

Machine Learning

Redes Neurales y Deep Learning

Dr. Edgar Acuna
Departamento de Matematicas

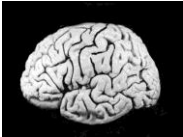

Universidad de Puerto Rico- Mayaguez

academic.uprm.edu/eacuna

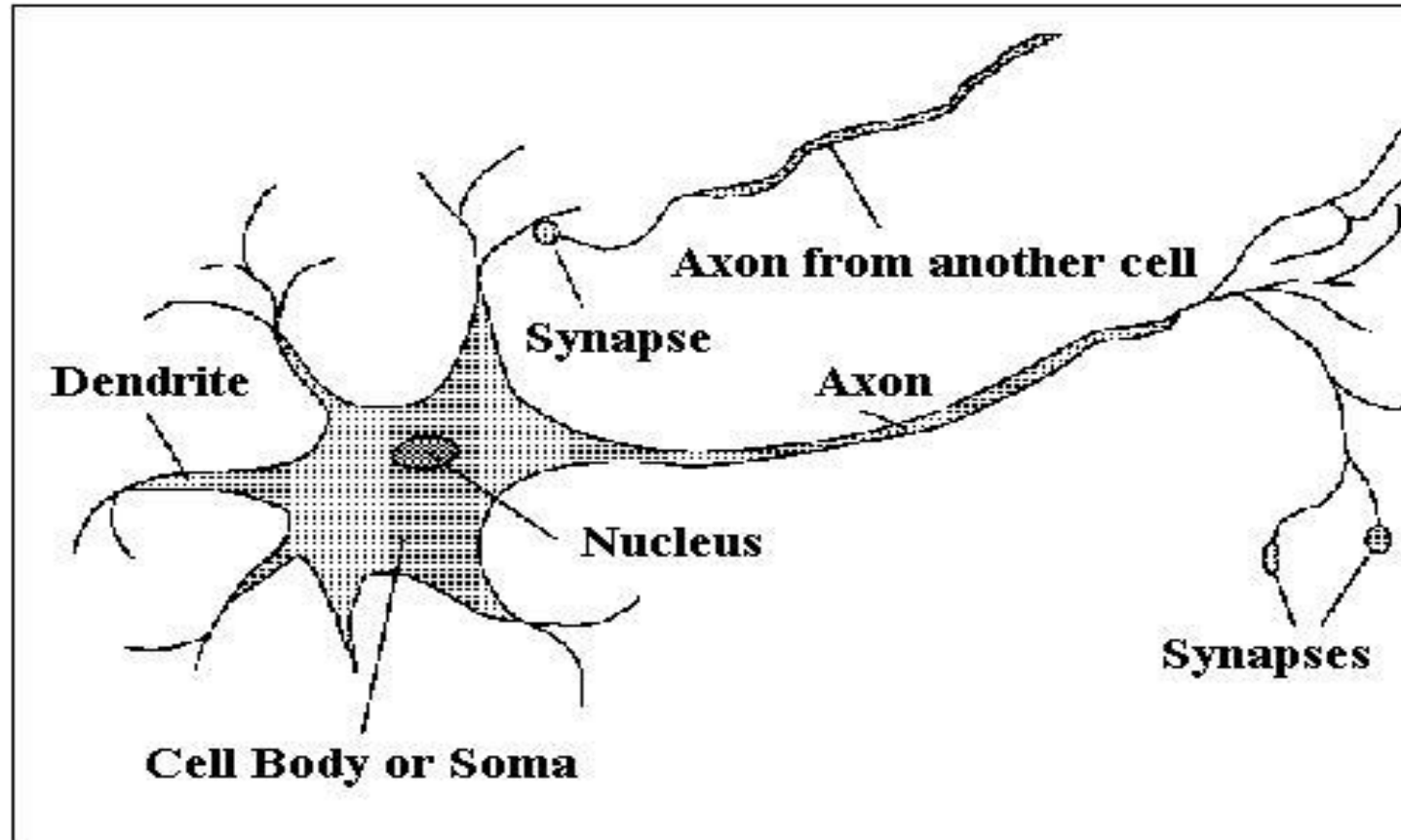
El cerebro tiene aproximadamente 1.5×10^{10} neuronas con un gran numero de conecciones (sinapses) que varia entre 10^3 a 10^4 . La red de neuronas del cerebro forma un sistema de procesamiento de informacion masivamente paralelo.

La redes neurales pueden ser consideradas como un intento de emular el cerebro humano. En general, redes neurales son representaciones de modelos matemáticos donde unidades computacionales son conectadas entre sí por un mecanismo que aprende de la experiencia, es decir de los datos que se han tomado.

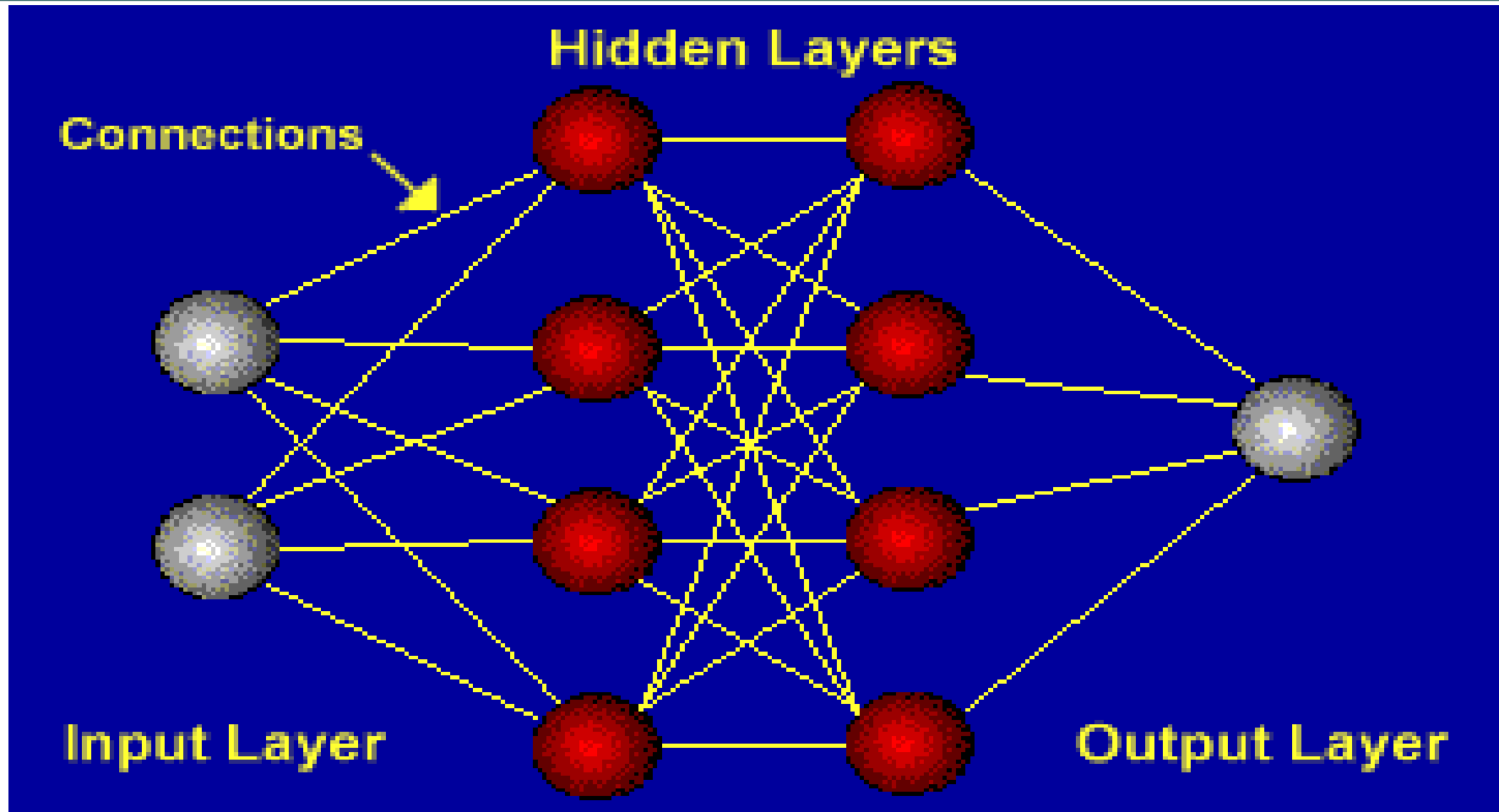
Las primeras ideas fueron introducidas por McCulloch y Pits (1943). Posteriormente, Rosenblatt (1958, 1962) introdujo el concepto de “Perceptron” (conjunto de neuronas) y lo trató de aplicar a clasificación. En 1974, Werbos, publico un primer algoritmo gradiente descendiente para backpropagation. Pero no fue hasta que en 1986, Hinton, Rumelhart y Williams que presentaron el algoritmo de “backpropagation” para el aprendizaje de una red neural, cuando estas cogieron auge.

	processing elements	element size	energy use	processing speed	style of computation	learns	Intellige nt, consci ous
	10^{11} synapses	10^{-6} m	30 W	1000 Hz	parallel, distributed	yes	usually
	2×10^9 transistors	10^{-6} m	70 W (CPU)	3×10^9 Hz	serial, centralized	something	Not(yet)

Una Neurona



Una red Neural



En estadística los trabajos de Ripley (1993) y Chen y Titterigton (1994) fueron pioneros en la incursión de los estadísticos en redes neurales.

Los usos en estadística de las redes neurales artificiales (ANN por sus siglas en inglés, muchas veces se elimina la palabra artificiales) incluyen:

- a. Análisis Discriminante
- b. Regresión
- c. Análisis por conglomerados.
- d. Deteccion de outliers
- e. Estimación de funciones de densidades

Tabla que relaciona terminos en redes neurales con terminos estadísticos

Estadística	Redes Neurales
Variables	features
Variables independientes	Inputs
Variables dependientes	Targets
Valores predichos	Outputs
Estimacion, Ajuste	Learning, training
Parametros	Weights
Transformaciones	Functional Links
Deteccion de Outliers	Novelty detection
Regresion, analisis discriminante	Supervised Learning
Analisis por conglomerados	Unsupervised Learning
Extrapolacion	Generalization

Tipos de redes neurales

- a) Para aprendizaje supervisado (análisis de regresión y análisis discriminante)
 - Multilayer Perceptron (MLP)
 - Radial basis function Networks (RBF)
 - Learning Vector Quantization (LVQ)

- b) Para aprendizaje no supervisado (análisis por conglomerados)
 - Hopfield Networks
 - Kohonen's Self-Organizing Maps
 - Adaptive Resonance Theory

Diagramas de redes

- Las neuronas son representadas por círculos y cajas, mientras que las conexiones entre ellas son representadas por flechas:
- Los círculos representan variables observadas. El nombre de la variable va adentro del círculo.
- Las cajas representan valores calculados con una función de uno o más argumentos. El símbolo dentro de la caja indica el tipo de función de activación usada.
- Dos líneas paralelas indican que los valores en cada extremo son ajustado por mínimos cuadrados, máxima verosimilitud o algún otro criterio de estimación.

El perceptron simple

Un "perceptron" es un modelo de una neurona. En términos de redes neurales un "perceptron" calcula una combinación lineal de inputs (posiblemente con un intercepto, que es llamado el término sesgo). Luego, una **función de activación**, la cual es por lo general no lineal es aplicada a esta combinación lineal para producir un output. Es decir, el output y_j es

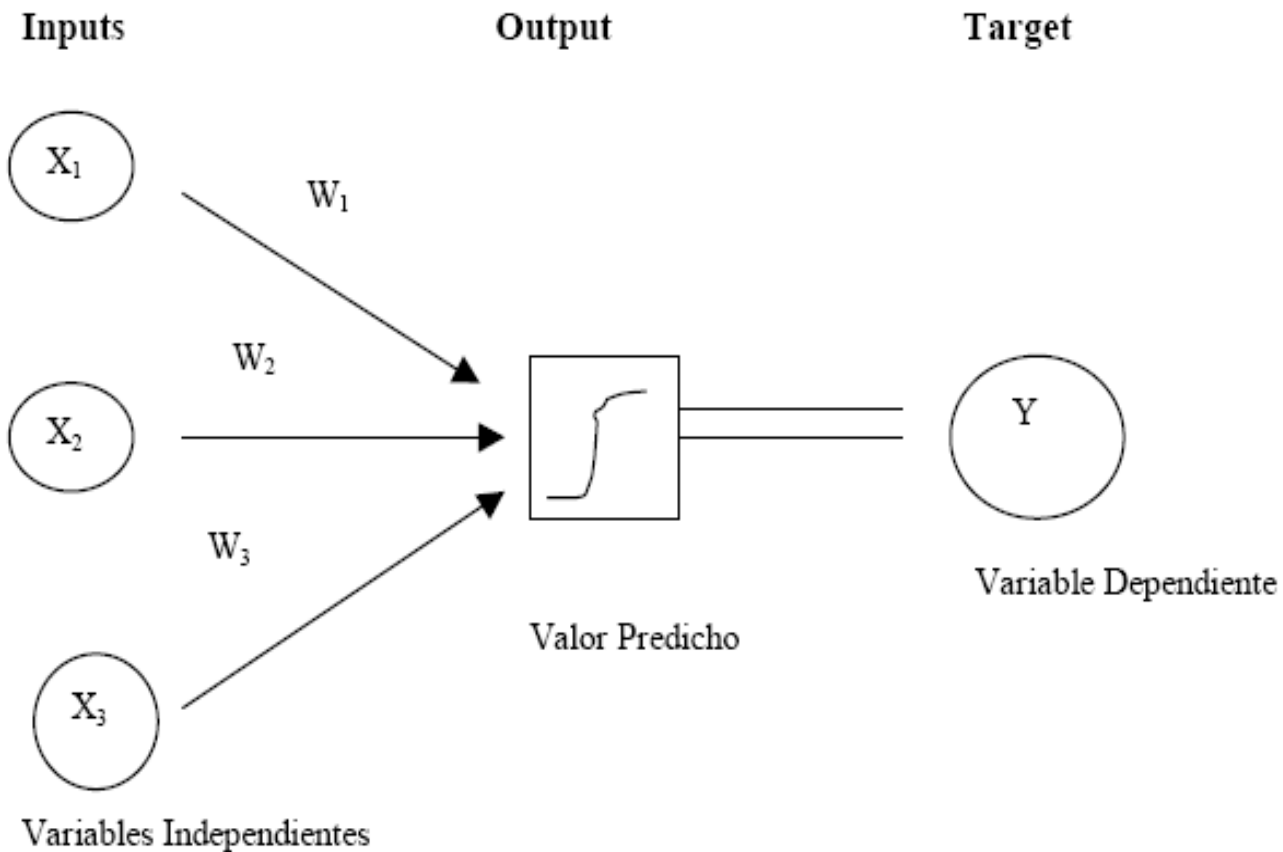
$$y_j = f_j \left(\sum_{inputs: i} w_{ij} x_i \right)$$

f_j representa a la función de activación y w_{ij} son los pesos, que los estadísticos llaman parámetros, La red neural **aprende** los pesos de los datos que se han recolectado.

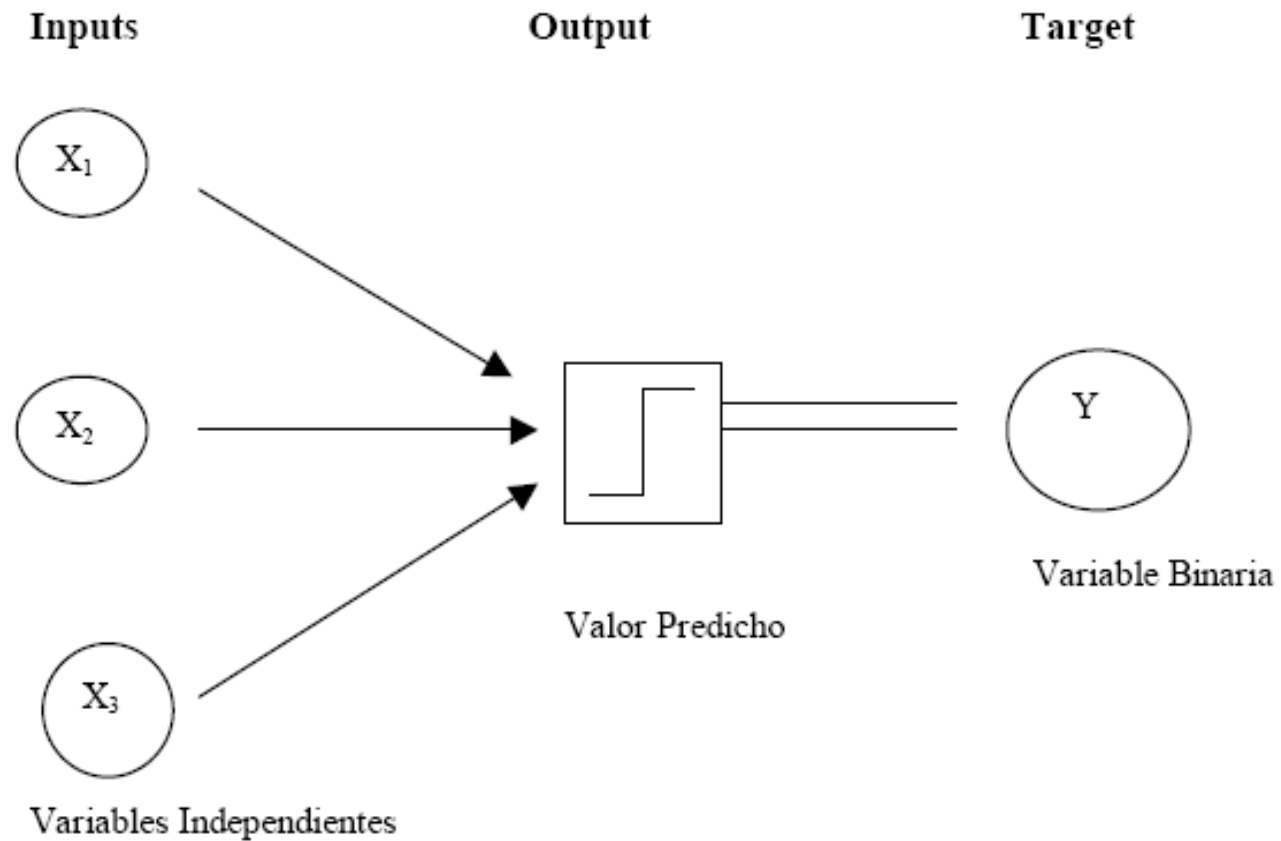
Funciones de activacion

Nombre	Funcion
Lineal	$A(x)=x$
Logistica	$A(x)=(1+e^{-x})^{-1}$
Gausiana	$A(x)=\exp(-x^2/2)$
Threshold	$A(x)=0$ si $x<0$, $A(x)=1$ en otro caso

Perceptron simple No-lineal=Regression Logistica



Perceptron para Analisis discriminante con dos clases



El Perceptron de varios niveles (Multilayer Perceptron, MLP)

Una red neural de dos capas puede ser escrita como un par de ecuaciones

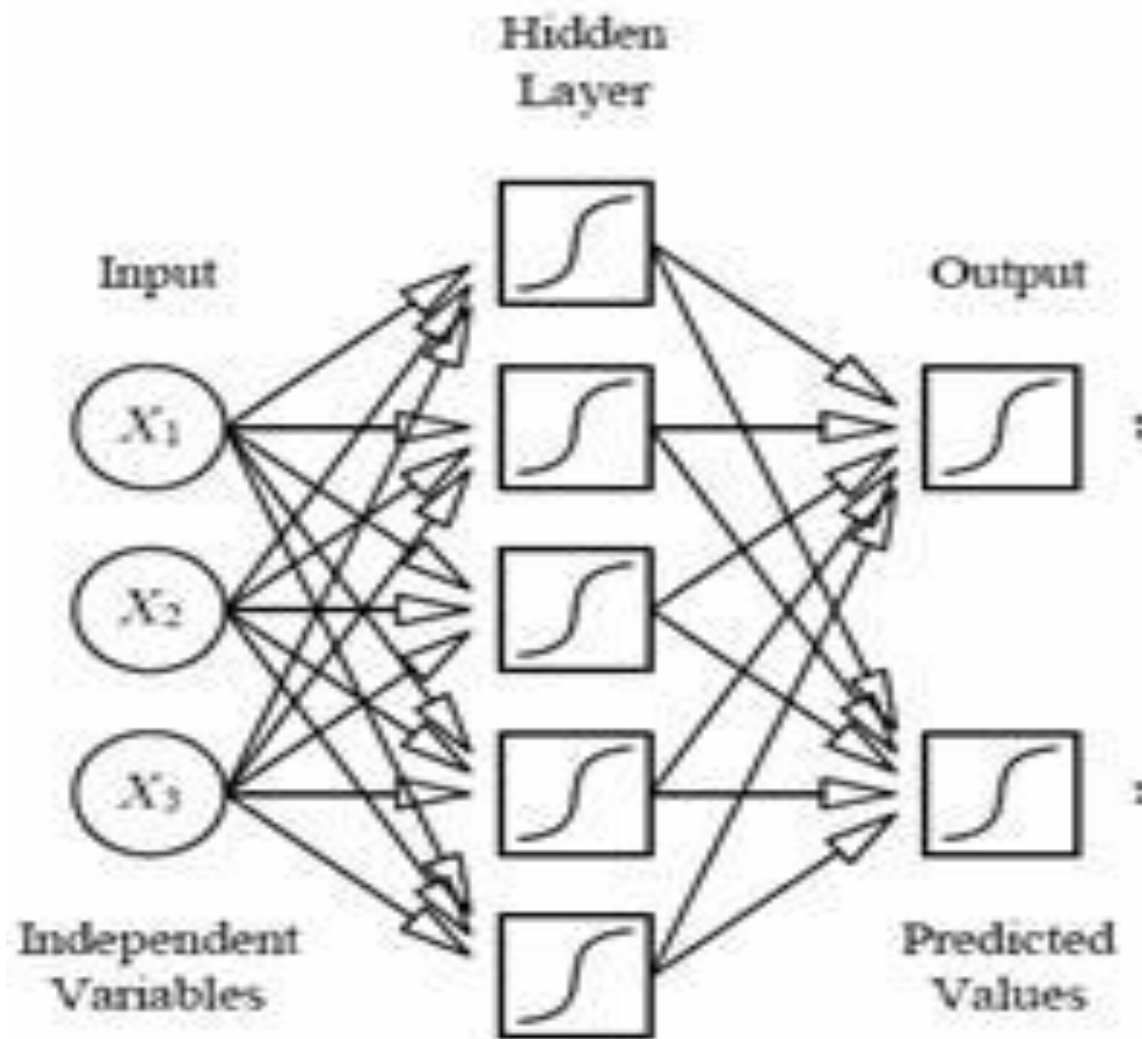
$$z_j = \phi_h(\alpha_j + \sum_i w_{ji} x_i)$$

$$y_k = \phi_o(\alpha_k + \sum_j w_{jk} z_j)$$

Aquí z_j son las unidades de la capa escondida ("Hidden Layer"), y_k las unidades de salida, ϕ_o y ϕ_h son funciones de activación. Casi siempre ϕ_h es la función logística, también conocida como la función de activación sigmoidea, ϕ_o puede ser lineal, logística o "threshold". Sustituyendo la primera ecuación en la segunda se obtiene

$$y_k = \phi_o(\alpha_k + \sum_j w_{jk} \phi_h(\alpha_j + \sum_i w_{ji} x_i))$$

donde w_{ij} y α_i representan los pesos.



En los problemas de clasificacion con mas de dos clases, digamos C, se acostumbra a usar como funcion de salida , la funcion “softmax” , ϕ_o , definida por

$$\phi_o(z) = (\phi_1(z), \dots, \phi_k(z), \dots, \phi_C(z))$$

donde

$$\phi_k(z) = \frac{e^{t_k z}}{\sum_{j=1}^C e^{t_j z}}$$

El MLP tambien es llamado red de alimentacion hacia adelante (FFNN) o red de propagacion hacia atras.

Entrenamiento de una red neural

- Esto es equivalente a estimación en estadística. Los pesos w son elegidos de tal manera que alguna medida de ajuste se minimize.
- **Para Regresión:** Se minimiza la suma de los cuadrados de los errores, definida por

$$E = \sum_{i=1}^n \sum_{j=1}^J (\hat{y}_j^i - y_j^i)^2$$

con respecto a los pesos $w=(\alpha_j, w_{ij})$

Entrenamiento de una red neural(cont)

- Para clasificacion: Para dos clases se minimiza la funcion de entropia cruzada, dada por

$$E = \sum_{i=1}^n [y_i \log(\frac{y_i}{\hat{y}_i}) + (1 - y_i) \log(\frac{1 - y_i}{1 - \hat{y}_i})]$$

Aqui $y_i = 1$ si la observación \mathbf{x}_i pertenece a la clase 1 e igual a 0 en otro caso. Las \hat{y}_i son las estimaciones de la probabilidad posterior de que caer en la clase 1 si se observa \mathbf{x}_i . Esto es equivalente a la estimación de parámetros en una regresión logística binaria

Para J clases se minimiza la funcion de entropia

$$E = \sum_{i=1}^n \sum_{j=1}^J y_j^i \log\left(\frac{y_j^i}{\hat{y}_j^i}\right)$$

E puede ser minimizada usando métodos de análisis numérico y modelos no lineales incluyendo:

Gradiente Descendente, métodos Quasi-Newton (recomendado si el número de pesos es menor de 1000), Método Gradiente Conjugado (recomendado si hay una gran cantidad de pesos a ser estimados), "Simulated Annealing", Particle Swarm Optimization" y Algoritmos Genéticos.

La mayor dificultad de minimizar E es la presencia de múltiples mínimos y se corre el riesgo de elegir uno de ellos que no sea óptimo. A menudo es necesario reinicializar el proceso de minimización usando distintos valores iniciales.

MINIMIZACION de E

Una aproximación lineal a $E(w)$ en una vecindad de una matriz de pesos w^o está dada por

$$E(w) \approx E(w^o) + \nabla E(w^o)(w - w^o)$$

También se puede usar una aproximación de segundo orden que involucra el uso de la matriz Hessiana $H^1 = \nabla \nabla E$ de segundas derivadas.

Sea w la matriz de pesos de la red neural entonces la gradiente de $E(w)$ es un campo vectorial de derivadas definido por

$$\nabla E(w) = \left(\frac{dE(w)}{dw_1}, \frac{dE(w)}{dw_2}, \dots \right)$$

Por la forma del modelo de una red neural la gradiente puede ser calculada usando la regla de la cadena y a través del algoritmo de "Backpropagation".

MLP en R

La funcion **nnet** de la libreria **nnet** permite hacer tanto regresion (por defecto) como clasificacion supervisada usando MLP. En este ultimo caso, mediante la estimacion de las funciones de probabilidades posteriores.

Por ejemplo, para hacer clasificacion del conjunto de datos notas, se ejecuta los siguientes comandos

```
clases=class.ind(eje1dis[,3])
```

```
a=nnet(eje1dis[1:32, c(1,2)],clases,entropy=T,size= 5)
```

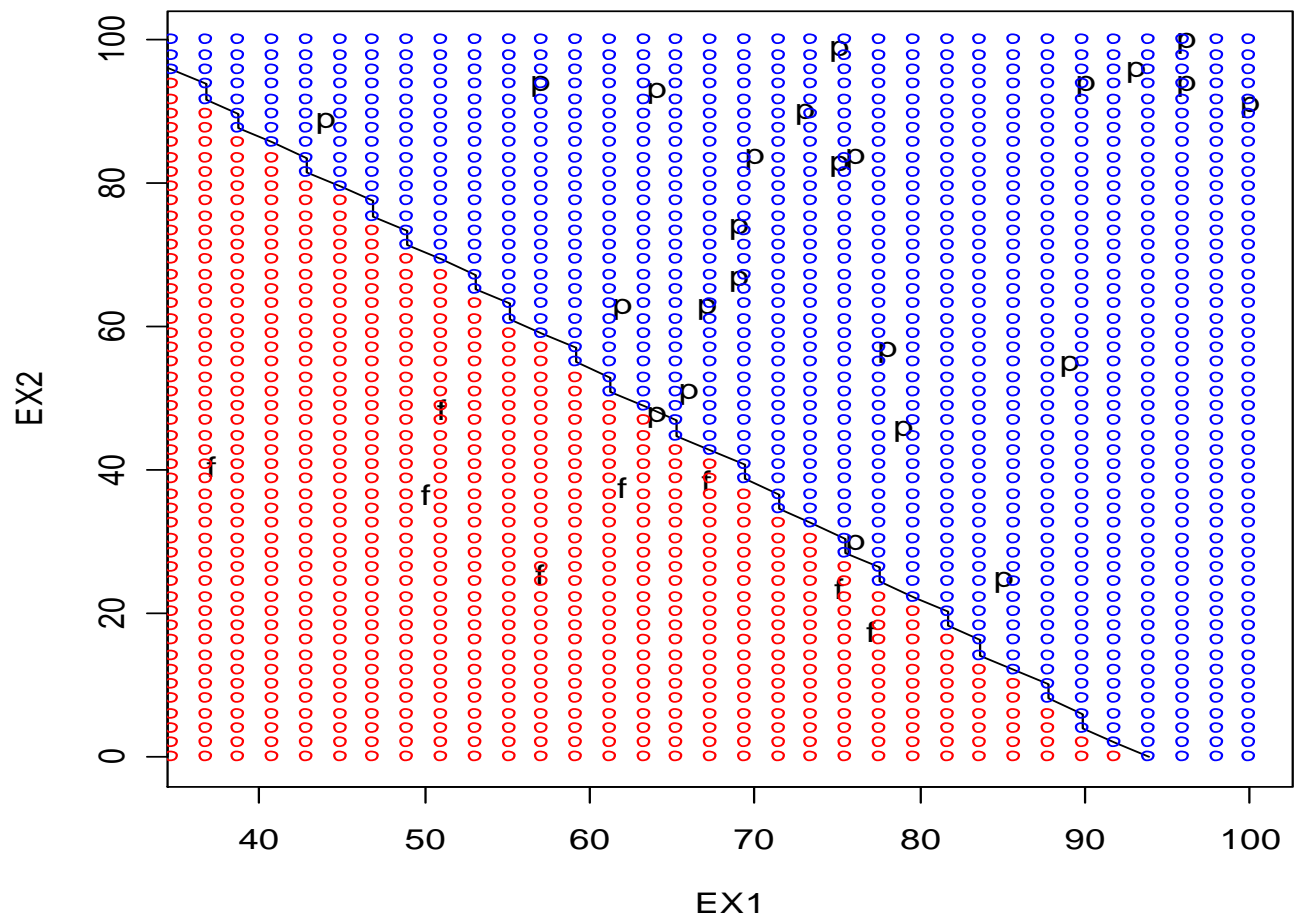
Aqui clases representa una matriz de orden $n \times k$, donde k es el numero de clases. Las entradas de esta matriz son ceros excepto en la columna correspondiente a la clase que pertenece la instance donde vale 1.

```
> error=mean(as.numeric(eje1dis[,3])!=max.col(a$fit))
```

```
[1] 0
```

El error aparente es 0. ver la figura siguiente

Grafica de la frontera usando redes neurales



MLP en R

Para ver todos los parametros estimados en la red

```
> summary(a)
```

a 2-5-2 network with 27 weights

options were - entropy fitting

b->h1 i1->h1 i2->h1

0.54 -0.30 -0.01

b->h2 i1->h2 i2->h2

-0.50 0.15 0.27

b->h3 i1->h3 i2->h3

-0.43 -0.10 -0.69

b->h4 i1->h4 i2->h4

-4.91 -1.87 2.64

b->h5 i1->h5 i2->h5

-63.91 0.70 0.42

b->o1 h1->o1 h2->o1 h3->o1 h4->o1 h5->o1

19.06 -0.18 17.86 -0.28 -3.59 -58.13

b->o2 h1->o2 h2->o2 h3->o2 h4->o2 h5->o2

-18.22 0.00 -16.89 -0.02 -1.77 59.99

MLP en R

Para ver las probabilidades posteriores

```
a$fit
```

```
  f p
```

```
1 0 1
```

```
2 0 1
```

```
3 0 1
```

```
4 0 1
```

```
5 0 1
```

```
6 0 1
```

```
7 0 1
```

```
8 0 1
```

```
9 0 1
```

```
10 0 1
```

```
.....
```

Para predecir las clases

```
> max.col(a$fit)
```

```
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
```


Ejemplo: Diabetes

```
classes=class.ind(diabetes$V9)
a=nnet(diabetes[,1:8],classes,entropy=T,size=5,maxit=1000)
> summary(a)
a 8-5-2 network with 57 weights
options were - entropy fitting
b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1
-0.67 -0.17 -0.69 -0.26 -0.62 -0.57 0.40 -0.33 0.18
b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2
-0.66 -1.99 -5.22 -8.09 4.13 4.24 -5.63 -0.75 -5.83
b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3
-391.25 3.94 6.29 -20.10 11.53 -3.16 17.50 61.76 27.69
b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4 i7->h4 i8->h4
0.29 -0.38 0.56 0.04 0.13 0.30 0.37 -0.59 0.42
b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5 i7->h5 i8->h5
-0.50 0.38 -0.20 -0.32 0.03 0.03 -0.01 -0.13 -0.10
b->o1 h1->o1 h2->o1 h3->o1 h4->o1 h5->o1
1.52 -0.03 -3.98 -3.51 2.43 0.46
b->o2 h1->o2 h2->o2 h3->o2 h4->o2 h5->o2
-2.06 -0.01 3.98 3.51 -1.90 -0.54
```



```
a$fit[1:10,]  
      1      2  
1 0.60852318 0.39147880  
2 0.60852318 0.39147880  
3 0.60852318 0.39147880  
4 0.98117196 0.01882776  
5 0.60852318 0.39147880  
6 0.60852318 0.39147880  
7 0.60852318 0.39147880  
8 0.60852318 0.39147880  
9 0.02833099 0.97166865  
10 0.60931439 0.39068758  
> error=mean(diabetes[,9]!=max.col(a$fit))  
> error  
[1] 0.34375
```

Ejemplo de overfitting con nnet

Consideraremos el conjunto de datos Diabetes

Numero de unidades en el layer	5	10	20	50	80	90
Error Aparente	29.29	26.82	20.05	14.71	7.55	5.85

Hay mucha variabilidad en la estimacion del error

Medidas remediales para “overfitting”

Las redes neurales tienden a caer en “overfitting”, es decir a dar errores de clasificación iguales a cero. Las siguientes son algunas medidas remediales al problema

a) Parar el proceso iterativo de estimar el minimo de E. Se asume que se tiene un conjunto de validación y se para el proceso iterativo cuando el rendimiento de la red neural en el conjunto de validación se empieza a deteriorar.

b) Regularización: En este caso se se añade una penalidad a $E(w)$ y luego se minimiza. Más específicamente,
$$\text{Min}_w[E(w) + \lambda \phi(w)].$$

Aquí λ es la constante de regularización y ϕ es la función de penalidad del modelo. El ejemplo más simple es la regularización conocido como “weight decay” (deterioro de pesos) definido como

$$\text{Min}_w[E(w) + \lambda \phi \sum w^2]$$

Esto es muy similar a regresión “ridge”.

-
- c) **Promediar:** En este caso se eligen varios valores iniciales para el proceso iterativo de minimizar $E(w)$ y se promedian las predicciones obtenidas.
 - d) **Añadir ruido.** En este caso se añade un ruido (“noise”) a cada variable de entrada y luego se ajusta la red neural. Se repite el proceso varias veces y luego se promedia.
 - e) **Bagging** (“Bootstrap aggregating”). Se toma varias muestras con reemplazamiento de la muestra de entrenamiento y se ajusta una red neural para cada una de ellas. Luego se promedian las predicciones finales en el caso de regresión o se asigna la observación a la clase más votada en el caso de clasificación. Otros métodos relacionados llamados ensembles o metaclassificadores son “Arcing”, “Stacking”, “Boosting”.

Effecto del uso de “decay”

Decay	0	5	10	.1	.5
Error aparente	14.71	20.31	24.47	3.12	6.64

Error por CV para MLP (en R)

```
cv10mlp=function(data,units,repet)
#This function finds an estimate for the misclassification error through 10 fold cross validation for the
  multilayer perceptron neural network ( two layers only:the input layer and one hidden layer)
#Requires the nnet function from Ripley's "nnet" library
# data: the dataset to be used
# units: the number of units in the hidden layer
# repet: the number of repetitions
cv10mlp(ionosphere,25,5)
[1] 0.07977208 0.09971510 0.09686610 0.09686610 0.09971510
The misclassification error is
[1] 0.0945869
> cv10mlp(vehicle,36,5)
[1] 0.3085106 0.3510638 0.3226950 0.3274232 0.3191489
The misclassification error is
[1] 0.3257683
```

Redes de Funciones bases radiales (RBF nets)

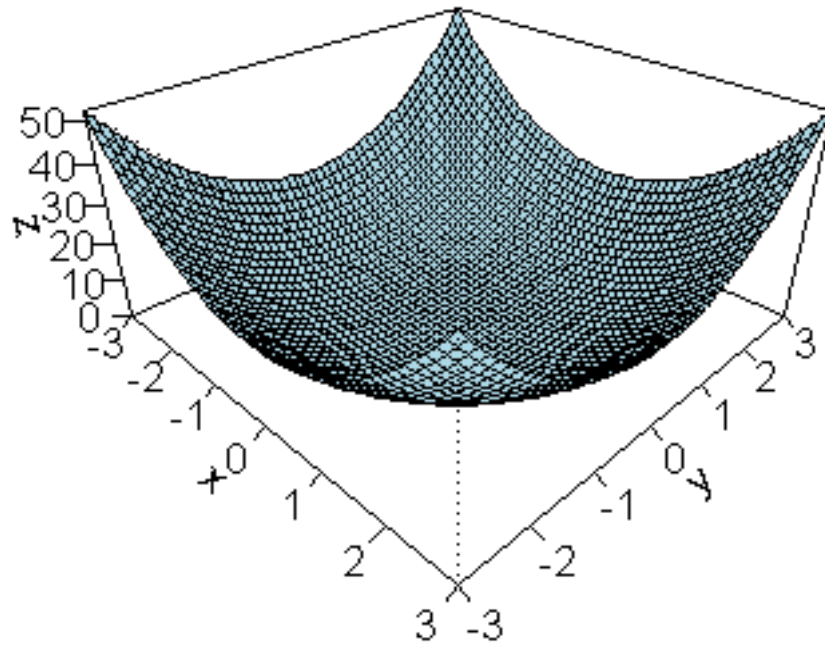
Las redes de funciones bases radiales fueron inicialmente aplicadas a aproximación de funciones.

Roomhead y Lowe (1988) fueron los primeros en usarlos para clasificación. La función base radial (RBF) puede ser escrita matemáticamente como una combinación lineal de funciones no lineales radialmente simétricas de la siguiente forma:

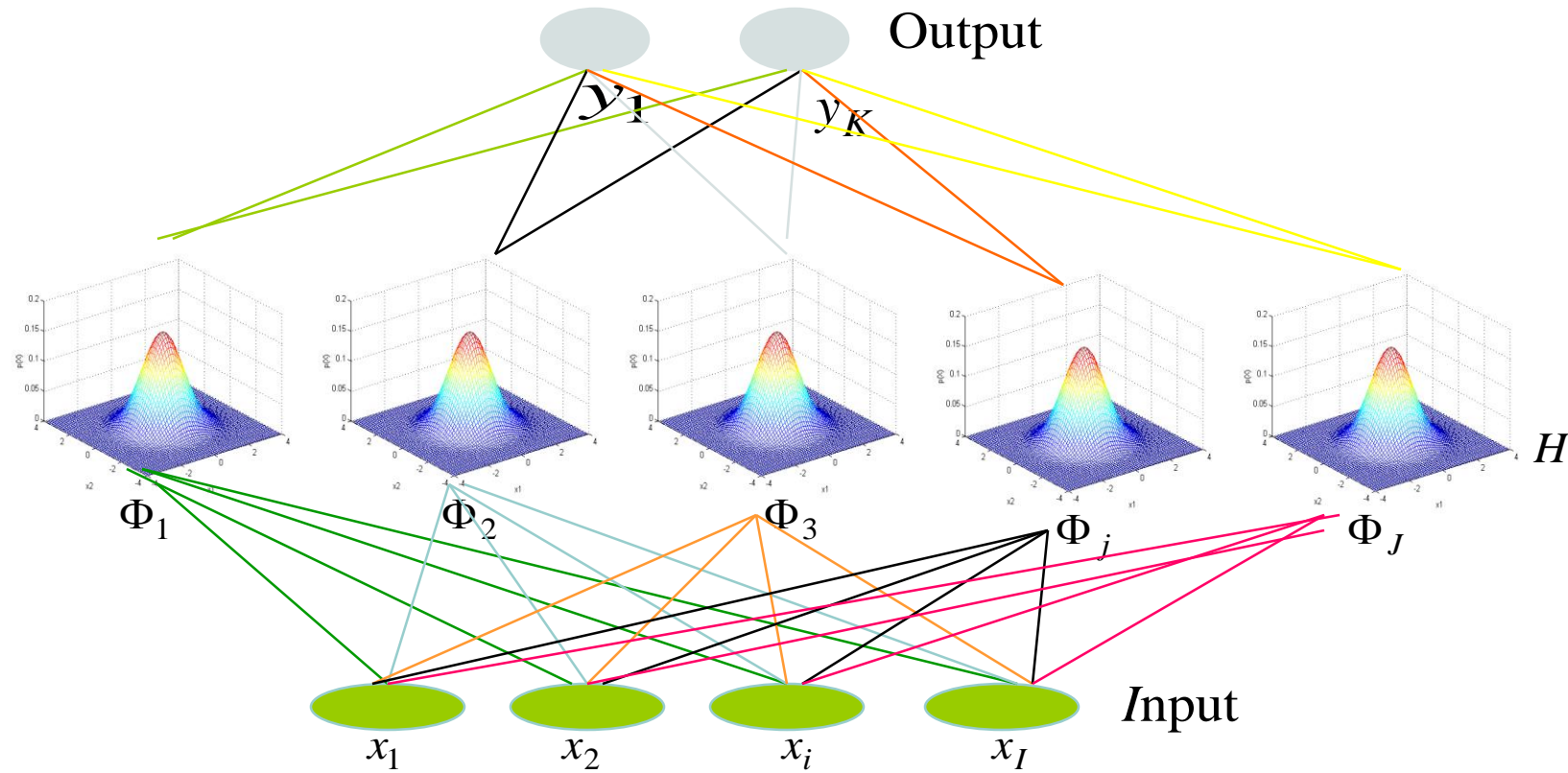
$$y = \alpha + \sum_{j=1}^M \beta_j \phi_j(\|x - c_j\|)$$

donde c_j son centros pre-especificados, α y β_j son pesos a ser estimados, las ϕ_j son las funciones bases usualmente se consideran las mismas para todo j y las mas usadas son la gaussiana $\phi(r)=\exp(-r^2/2\sigma)$, la cuadrática $\phi(r)= r^2+ar +b$ y la "thin plate spline" $\phi(r)=r^2\log(r)$. Aqui r representa radio. En 2D, $\phi(-x,-y)=\phi(x,y)$.

Grafica de la funcion 2D Thin plate spline



Arquitectura de Net RBF Gaussiana



El modelo es muy similar a estimacion de densidad usando mezclas gaussianas o estimadores por kernel. Por lo general los centros son los centroides de los clusters determinados por cualquier algoritmo de clustering (por ejemplo k-means). Tambien se puede usar los parametros que resultan de estimar mezclas gaussianas por el algoritmo EM.

El número de componentes M es elegido de tal manera que no ocurra ni "underfitting" ni "overfitting" y usualmente es estimado por validación cruzada.

En terminos de clasificacion, RBF nets es usado para estimar la probabilidad posterior de cada clase, $P(C_j/x)$ y asignar luego el objeto x a la clase j con la maxima probabilidad posterior

Aplicacion al Problema XOR

Input 1(x_1)	Input2(x_2)	Clase
0	0	1
1	1	1
0	1	0
1	0	0

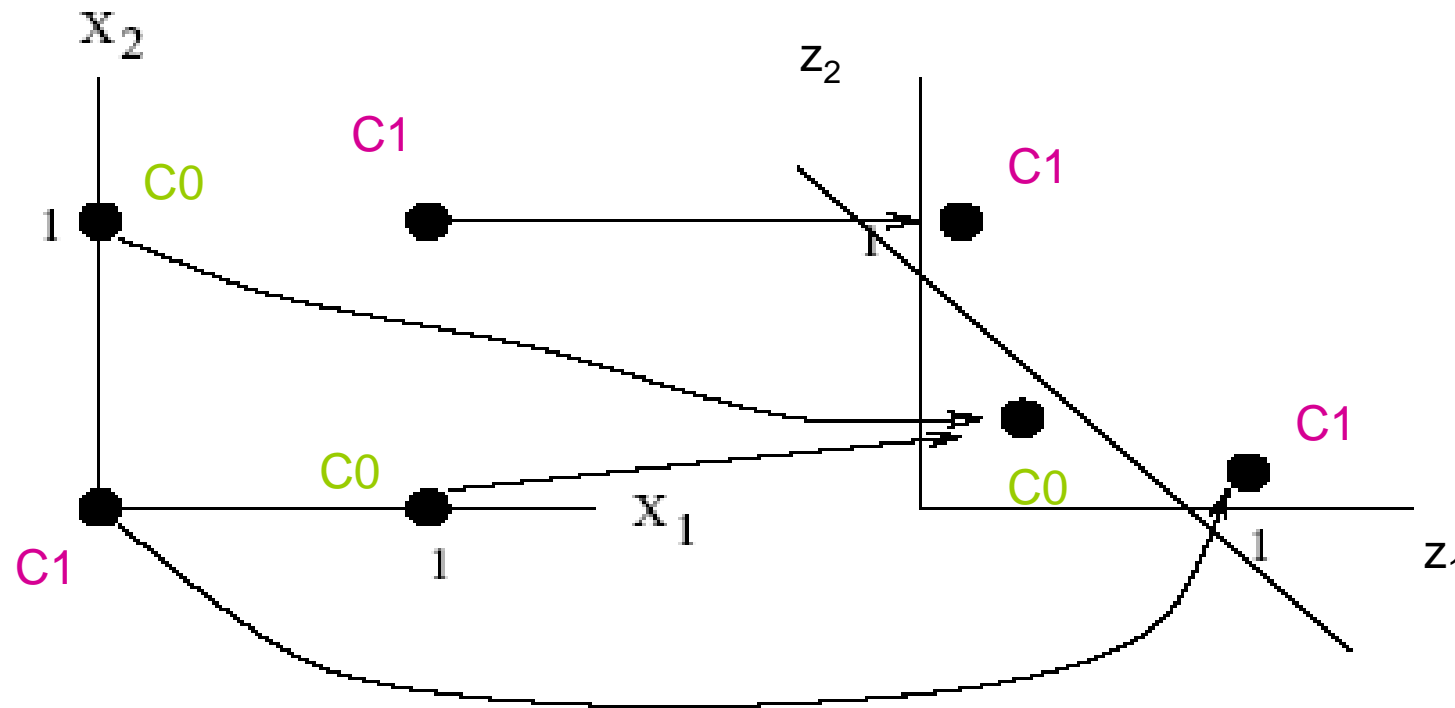
Usando una RBF Gaussiana

$$z_1 = \phi_1(x) = e^{-\|x - c_1\|^2} \text{ con } c_1 = (0, 0)$$

$$z_2 = \phi_2(x) = e^{-\|x - c_2\|^2} \text{ con } c_2 = (1, 1)$$

x1	x2	z1	z2
0	0	1	.135
0	1	.37	.37
1	0	.37	.37
1	1	.135	1

RBF para XOR (Cont)



Notar que en el plano (Z_1, Z_2) las dos clases C_0 y C_1 se separan

Error de clasificacion usado el CLASIFICADOR RBF (LOZANO, 2002)

Conjunto de Datos	Error (%)
Iris	2.67
Breastw	2.64
Bupa	27.25
Crx	14.09
Diabetes	24.48
Glass	29.84
Heartc	21.55
Ionosfera	9.4
Sonar	21.15
Vehicle	28.05
Segment	11.82
Landsat	13.24

-
- Una RBF net se entrena mas rapido que un MLP net y es mas facil interpretarlo. Pero el MLP es mas rapido para hacer predicciones.
 - RBF nets son mas sensitivos a la maldicion de dimensionalidad que MLP nets. Tiene problemas cuando el numero de variables predictivas es grande.

Ventajas y desventajas de redes neurales:

- a) Son buenos para hacer predicciones pero malos para ser entendidos.
- b) Buenos para analizar conjuntos de datos de gran tamaño y de estructura compleja.
- c) Pueden rendir mejor o peor que los métodos estadísticos dependiendo del tipo de problema.
- d) Carecen de una buena teoría estadística para hacer inferencias, diagnósticos y selección de modelos.

Deep Learning (Aprendizaje profundo)

En 2006, Geoffrey Hinton(ahora en Google) demostro que un tipo de red neural llamada, deep belief network, puede ser eficientemente entrenada usando una estrategia llamada greedy layer-wise pretraining.

Hinton y sus asociados popularizaron el uso del termino “deep learning” para enfatizar que ahora se podian entrenar redes neurales mas profundamente que antes.

Al presente se afirma que, deep neural networks, tiene mejor rendimiento que otros metodos de Machine Learning.

El exito de deep learning ha hecho que aparezcan diversas librerias para varios lenguajes de programacion. Por ejemplo, Keras (python y R), Theano, Torch y TensorFlow (Google Brain).

Los programas de Deep-learning aprenden a reconocer patrones en representacion digital de sonidos, imagenes y de otro tipo de datos.

La idea basica,que es de varias decadas atras) es que el programa pueda simular el arreglo de gran numero de neuronas de la neocorteza en un red neural artificial.

Gracias a las mejoras en la formulacion matematica y en el constante incremento en el poder de las computadoras(procesadores GPU y clusters), los cientificos pueden modelar ahora muchas mas capas de neuronas que antes.

En Junio del 2012, un sistema de Deep Learning de Google pudo reconocer gatos en 10 millones de videos de Youtube con el doble de eficiencia que previos programas de reconocimiento de imagenes. Google Tambien ha usado la tecnologia para reducir la tasa de error de su programa de reconocimiento de habla en los celulares androides.

Google, en particular esta muy dedicada a deep learning. En Marzo del 2013, contrato a Geoffrey Hinton, professor de CS de la University of Toronto.

Ese mismo año Facebook contrato a Yann LeCun para dirigir su Applied Machine Learning Group.

El objetivo es extender deep learning a aplicaciones mas alla de reconocimiento de habla y de imagenes lo cual requiere mas desarrollo de software y de poder de procesamiento.

Tipos de Deep Learning arquitecturas

Convolutional Neural Networks (CNNs), que son muy populares para datos de imagenes
Recurrent Neural Networks (RNNs), usadas para aprendizaje de datos secuenciales
incluyendo texto, audio and video.
Multilayer Perceptrons (MLPs), utiles para datos numericos.

Deep Learning usando h2o

Solo hace deep learning de Multilayer Perceptron

En R y Python

```
library(h2o)
```

```
> h2o.init()
```

```
dl_fit3 <- h2o.deeplearning(x = x, y = y, training_frame = data, epochs = 20, hidden=  
c(10,10), seed = 1)
```

```
dl_perf3 <- h2o.performance(model = dl_fit3, newdata = data)
```

Para Convolutional Networks y Recurrent Neural Networks, usar Deep Water o Theano, Tensorflow, Lasagne, Keras, Mxnet. Caffe,

Paquetes en R para deep learning: H2O, KerasR, MxNetR.

Deep Learning usando h2o

H2OBinomialMetrics: deeplearning

MSE: 0.14129

RMSE: 0.37588

LogLoss: 0.42824

Mean Per-Class Error: 0.21228

AUC: 0.87013

Gini: 0.74025

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:

	1	2	Error	Rate
1	381	119	0.238000	=119/500
2	50	218	0.186567	=50/268
Totals	431	337	0.220052	=169/768

Deep Learning usando h2o

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
1	max f1	0.281453	0.720661	229
2	max f2	0.088843	0.825949	322
3	max f0point5	0.544483	0.735849	137
4	max accuracy	0.488910	0.800781	157
5	max precision	0.986791	1.000000	0
6	max recall	0.031608	1.000000	364
7	max specificity	0.986791	1.000000	0
8	max absolute_mcc	0.281453	0.552705	229
9	max min_per_class_accuracy	0.309897	0.782000	218
10	max mean_per_class_accuracy	0.281453	0.787716	229