\overline{\rho}$: media de las células.
- ρ: carencia o sparcity.

Dicha función de penalización tiene la propiedad: $KL(\rho||\overline{\rho}_j) = 0 \; si \; \overline{\rho}_j = \rho$. De manera que el mínimo está en ρ y aumenta hasta el infinito a medida que se aleja de dicho valor, tanto por exceso como por defecto.

6.1.7. Error sparse

Al incorporar la divergencia KL en la función de error, quedaría:

$$E_{sparse}(W, b) = E(W, b) + \beta \sum_{j=1}^{s} KL(\rho||\overline{\rho}_j)$$

Al calcular los deltas sin sparse:

$$\delta_i^{(2)} = \left(\sum_{j=1}^{s} w_{ji}^{(2)} \delta_j^{(2)} \right) f'(z_i^{(2)})$$

Si incorporamos el término KL:

$$\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s} w_{ji}^{(2)} \delta_j^{(2)} \right) + \beta \sum_{j=1}^{s} KL(\rho||\overline{\rho}_j) \right) f'(z_i^{(2)})$$

Hace que las células no se saturen tanto.

Se propagan todos los ejemplos de entrenamiento para unos valores fijos de los pesos, y se calculan los $\overline{\rho}_i$ para cada célula de la capa oculta.

Calcular el término de divergencia KL para cada célula de la capa oculta.

Incluir dicho término en el cálculo de los deltas para cada célula de la capa oculta.

Al incorporar el término de convergencia KL en el error, en cada iteración no solo se tratará de minimizar el error cometido por la red, sino que se minimiza también la diferencia entre el valor medio de activación de las células y el prefijado (ρ), de forma que cuando una célula se activa fuertemente, las demás conservan su activación en valores muy bajos, y esa misma célula se mantendrá también en valores bajos para patrones que activen otras células.

6.1.8. Visualización

En las redes sparse, las células se activan principalmente para ciertos patrones de entrada, y se mantienen inactivas para el resto.

Se puede, por lo tanto, visualizar qué patrones de entrada producen la activación de cada una de las células, esta visualización proporciona información acerca de qué está ocurriendo en la red, para ello hay que preguntarse qué patrón de entrada produce que el valor de activación de una célula a_i sea máximo.

6.1.9. Estancamiento

En redes neuronales con muchas capas (profundas):

- Cuando las RN tienen muchas capas, las magnitudes de los gradientes en las últimas capas y en las primeras son muy diferentes.
- La forma de la función objetivo hace difícil, para el método del descenso del gradiente, recorrerla en dirección a buenos mínimos (aunque sean locales).
- Las redes grandes tienen muchos parámetros que pueden ser memorizados y que dificultan la capacidad de generalización.

A partir de 2006 se observó que los autoencoders podían ser utilizados para pre-entrenar una RN y dicho proceso era capaz de atenuar los problemas anteriores, para ello se inicializan los pesos de forma sucesiva en cada capa utilizando autoencoders.

6.1.10. Inicialización

Entrenar cada capa oculta por separado y en orden, utilizando datos no supervisados. Para esto se va cogiendo la red por capas, cada vez 2, de manera que la primera capa representa la entrada y la siguiente la de compresión.

Entrenar la última capa mediante datos supervisados.

Se utiliza Backpropagation para ajustar toda la red mediante datos supervisados.

Procedimiento de inicialización

1. Se entrena un primer autoencoder con las entradas ($\vec{x}^{(i)}$, $\vec{x}^{(i)}$), obteniéndose W_1, W_1'.
2. Se utilizan los pesos W_1 para obtener valores de entrada de la primera capa oculta ($\vec{z}^{(i)}$), para todos los datos de entrada.
3. Estos datos se utilizan para entrenar un segundo autoencoder formado únicamente con la segunda capa oculta, y las entradas de la capa oculta, obteniéndose los pesos W_2, W_2'.
4. El proceso se repite tantas veces como capas ocultas hay en la red.
5. La última capa puede pre-entrenarse utilizando datos supervisados, en vez de un autoencoder.

6.2. Redes profundas

6.2.1. Conectividad

En general, cada neurona de la capa oculta está conectada a todas las entradas.

Si la entrada es altamente multidimensional, es inabordable, como sería una imagen con todos sus pixeles.

Para evitarlo se puede conectar cada neurona oculta solo con un subconjunto de las entradas. Es importante que la conectividad siga un patrón, como conectar pixeles adyacentes.

6.2.2. Compartición de pesos

Se puede extender este esquema a todas las capas ocultas, no solo la primera, y obtener una red profunda con conexiones locales.

El procedimiento de aprendizaje es equivalente, suponiendo que las conexiones inexistentes tienen un valor de cero, esto produce una significante reducción de la cantidad de conexiones.

Se puede reducir aún más mediante el mecanismo de compartición de pesos, los pesos serán los mismos cuando se va repitiendo el patrón sobre las entradas.

El modelo es muy compacto y sigue esquemas que se dan en el cerebro.

La idea de compartición de parámetros viene del procesado de imágenes, y se conoce como convolución. Es como pasar un filtro, los pesos compartidos, a subconjuntos de datos de las señales.

6.2.3. Pooling

Cada capa convolucional sigue este esquema, con lo que capas convolucionales sucesivas también comparten pesos.

Es habitual añadir, para cada capa convolucional, una capa adicional llamada "max-pooling". En esta nueva capa, las neuronas calculan simplemente el valor máximo (o cualquier otro estadístico) de las células a las que están conectadas, y producen dicho valor como salida.

Una propiedad interesante de esta aproximación es que la salida de las células del "max-pooling" son invariantes a los desplazamientos en las entradas.

6.2.4. Convolución

A este tipo de redes se les llama redes convolucionales.

A la capa de pooling se la denomina "subsampling layer", ya que reduce significativamente el tamaño de la entrada.

En la capa de pooling, el máximo puede ser sustituido por algún otro estadístico.

Se puede hacer Local Contrast Normalization ($\frac{maximo-media}{varianza}$), lo cual mantiene invarianza al brillo, muy útil en tratamiento de imágenes.

La salida de una capa convolucional puede ser la entrada de una nueva capa convolucional, y así sucesivamente, construir una sucesión de capas convolucionales en cascada que dota a la red de mayor profundidad.

Cada capa convolucional se puede asemejar a un filtro o a un preprocesado, indispensable para tratamiento de señales.

Esto permite tratar con la señal "en crudo", ya que las capas convolucionales realizan las transformaciones de los datos que antes había que hacer a mano, mediante prueba y error.

Para que el proceso se realice automáticamente hay que modificar el procedimiento de BackPropagation para que pueda incorporar las capas convolucionales:

- Calcular el gradiente, $\frac{\partial J}{\partial w_1}, \cdot, \frac{\partial J}{\partial w_n}$ para todos los pesos.
- Utilizar la media del gradiente para los pesos compartidos.

$$w_1 = w_1 - \alpha(\frac{\partial J}{\partial w_1} + \frac{\partial J}{\partial w_4} + \frac{\partial J}{\partial w_7})$$
$$w_4 = w_4 - \alpha(\frac{\partial J}{\partial w_1} + \frac{\partial J}{\partial w_4} + \frac{\partial J}{\partial w_7})$$
$$w_7 = w_7 - \alpha(\frac{\partial J}{\partial w_1} + \frac{\partial J}{\partial w_4} + \frac{\partial J}{\partial w_7})$$

- La capa de maxpooling calcula el valor máximo, por lo que hay que calcular solo el gradiente del máximo.

6.2.5. Múltiples canales

Las señales pueden tener varios canales independientes, p. ej. imágenes 3 canales.

Se puede replicar el modelo anterior para que cada copia trabaje con un canal de entrada.

Los pesos entre canales no se comparten.

6.2.6. Múltiples mapas

A cada entrada se le aplica un solo filtro.

Se puede extender el esquema para aplicar más de un filtro a cada entrada.

A cada conjunto de salidas producidas por cada filtro se le denomina "mapa".

En la figura tenemos dos filtros → dos mapas.

Para propagar la salida, se puede tratar a cada mapa como si fuera un canal diferente, como en el caso anterior.

6.3. Procesado de imágenes

El ordenador trata a las **imágenes como una colección de valores numéricos**.

Se utiliza para **tareas muy variadas y complejas**, como conducción automática, imágenes médicas, segmentación. Hay que hacer clasificación.

Una de las principales tareas es la **clasificación de imágenes**, por lo que hay que ser capaces de **detectar características** relevantes.

La tarea **se complica debido al número de factores** que intervienen: Variación del punto de vista, condiciones de iluminación, variación de escala, deformación, camuflaje con el entorno, variación intraclase (misma clase pero muy distintas), ...

Jerarquía de características, habría que **detectar características a distintos niveles**, y combinarlas entre sí.

Para **detectar características se utilizan filtros**, que recorren la estructura espacial de la imagen. Diferentes filtros detectan diferentes características.

6.4. Redes convolucionales

Cada **filtro, conjunto de pesos**, se aplica sobre una parte de la imagen, y se va desplazando. Esto se denomina convolución.

El **filtro se va desplazando por la imagen hasta cubrirla** completamente. Los pesos del filtro no varían al desplazarse a través de la imagen → pesos compartidos.

El filtro **se puede desplazar una unidad, o varias "stride"**. En cuyo caso se reduce el tamaño de la salida.

Cada filtro puede ser capaz de realizar una operación sobre las imágenes, detectar bordes, suavizar la imagen, etc. En conjunto serán un preprocesado de la imagen.

Si se quiere **mantener el tamaño de la imagen después de la convolución**, habría que realizar una operación llamada **"padding"**. Se rellena con 0's la imagen, puede ser otro valor.

Después de la convolución se realiza algún tipo de **"pooling". Esto reduce el tamaño de la entrada**

Si hay **múltiples canales, se aplica el filtro a cada uno** de ellos y **se combinan** todos para crear una salida única.

Si hay **múltiples canales y múltiples filtros**, se obtiene **una salida para cada filtro en la que se combinan todos sus canales.**

Una **capa convolutiva** está compuesta por los **filtros** (con o sin padding y combinando los canales si hay más de uno), una **capa de pooling** y una **función ReLu**. La salida suele ser de una menor dimensión

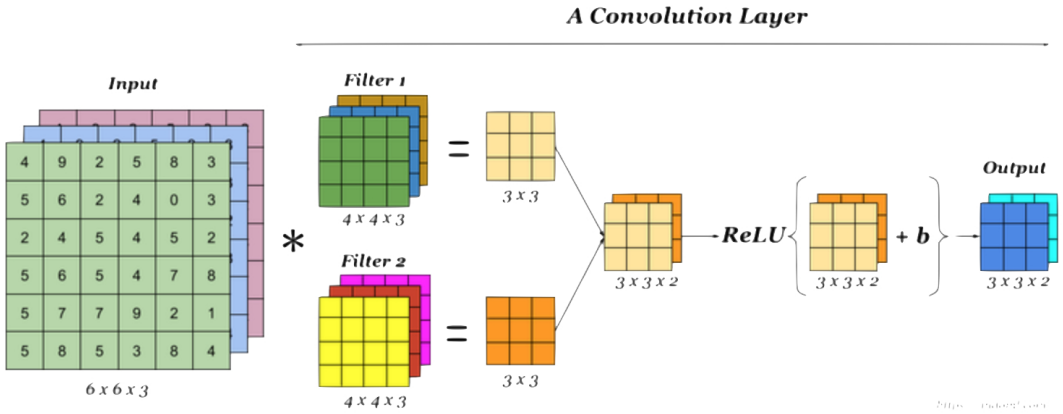


Fig. 6.1: Capa convolutiva

Las células utilizan la **función de activación ReLu**, aplicada a la entrada para conseguir la **no linealidad** del modelo.

Una **red convolutiva** está compuesta por **varias capas convolutivas** de manera que la **salida de cada una es la entrada de la siguiente**. Al **final** se incluye una **capa "full conected"** que puede ser un Perceptrón Multicapa, una red recurrente, etc.

6.4.1. Resumen

Una red convolucional está formada por una sucesión de capas de parejas de filtros y "pooling".

Para cada capa hay que especificar el número de filtros, su tamaño (f) y desplazamiento (s) (todos iguales) y el tamaño del pooling (p).

Si N es la dimensión de entrada a un filtro, la dimensión de la salida de cada filtro será:

$$M = \frac{\frac{N-f+s}{s}}{p}$$

Funciona la fórmula si el pooling tiene stride de 1, si no hacer la fórmula para los filtros y después para el pooling.

6.5. Aprendizaje en CNN

La capa convolucional aplica filtros específicos a la imagen (kernel o pesos) para identificar diferentes partes y propiedades de la imagen La función no lineal (ReLU) permite el procesado no lineal de los filtros.

El pooling reduce el tamaño de la imagen y ayuda a identificar qué características, de las obtenidas mediante los filtros, son importantes.

La tarea de clasificación se realiza en la parte final de la red que es una arquitectura "full-conected" clásica.

Se trata de una BackPropagation en el que se ha sustituido la parte de tratamiento de la señal por un procedimiento automática capaz de descubrir los filtros apropiados en cada caso (especificado por las etiquetas de los datos).

El proceso de entrenamiento está compuesto por cuatro pasos básicos:

1. Propagación hacia delante de las imágenes. Se introduce una entrada, se pasa por las capas de convolución usando la función ReLu y pooling, finalmente se pasa por la capa full conected.
2. Cálculo del error.
3. Retropropagación del error.
4. Optimización mediante la modificación de los pesos, tanto de la parte de clasificación, como de las capas convolucionales (aprendizaje de los filtros).

Cada capa aprende un filtro cada vez más complejo (por capas).

- Primeras capas. filtros de detección de características básicas: esquinas, bordes, verticalidades, etc.
- Capas intermedias. filtros para detectar partes de objetos: ojos, nariz, etc.
- Capas superiores. reconocen objetos completos de diferentes tamaños y en diferentes posiciones.

6.5.1. Cálculo y retropropagación del error

Se calcula el error cometido por la red para un patrón.

En la capa "full-conected" se modifican los pesos de la forma habitual.

Si la parte de clasificación tiene más de una capa, se retro-propaga el error siguiendo el esquema de un MLP.

En las capas convolucionales, se calcula un delta para todas las células siguiendo el MLP, con ciertas consideraciones:

- En la capa "maxpool" solo se calcula el delta solo para la célula de mayor activación (la que ha producido una salida). Para el resto su delta es cero, y sus pesos asociados no varían.
- Los pesos que varían no son los del "maxpool", que no existen, sino los de su célula asociada en la capa convolucional.
- Si se utiliza "averagepool", se ignora la capa de pooling en el proceso de aprendizaje.

Hay que tener en cuenta que cuando se modifican los pesos, debido al carácter de las capas convolucionales como filtros, las células comparten pesos (los filtros son los mismos para diferentes zonas de la capa convolucional.

6.5.2. Dropout

Dropout se refiere al resultado de ignorar ciertas células durante la fase de aprendizaje.

Cuando se calcula el delta de una célula, para un conjunto de ellas, elegido aleatoriamente, su valor se anula, de forma que los pesos asociados a dicha célula no se varían.

En cada ciclo, un nuevo conjunto de células es seleccionado para realizar dropout, con una probabilidad p, que será un parámetro del método.

Durante el entrenamiento, solo una cantidad pequeña (1-p) de las células son seleccionadas para el aprendizaje. Las células eliminadas son diferentes en cada ciclo, hay 2^n combinaciones (modelos) diferentes de dropout.

La red resultante debería ser la media de los distintos modelos, pero no se tienen "físicamente" los 2^n modelos.

Una buena aproximación consistiría en utilizar una versión reducida del modelo resultante en periodo de test, de manera que se **utiliza la totalidad de los pesos, pero reducidos en exactamente una cantidad "p"** ($p \cdot w$).

En redes complejas existe un gran número de parámetros (pesos) necesarios para poder identificar problemas complejos.

La existencia de tantos pesos permite que la red se adapte muy eficazmente a los datos de ejemplo. Si se reduce el tamaño de la red, se evita este problema, pero con una parecida reducción en la eficacia de la red.

La solución de eliminar parámetros en el aprendizaje evita el sobreentrenamiento. Al utilizar todos los parámetros en test conseguimos redes complejas capaces de adaptarse mejor a datos complejos.