Redes Neuronales

Aprendizaje Automático

Víctor de la Cueva
vcueva@itesm.mx

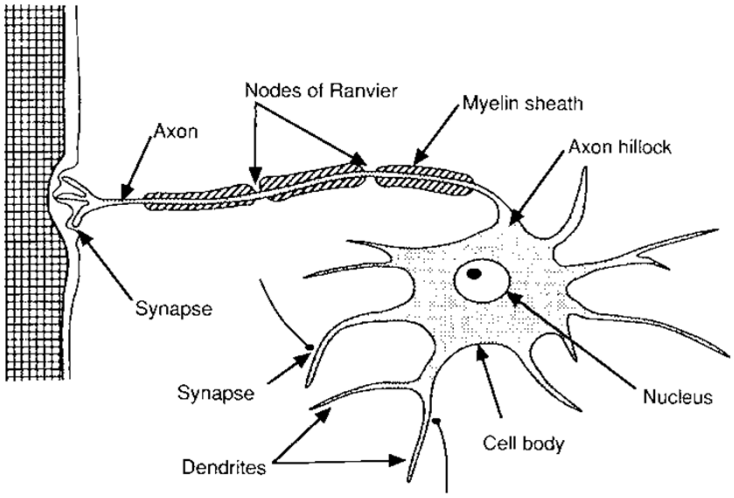
Supervisado

REDES NEURONALES

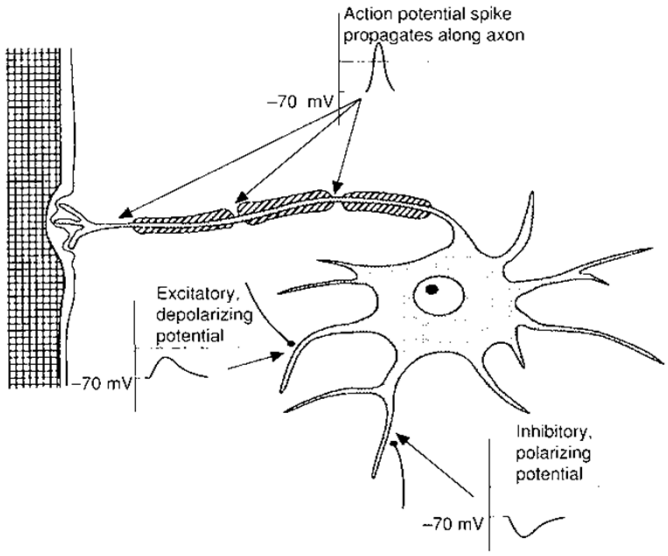
Neurocomputación

- Es una rama de las CS en donde convergen 2 ramas principales, las ciencias cognitivas y la ingeniería.
- Es un intento por hacer dispositivos que estén diseñados en forma similar al cerebro y que, tal vez, puedan hacer algunas cosas que éste hace.
- La herramienta fundamental en la neurocomputación son las redes neuronales artificiales o simplemente NN, las cuales tratan de imitar el comportamiento de las neuronas (de ahí su nombre).

Neurona Biológica



Funcionamiento

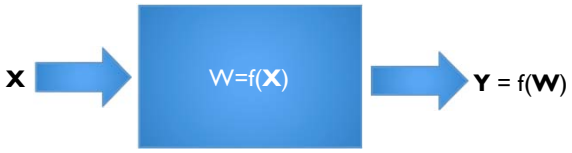


Redes Neuronales

- Las redes neuronales son estructuras formadas por elementos de procesamiento y la conexión y comunicación entre ellos.
- Un elemento de procesamiento (PE) es un burdo intento de imitar una neurona. También se les conoce como Neuronas Artificiales.
- Un PE recibe una serie de entradas, las procesa y nos entrega una salida.

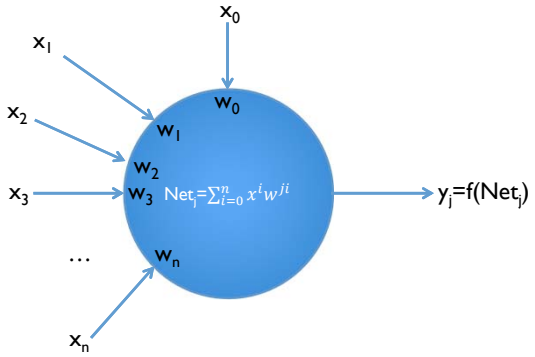
PE y NN

- En general, cualquiera de los dos se puede ver como una caja negra que recibe entradas, las procesa y entrega salidas.



PE U_j

- La unidad básica de dicho procesamiento es el trabajo de un PE.
- Un PE tiene el siguiente funcionamiento:



Funcionamiento de un PE

- La idea es sumamente simple.
- Un PE recibe entradas x que son números reales.
- Cada entrada es multiplicada por un peso, que también es un número real, el cual define la "importancia" de la entrada.
- El proceso interno del PE termina al sumar todos estos productos para encontrar Net.
- La salida y se obtiene aplicando a Net una función especial llamada Función de Activación.
- Se pueden implementar en *hardware* en forma relativamente simple.

Los pesos w_i

- En un PE están dadas:
  - Las entradas
  - La función de activación
- Por lo tanto, las únicas variables que quedan son los pesos w_i, por lo que dichos pesos son los parámetros del modelo que se deben determinar.
- El problema de una NN es encontrar los pesos que minimizan el error de la salida obtenida con respecto a la salida deseada.

PE de entrada

- Hay un tipo especial de PE que se utiliza como entrada (para propagar la entrada).
- Este PE no tiene procesamiento interno.
- Simplemente recibe una sola entrada y la manda como salida.
- Se utiliza para poder generalizar las estructuras de datos utilizadas para programar las NN.

Notación

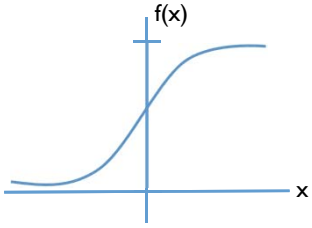
- La notación de las entradas y salidas va cambiando conforme se hacen las NN.
- Por ejemplo, las entradas de un PE normalmente son salidas de otros PE y entonces se hace referencia a ellas por medio de dos índices i y j, donde i indica el PE del que viene y j a la que llega.
- Esta notación también tiene como propósito poder simplificar el manejo de una NN por medio de estructuras de datos en un programa de computadora.
- La entrada x_0 siempre es 1 por lo que a su peso se le conoce como umbral (*threshold*) que se usa para que la función no quede centrada en 0.

Función de activación

- Existen muchos tipos de funciones de activación que han sido utilizadas con éxito:
  - Discontinuas:
    - Escalón (puede ser binaria o bipolar)
    - Mixta
  - Continuas:
    - Lineal
    - Gaussiana
    - Sigmoidal
    - Tanh
    - ReLU (*rectified linear unit*) = max(0,x)

Sigmoidal

- El comportamiento más utilizado en la salida es el escalón (dos valores).
- Sin embargo, varios de los algoritmos utilizados requieren el uso de derivadas.
- Por esta razón, la principal función de activación utilizada es la Sigmoidal, cuyo comportamiento es muy cercano al escalón pero es continua.

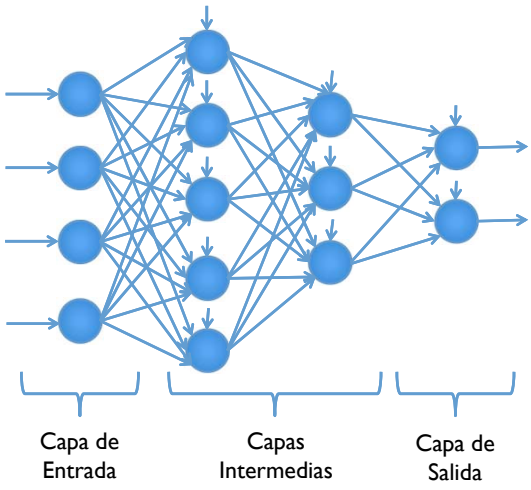


$$f(x) = \frac{1}{1+e^{-x}}$$

Estructura (topología) de una NN (Construcción)

- La NN más común se puede definir como un conjunto de PE con las siguientes características:
  - Ordenados en N capas.
    - La primera capa se le llama de ENTRADA y está formada por PE sin procesamiento.
    - La última capa se le llama de SALIDA.
    - Al resto de las capas se les llama INTERMEDIAS.
  - Las salidas de un PE sirve como entrada a todos los PE de la capa siguiente (conexión hacia adelante o *feedforward*).
  - Los PE no se pueden conectar con los PE de la misma capa.
- En lo sucesivo se supone este tipo de estructura a menos que se especifique otra cosa.

Representación gráfica de una NN



¿Cuántas capas?

¿Cuántas neuronas por capa?

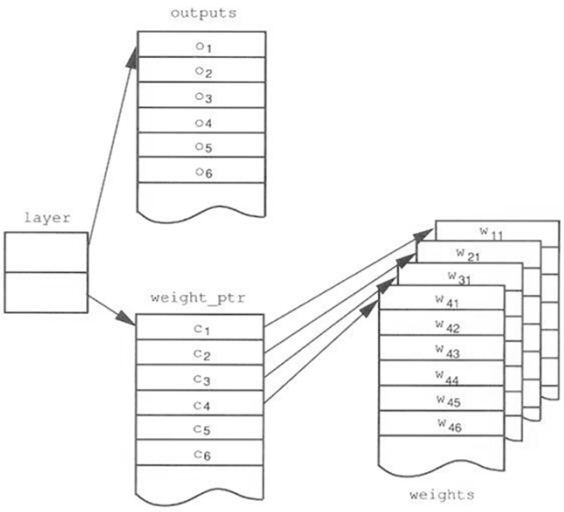
Número de capas y PE

- El número de PE en la capa de entrada debe ser igual al número de entradas.
- El número de PE en la capa de salida debe ser igual al número de salidas.
- Una NN debe contener al menos estas dos capas.
- El número de capas intermedias así como el número de PE en cada capa dependen del problema a ser resuelto. A la fecha no existe ninguna forma de obtener el número óptimo por lo que se determina a prueba y error.

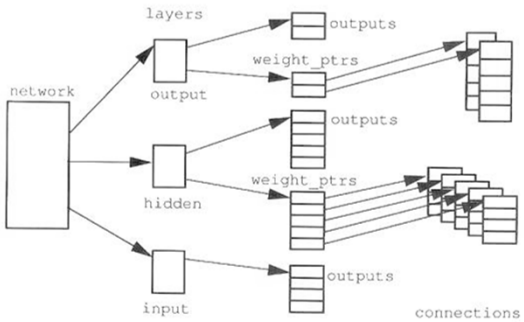
Hiperparámetros

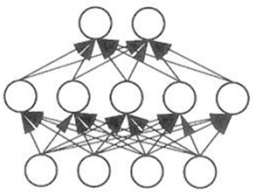
- Una NN tiene como parámetros a los pesos de la red, los cuales se deben determinar basados en los ejemplos de entrenamiento.
- Sin embargo, también existen algunos otros parámetros que los define el diseñador de acuerdo a su experiencia (o a prueba y error), los cuales se conocen como hiperparámetros:
  - Número de capas intermedias
  - Número de neuronas por capa intermedia
  - Tipo de función de activación
  - Algoritmo de optimización para calcular los pesos, etc.

Estructura de una capa (*layer*) con pointers



Estructura general de una NN con pointers





Actualmente: Matrices

- Con el auge que han tenido los lenguajes de programación orientados al manejo de matrices (e.g. Matlab u Octave) y las librerías de manejo de matrices para algunos otros lenguajes (e.g. numpy) todos los parámetros y datos utilizados en una NN se manejan por medio de matrices.
- Esto nos permite vectorizar prácticamente todo el proceso de entrenamiento y producción de una NN.

Objetivo de una NN

- Hay muchos tipos de NN, cada uno de ellos con comportamientos diferentes.
- Sin embargo, en todos los casos una NN se debe comportar como una FUNCIÓN, es decir, se espera que, dada una entrada específica, obtenga la salida correcta que le corresponde.
- Así, una NN puede usarse para:
  - Clasificar
  - Pronosticar
  - Recordar
  - Agrupar, etc.

Entrenamiento

- Para que una NN pueda "aprender" la función que relaciona a ciertas entradas con sus salidas es necesario entrenarla.
- Para esto se utiliza un algoritmo (llamado de entrenamiento o de aprendizaje) que recibe como entradas un conjunto de ejemplos (entradas, con sus salidas) y determina el valor que deben tomar los pesos de cada uno de los PE que componen la NN.
- Generalmente, el método usado es iterativo y va adaptando los pesos con varias lecturas del conjunto de ejemplos.

Nota sobre el conjunto de ejemplos

- La forma de entrenamiento más común de una NN es el aprendizaje supervisado.
- En este caso, el conjunto de ejemplos debe contener la salida esperada (deseada) para cada entrada.
- Sin embargo, una NN también puede entrenarse con técnicas de aprendizaje no supervisado, en cuyo caso el conjunto de ejemplos no requerirá de la salida.
- La definición del problema de entrenamiento dada a continuación supone aprendizaje supervisado a menos que se indique que se trata de otro tipo.

Problema de optimización

- En realidad, la determinación de los pesos es un problema de optimización, es decir, se desea encontrar el conjunto de pesos que logran minimizar para cada ejemplo, el error cometido por la salida de la NN con respecto a la salida deseada proporcionada con dicho ejemplo.

  $ejemplo_i = [\mathbf{X}_i, d_i]$ Salida NN = y_i
  $error_i = (d_i - y_i)$

- El problema de optimización pueda tratar de MINIMIZAR el error TOTAL, es decir, el error de todo el conjunto.

Producción

- Una vez que la NN está entrenada, si el algoritmo de entrenamiento fue adecuado, se espera que la NN represente "La" función que relaciona los ejemplos.
- Pero lo más importante es que la NN haya podido generalizar dicha función, es decir, que también funcione bien con entradas que no hayan aparecido en el conjunto de ejemplos, obteniendo su salida correcta.
- Si es así, la NN se puede usar en el modo de producción, es decir, se le da una entrada y su salida, considerada correcta, se utiliza para resolver algún problema particular.

Buen entrenamiento

- Un buen entrenamiento de una NN significa, como ya se comentó, que logre generalizar una función dados los ejemplos.
- Esto depende de varios factores pero los dos principales son:
  - El algoritmo de entrenamiento adecuado.
  - El conjunto de ejemplos adecuado.
- Los algoritmos han sido muy probados y se sabe que son adecuados.
- La selección del conjunto adecuado de ejemplos es entonces crucial para el buen funcionamiento de una NN.
- El conjunto de ejemplos debe ser REPRESENTATIVO (estadísticamente) de la población total.

Resumen: etapas de una NN

- La construcción de una NN consta de 3 etapas básicas:
  - Construcción. Se crean TODAS las estructuras de datos necesarias para contener todos los PE de la red y sus conexiones. Se debe conocer el número de capas intermedias y el número de PE por capa.
  - Entrenamiento. Se utiliza un conjunto representativo de ejemplos y un algoritmo para encontrar los pesos adecuados de cada uno de los PE de la NN.
  - Producción. Se da una entrada y la salida puede ser utilizada para tomar decisiones por parte del usuario.

Tipos de NN

- Hay muchos tipos de NN y éstos están definidos por dos variables:
  - La estructura (topología) de la NN.
  - El algoritmo de aprendizaje.
- Es muy común que la NN tome el nombre de su autor o de su algoritmo de aprendizaje.

Herramientas

- Hay tres herramientas fundamentales para poder construir NN:
  - Estructuras de datos
  - Álgebra matricial
  - Métodos de Optimización
- Algunos lenguajes de alto nivel se especializan en estos 3 puntos. Tal es el caso de Matlab® y Octave®.

Redes Neuronales

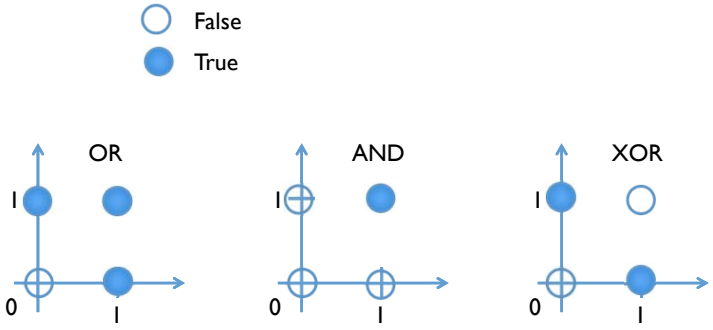
PERCEPTRÓN

El Perceptrón

- Fue el primer modelo de ANN desarrollado por Rosenblatt en 1958.
- Despertó un gran interés en los 60´s debido a su capacidad de aprender patrones sencillos.
- El perceptrón sólo tiene 2 capas.
  - Su capa de entrada con n neuronas.
  - Su capa de salida con 1 neurona.
- Su función de activación es escalón (binaria).
- [Minsky y Papert, 1969] lo atacan por resolver sólo problemas linealmente separables (problemas de orden 1 de acuerdo a su terminología).

Problema linealmente separable

- Un problema es linealmente separable cuando sus clases se pueden separar por medio de una línea recta.
- Ej. El AND y el OR son LS pero el XOR no lo es:



Algoritmo de aprendizaje

- Es de tipo supervisado, lo que requiere que se evalúe su salida y se modifique el sistema si es necesario.
  1. Inicializar los pesos (incluyendo el umbral). Se hace en forma aleatoria.
  2. Presentación de UN nuevo ejemplo ($\mathbf{X}_i$, d_i). Donde d_i es la salida deseada
  3. Cálculo de la salida actual con una función de transferencia escalón.
  4. Adaptación de los pesos.

     $$w_i(t+1) = w_i(t) + \alpha[d(t) - y(t)]x_i(t) \quad (0 \le i \le n)$$
  5. Si el error no fue menor que una constante ε para TODOS los patrones de entrada, ir a 2.
  6. FIN
- α es un factor de ganancia entre 0.0 y 1.0. Se debe ajustar a mano.
- n es el número de pesos.

Ejemplo

- Aprender la función OR escalón-binaria (1 = T, 0 = F).

X0	X1	X2	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Sean los pesos iniciales (w_0=1.5,w_1=0.5,w_2=1.5), seleccionados en forma aleatoria.
- Se van tomando uno a uno los patrones de entrada y se aplica el método expuesto anteriormente.

Proyecto

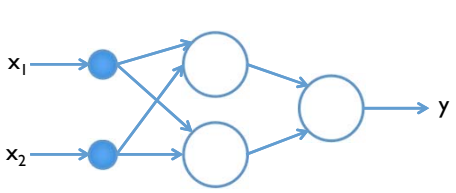
- Programar el Perceptrón para que se entrene partiendo de un archivo de N ejemplos.
- Probarlo con AND, OR y XOR (no debe poder).

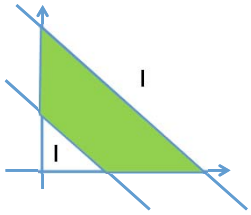
Redes Neuronales

PERCEPTRÓN MULTICAPA

El Perceptrón Multicapa

- Si el problema no es linealmente separable, se pueden generar múltiples regiones usando múltiples rectas (múltiples perceptrones).
- Esto requeriría un perceptrón más en la salida para seleccionar una de dichas rectas, lo cual nos lleva a tres capas.
- Para el caso del XOR la estructura sería:





Regla de Aprendizaje

- No se puede utilizar la misma regla que en el perceptrón debido a que sólo se conocen las salidas deseadas para las neuronas de la capa de salida pero no se conocen las salidas deseadas para las neuronas de las capas intermedias.
- Este problema, junto con el ataque de Minsky y Papert, ocasionaron que el avance en las NN se estancara por alrededor de 20 años.

Redes Neuronales

ADALINE Y MADALINE

La Adaline

- Las redes Adaline (ADAptive LINear Element) y Madaline (Multiple Adaline) fueron desarrolladas por Bernie Widrow en Stanford poco después de que Rosemblatt desarrollara el Perceptrón.
- Sus arquitecturas son básicamente las mismas que la del Percetrón y el Perceptrón Multicapa, respectivamente.
- En ambos casos la función de transferencia de sus PE es Escalón.
- La única diferencia es su mecanismo de aprendizaje.
- Ahora usan la Regla Delta de Widrow-Hoff, también conocida como LMS (mínimo error cuadrado por sus siglas en inglés, *Least Mean Squared error*).

Aprendizaje de la red Adaline

- El mejor vector de pesos se define como aquel que minimiza el error cuadrático medio de todo el conjunto de ejemplos.

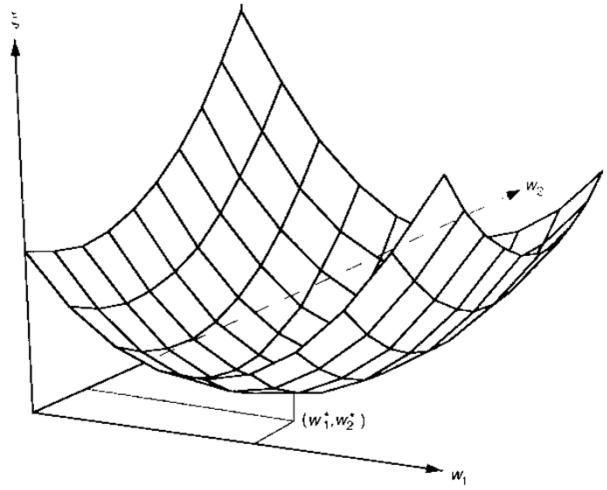
$$< \varepsilon^2 > = \frac{1}{2L} \sum_{k=1}^{L} \varepsilon_k^2$$

$$\varepsilon_k = (d_k - s_k)$$

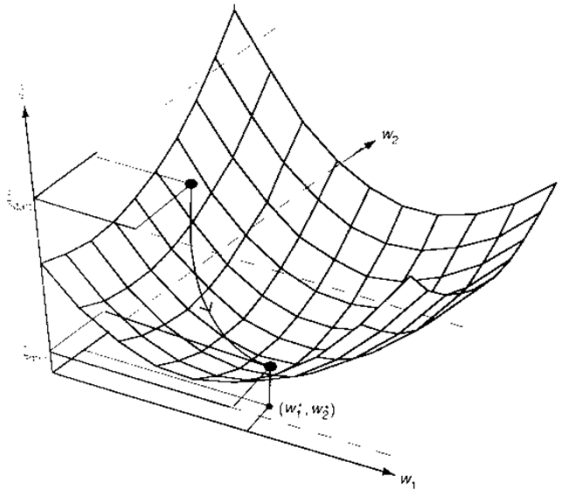
$$s_k = X_k W^T = \sum_{j=0}^{N} w_j \, x_j^k = Net_k$$

- El algoritmo es el mismo que el del Perceptrón excepto que el error lo calcula con Net y no con y.

Error en el Espacio de pesos



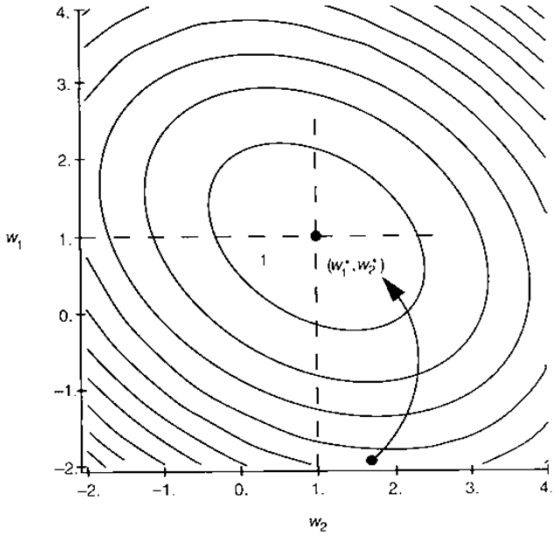
Moverse al óptimo



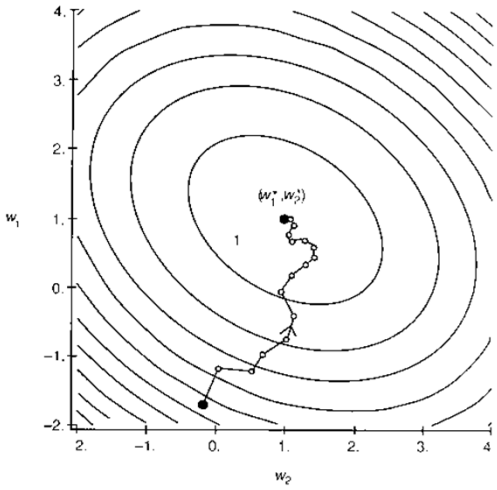
Gradiente descendente

- La idea es irse moviendo en el espacio de pesos hasta encontrar el punto en el que el error se minimiza.
- Se requiere algo que guíe la búsqueda.
- Una opción puede ser el gradiente negativo (para que disminuya) de la función del error.
- Usando *Hill Climbing* se resolvería el problema si es que el espacio no tiene óptimos locales.
- Como HC usa el gradiente negativo para bajar en lugar de subir se le conoce como Gradiente Descendente.

GD en 2D

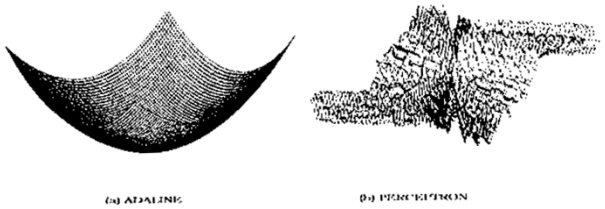


GD con paso α



Diferencia entre Perceptrón y LMS

- Aunque a simple vista no parece que exista gran diferencia entre ambos tipos de mecanismos de aprendizaje, el caso de Adaline mejora el del Percepetrón, ya que va a ser más sencillo alcanzar el mínimo del error porque el espacio de búsqueda no tiene variaciones.



Tomado de: Redes Neuronales Artificiales, Hilera y Martínez, Addison Wesley Iberomaricana, 1995

Obtención del gradiente del error

- El algoritmo de actualización de pesos es idéntico al del perceptrón sólo que la salida es lineal.
- Es decir:
  - $w_i = w_i + \alpha[d^{(k)} - y^{(k)}]x_i^{(k)}$
  - Donde:
    - (k), indica el número de ejemplo en la entrada.
    - i, indica el número de entrada
    - d, es la salida deseada, dada en el ejemplo de entrenamiento (k).
    - y, es la salida obtenida con la entrada (k), con función lineal.

Pág 57 de Feeman y Skapura

Algoritmo regla delta para Adaline

1. Inicializar los pesos a valores aleatorios.
2. Aplicar un vector o patrón de entrada, X_k, en las entradas de la ADALINE.
3. Obtener la salida lineal $s_k = X_k * W^T = \sum_{j=0}^{N} w_j x_i^{(k)}$, y calcular el error $\varepsilon_k = (d_k - s_k)$.
4. Actualizar los pesos
5. Repetir los pasos del 1 al 3 con todos los vectores de entrada (L).
6. Si el error cuadrático medio $\langle {\varepsilon_k}^2 \rangle = \frac{1}{2L}\sum_{k=1}^{L} {\varepsilon_k}^2$, es un valor reducido aceptable, termina el proceso de aprendizaje; si no, se repite otra vez desde el paso 2 con todos los patrones.

Redes Neuronales

RED MADALINE

La red MADALINE

- Es una idea semejante a la del perceptrón multicapa.
- Surge como una combinación de módulos ADALINE básicos en una estructura de capas que supera algunas de las limitaciones de la ADALINE.

Entrenamiento

- El entrenamiento no puede ser con LSM ya que sólo se puede aplicar a la capa de salida.
- La forma de aplicar las ideas del algoritmo LMS para entrenar una estructura tipo MADALINE pasa por sustituir la función de salida por una función continua derivable (se verá posteriormente).
- Existe otro método conocido como Regla II de MADALINE (MRII) parecido a un proceso de "prueba y error" con una inteligencia adicional basada en el principio de mínima perturbación [Widrow, 1988].

MRII

- El entrenamiento equivale a hacer una reducción del número de neuronas de salida incorrectos para cada uno de los ejemplos de entrenamiento que se den como entrada.
- Se debe notar que la salida de la red es una serie de PEs bipolares.
- Este método se puede aplicar con un algoritmo propuesto por [Freeman, 1991].

Proyecto

- Programar el algoritmo LMS para entrenar una ADALINE.
- Prácticamente es el mismo del perceptrón (ver proyecto del perceptrón).
- Córralo para el mismo problema (e.g. el AND) y en cada pasada:
  - Grafique el error BATCH para Perceptrón.
  - Grafique el error BATCH para el Adaline.
- Compárelos y comente al respecto.

Redes Neuronales

BACKPROPAGATION O BPN

Historia

- En 1986, Rummelhart, Hinton y Williams [Rummelhart, 1986], basándose en trabajos de [Werbos, 1974] y [Parker, 1982], formalizaron un método para que una red neuronal aprendiera la asociación que existe entre los patrones de entrada a la misma y las clases correspondientes, utilizando una estructura MADALINE.
- Este método se conoce como *backpropagation* (propagación del error hacia atrás).
- A pesar de sus limitaciones, ha ampliado considerablemente el rango de aplicaciones de una red neuronal.

Algoritmo Backpropagation

- Es una regla de aprendizaje que está basada en generalización de la regla delta.
- Una característica importante es la representación interna del conocimiento que es capaz de organizar en las capas intermedias para conseguir cualquier correspondencia entre la entrada y la salida de la red.
- Emplea un ciclo propagación-adaptación de dos fases.

Dos fases

- Fase 1
  - Aplicar un patrón de entrada como estímulo para la primera capa de las neuronas de la red.
  - Propagarla hasta generar una salida.
  - Comparar la salida obtenida con la salida deseada y calcular un valor de error para cada neurona de salida.
- Fase 2
  - Los errores se transmiten hacia atrás, partiendo de la capa de salida, hacia TODAS las neuronas de la capa intermedia que contribuyan a la salida, recibiendo un porcentaje de error aproximado proporcional a la participación de la neurona intermedia en la salida original.
  - Basándose en el valor de error recibido, se ajustan los pesos de la conexión de cada neurona.

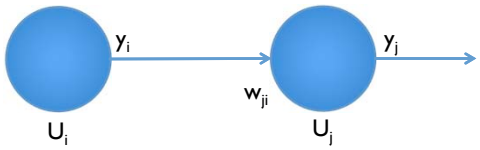
La regla delta generalizada

- Se aplica RN con capas intermedias con conexiones hacia adelante (*feedforward*) y cuyos PE tienen funciones de activación continuas (lineales o sigmoidales), dando lugar al algoritmo de retropropagación (*backpropagation*).
- Estas funciones continuas son no decrecientes y derivables.
- Utiliza también una función (o superficie) de error asociada a la red, buscando el estado de mínima energía (mínimo error) a través del camino descendente de la superficie (*Hill Climbing* inverso o gradiente descendente).

Funcionamiento y notación

- Su método es exactamente el mismo que la regla delta, es decir, la actualización del peso es proporcional al error (δ = sal.deseada – sal.obtendia).
- Dada una neurona (unidad U_i) y la salida que produce, y_i, el cambio que se produce en el peso de la conexión que une la salida de dicha neurona con la unidad U_j (w_{ji}) para un patrón de aprendizaje p determinado es:

$$\Delta w_{ji}(t+1) = \alpha \delta_{pj} y_{pi}$$



Modificación de neuronas internas

- Si consideramos la unidad U_j de salida, el error es como se conoce:

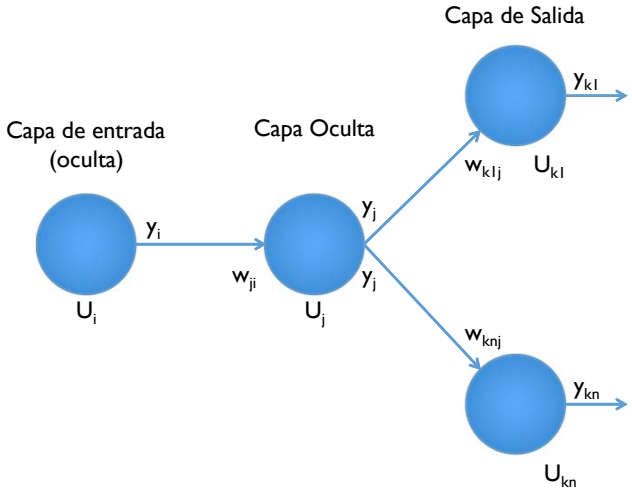
$$\delta_{pj} = (d_{pj} - y_{pj}) f'(net_j)$$

- Si U_j no es de salida, el error que se produce estará en función del error que se cometa en las neuronas que reciben como entrada la salida de dicha neurona:

$$\delta_{pj} = (\sum_k \delta_{pk} w_{kj}) f'(net_j)$$

- Donde el rango k cubre todas aquellas neuronas a las que está conectada la salida de U_j.
- Esto es lo que se denomina procedimiento de propagación del error hacia atrás.

Conexiones en una neurona de la capa oculta



Adición de un momento

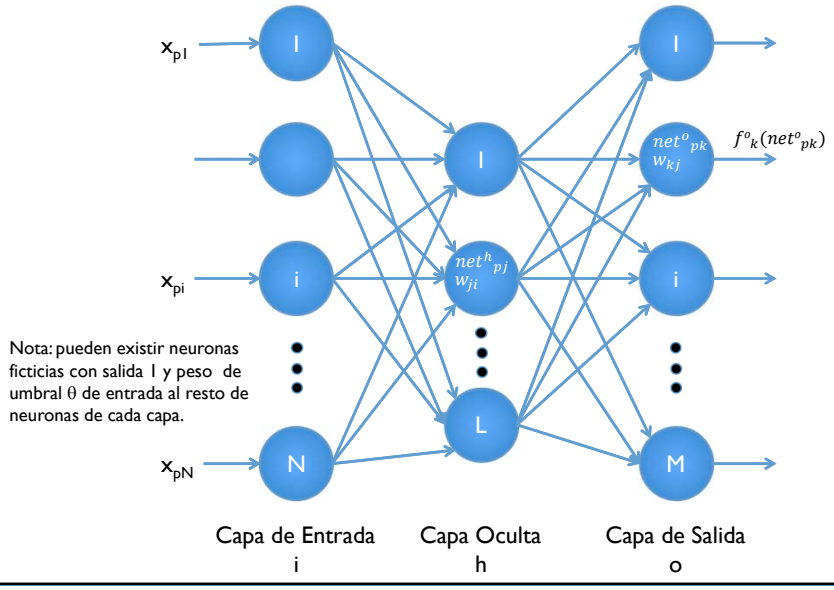
- Mientras mayor sea el paso α, mayor es la velocidad de convergencia, pero esto puede producir oscilaciones.
- Para filtrar esas oscilaciones, Rumelhart, Hinton y Williams, en 1986, sugirieron añadir a la expresión un término β (momento), quedando:

$$w_{ji}(t+1) = w_{ji}(t) + \alpha\delta_{pj}y_{pi} + \beta(w_{ji}(t) - w_{ji}(t-1))$$
$$\Delta w_{ji}(t+1) = \alpha\delta_{pj}y_{pi} + \beta\Delta w_{ji}(t)$$

Explicación del momento

- Con este momento se consigue la convergencia en menor número de iteraciones ya que si en t-1 el incremento de un peso era positivo y en t también, el descenso en t+1 es mayor.
- Si en t-1 el incremento era positivo y en t es negativo, el paso en t+1es más pequeño, lo cual es adecuado, ya que eso significa que se pasó por un mínimo y que los pesos deben ser menores para poder alcanzarlo.

Estructura de la red BPN



Algoritmo Aprendizaje de la BPN

1. Inicializar los pesos de la red con valores pequeños aleatorios.
2. Presentar un patrón de entrada X_p:x_{p1},x_{p2},…,x_{pN}, y especificar la salida deseada que debe generar la red, d_1,d_2,…,d_M (si la red se utiliza como clasificador, todas las salidas son cero excepto una, que será la de la clase a la que pertenece el patrón de entrada).
3. Calcular la salida actual de la red:
   1. Neurona j oculta:

$$net^h{}_{pj} = \sum_{i=1}^{N} w^h{}_{ji}\, x_{pi} + \theta^h{}_j$$

$$y_{pj} = f^h{}_j\left(net^h{}_{pj}\right)$$

   2. Neurona k de salida:

$$net^o{}_{pj} = \sum_{j=1}^{L} w^o{}_{kj}\, y_{pj} + \theta^o{}_k$$

$$y_{pk} = f^o{}_k\left(net^o{}_{pk}\right)$$

4. Calcular los términos del error para todas las neuronas

- Si la neurona k es de salida, su valor de delta es:

$$\delta^o{}_{pk} = (d_{pk} - y_{pk}) f^{o\prime}{}_k\left(net^o{}_{pk}\right)$$

- La función f debe cumplir el requisito de ser derivable:

$$f^o{}_k\left(net^o{}_{jk}\right) = \frac{1}{1+e^{-net^o{}_{jk}}} \text{ o } f^o{}_k\left(net^o{}_{jk}\right) = net^o{}_{jk}$$

- Para la función lineal tenemos

$$f^{o\prime}{}_k = 1.$$

- Para una sigmoidal:

$$f^{o\prime}{}_k = f^o{}_k\left(1 - f^o{}_k\right) = y_{pk}(1 - y_{pk})$$

- Por lo que los términos del error de salida:

  Para lineal $\delta^o{}_{pk} = (d_{pk} - y_{pk})$

  Para sigmoidal $\delta^o{}_{pk} = (d_{pk} - y_{pk}) y_{pk}(1 - y_{pk})$

4. Si la neurona j no es de salida

- La derivada parcial del error no puede ser evaluada directamente. Por lo tanto, se obtiene a partir de valores que son conocidos y otros que pueden ser evaluados.
- La expresión es:

$$\delta^h{}_{pj} = f^{h\prime}{}_j\,(net^h{}_{pj}) \sum_k \delta^o{}_{pk} w^o{}_{kj}$$

- En particular para la sigmoidal:

$$\delta^h{}_{pj} = x_{pi}(1 - x_{pi}) \sum_k \delta^o{}_{pk} w^o{}_{kj}$$

- Donde k se refiere a todas las neuronas de la capa superior a la de la neurona j.

5. Actualización de los pesos

- Se utiliza un algoritmo recursivo, comenzando por las neuronas de la capa de salida y trabajando hacia atrás hasta llegar a la capa de entrada, ajustando los pesos de la siguiente forma:
  - Neuronas de salida:

$$w^o{}_{kj}(t+1) = w^o{}_{kj}(t) + \Delta w^o{}_{kj}(t+1)$$
$$\Delta w^o{}_{kj}(t+1) = \alpha \delta^o{}_{pk} y_{pj}$$

  - Neuronas de capa oculta:

$$w^h{}_{ji}(t+1) = w^h{}_{ji}(t) + \Delta w^h{}_{ji}(t+1)$$
$$\Delta w^h{}_{ji}(t+1) = \alpha \delta^h{}_{pj} y_{pi}$$

- En ambos casos, para acelerar el proceso de entrenamiento (aprendizaje) se puede añadir el término momento:
  - Neurona de salida:

$$\beta(w^o{}_{kj}(t) - w^o{}_{kj}(t-1))$$

  - Neurona oculta:

$$\beta(w^h{}_{ji}(t) - w^h{}_{ji}(t-1))$$

6. Terminación

El proceso se repite hasta que el término de error total (para todos los patrones):

$$E_p = \frac{1}{2}\sum_{k=1}^{M} {\delta_{pk}}^2$$

Resulta aceptablemente pequeño para cada uno de los patrones aprendidos.

Deducción de la regla backpropagation

96

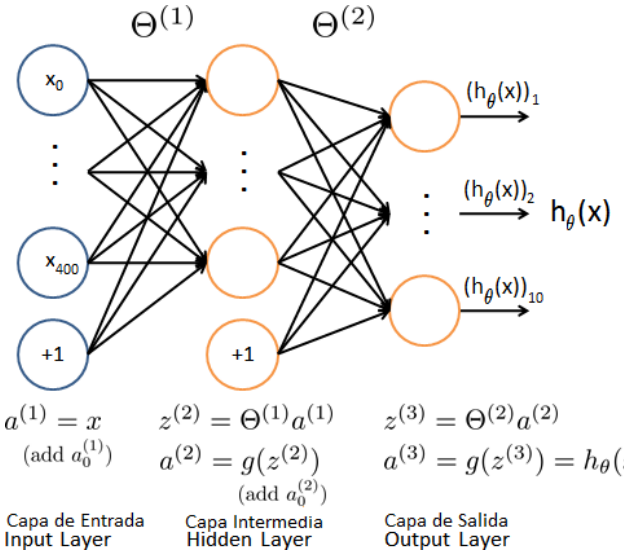
Proyecto One vs All

- El archivo ex3data1.mat contiene 5000 ejemplos de dígitos escritos a mano.
- Cada ejemplo de entrenamiento es una imagen en escala de grises de 20X20 pixeles.
- Cada imagen se desenrolló en una fila de 400 datos.
- Cada ejemplo está etiquetado del 1 al 10 (el 10 es para el 0).
- Programe "One vs All" para clasificar los dígitos. Reporte el porcentaje de aciertos sobre el archivo de entrenamiento.

Proyecto RN sólo feedforward

- El archivo ex3weights.mat, contiene los pesos de una RN ya entrenada, los cuales están en dos matrices Theta1 (25X401) y Theta2 (10X26).
- La RN consta de 3 layers:
  - 1 Layer de entrada de 400 neuronas.
  - 1 Layer intermedio de 25 neuronas, cada una con 401 pesos.
  - 1 Layer de salida de 10 neuronas, una para cada clase, cada una con 26 pesos.
- Haga una función que construya la RN, cargue los pesos y regrese la predicción del archivo de entrenamiento (debe ser de 97.5%).
- La salida de las neuronas es sigmoidal (ver figura).
- La matriz X contiene los ejemplos. No olvide la columna de 1's.

Arquitectura de la RN



Proyecto RN simple para XOR

- Crear una RN multicapa y entrenarla con BP, usando un archivo de entrenamiento que contiene m ejemplos.
- El proyecto debe pedir (al usuario, por archivo o por parámetros):
  - Número de neuronas en la capa de entrada.
  - Número de neuronas en la capa de salida.
  - Número de capas ocultas.
  - Número de neuronas en cada capa oculta.
  - Archivo de entrenamiento (se puede pedir el número de ejemplos que contiene el archivo o puede ser el primer dato del archivo).
- Pruébelo con una RN sencilla que pueda resolver el problema del XOR y luego con una complicada que resuelva el problema de la clasificación de dígitos escritos a mano.

Referencias

- J.A. Freeman and D.M. Skapura. Neural Networks: Algorithms, Applications, and Programming Techniques. Addison-Wesley, USA (1991).
- J.R. Hilera y V.J. Martínez. Redes Neuronales Artificiales: Fundamentos, modelos y aplicaciones. Adison-Wesley Iberoamericana, USA (1995).
- J.A. Anderson. Redes Neuronales. Alfaomega, México (2007).
- S. Russel and P. Norvig. Inteligencia Artificial un enfoque moderno. 2ª edición, Pearson, España (2004).