

Machine Learning

Redes Neuronales y Deep Learning

Dr. Edgar Acuna
Departamento de Matematicas

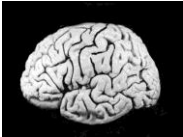

Universidad de Puerto Rico- Mayaguez

academic.uprm.edu/eacuna

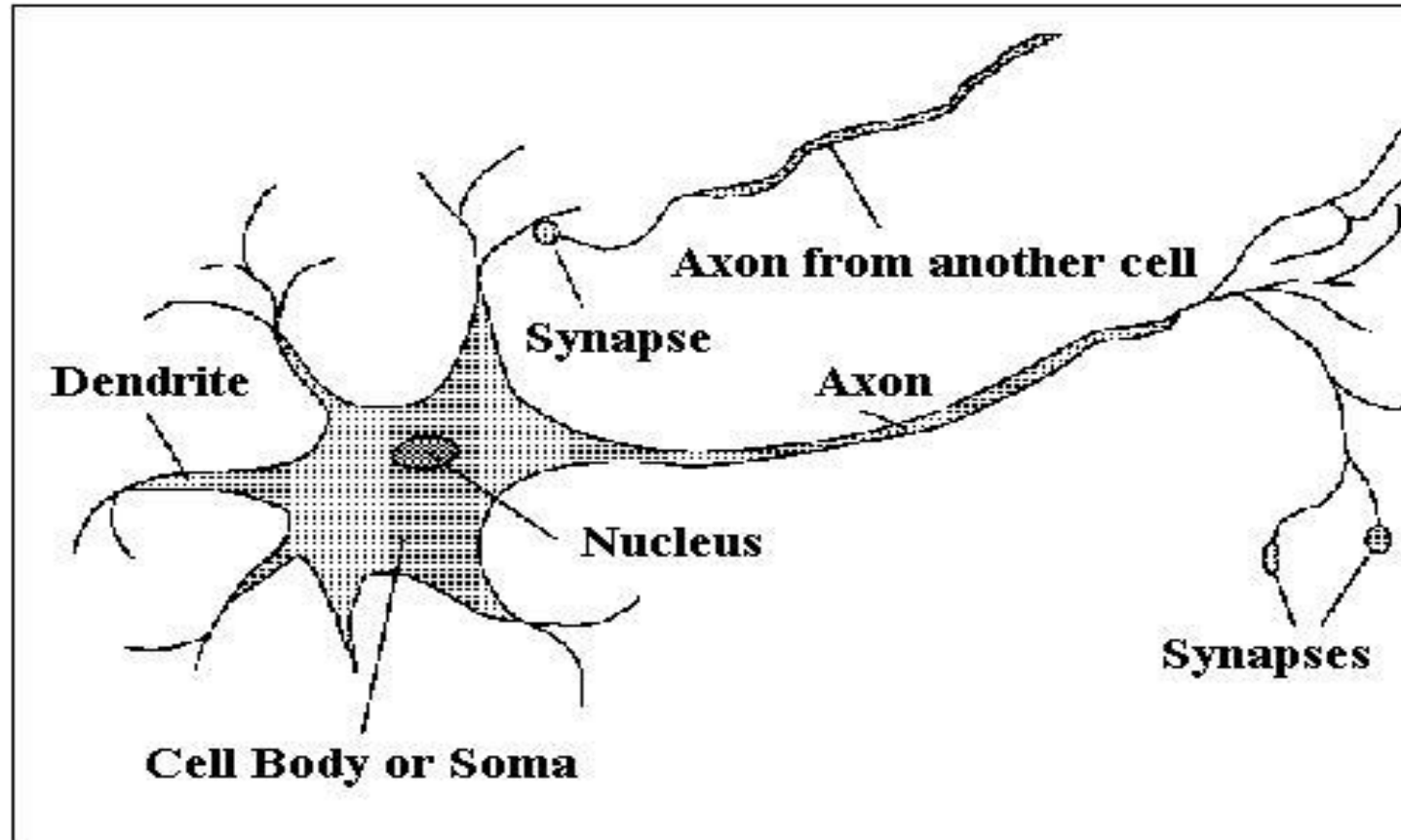
El cerebro tiene aproximadamente 1.5×10^{10} neuronas con un gran numero de conecciones (sinapses) que varia entre 10^3 a 10^4 . La red de neuronas del cerebro forma un sistema de procesamiento de informacion masivamente paralelo.

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.

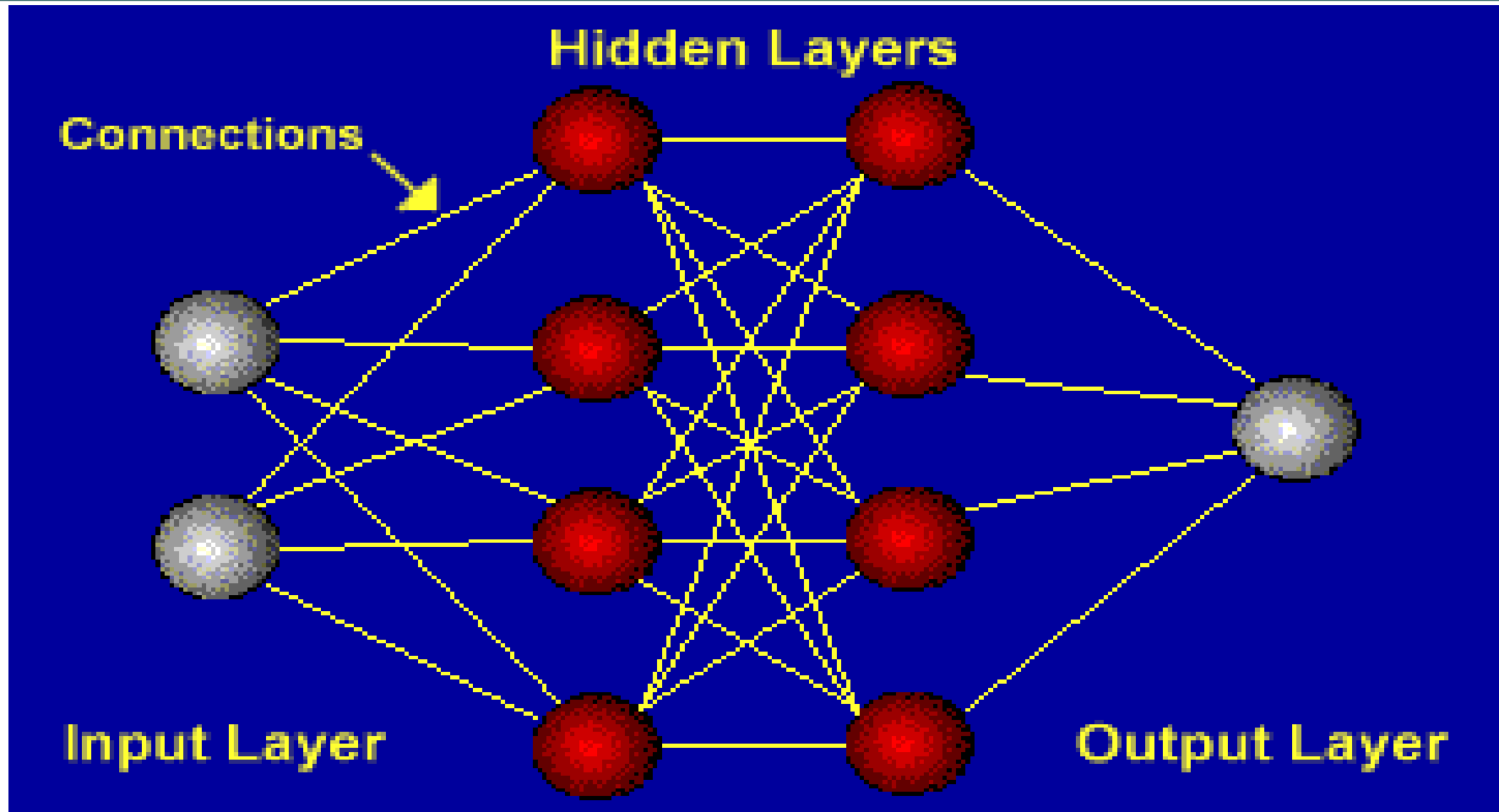
Las primeras ideas fueron introducidas por McCulloch y Pits (1943). Posteriormente, Rosenblatt (1958, 1962) introdujo el concepto de “Perceptron” (conjunto de neuronas) y lo trató de aplicar a clasificación. En 1974, Werbos, publico un primer algoritmo gradiente descendiente para backpropagation. Pero 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 Neuronal



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

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

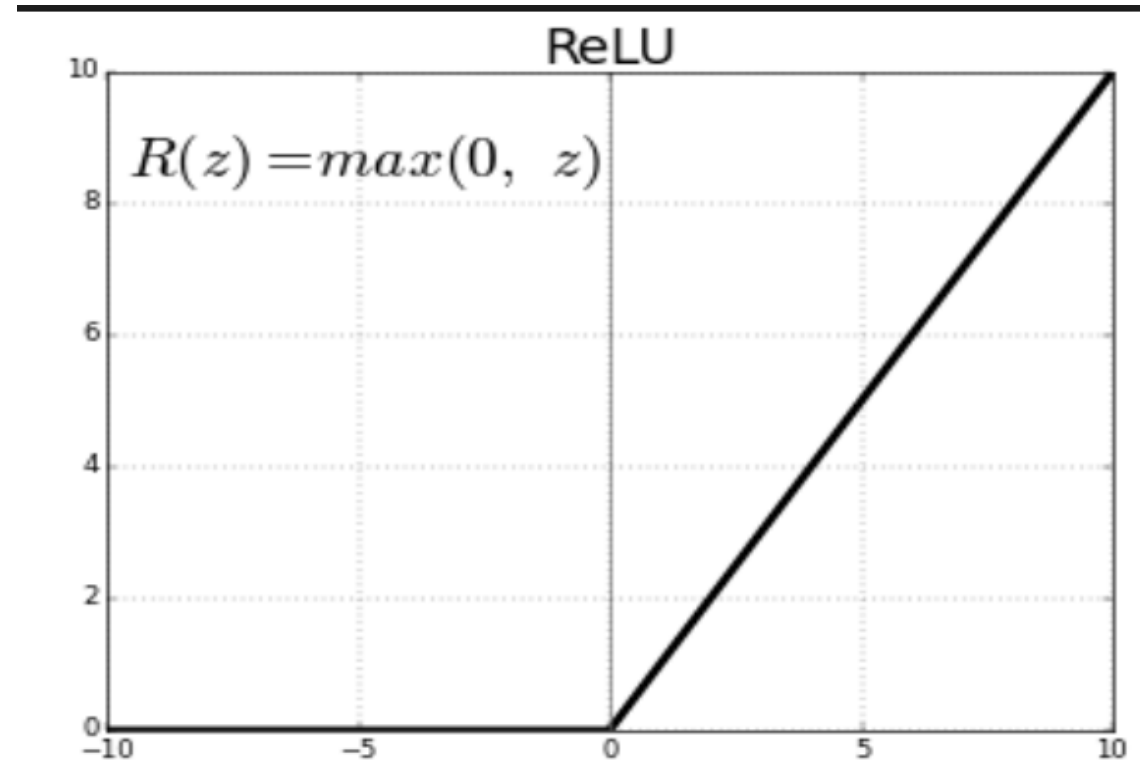
$$y_j = f_j \left(\sum_{inputs: i} w_{ij} x_i \right)$$

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.

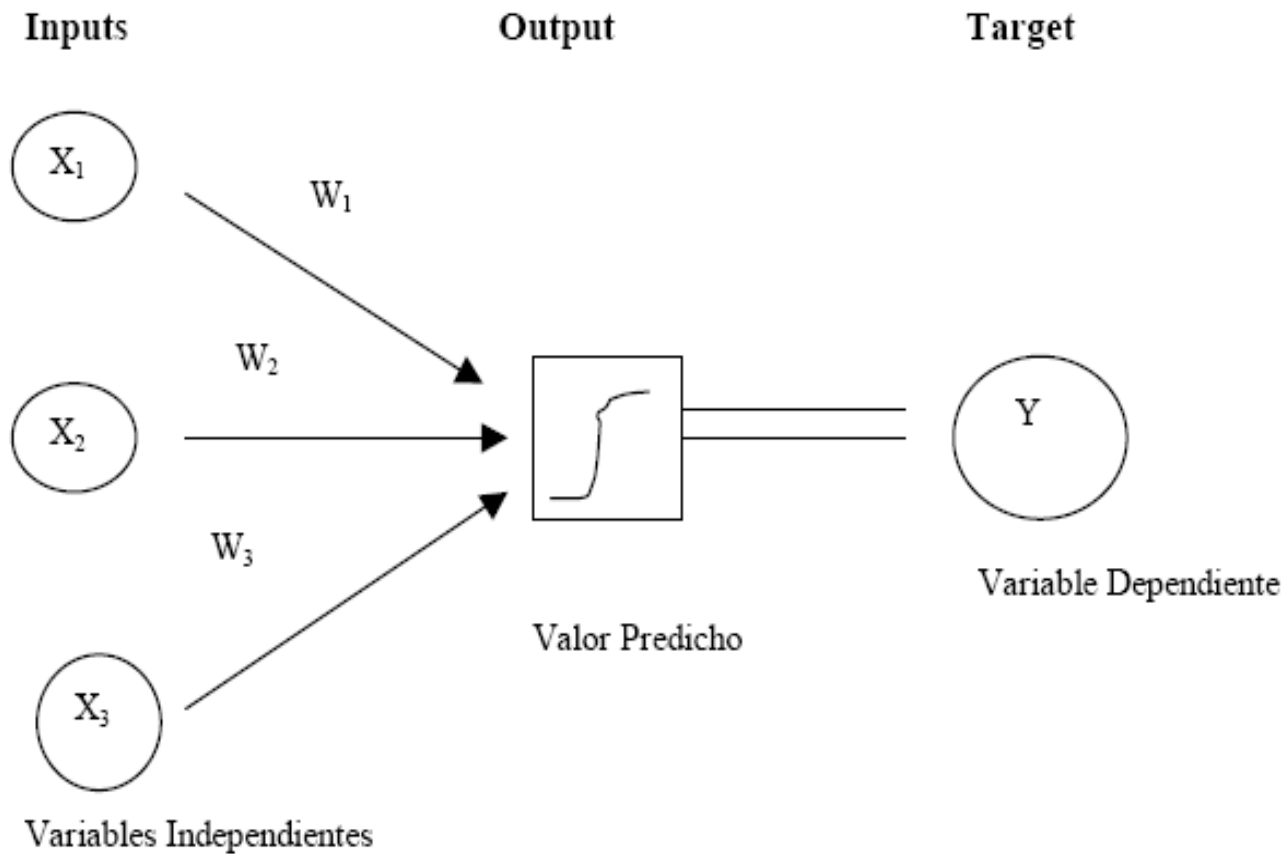
Funciones de Activacion

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

The Relu activation function



Perceptron simple No-lineal=Regression Logistica



El Perceptron de varios niveles (Multilayer Perceptron, MLP)

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

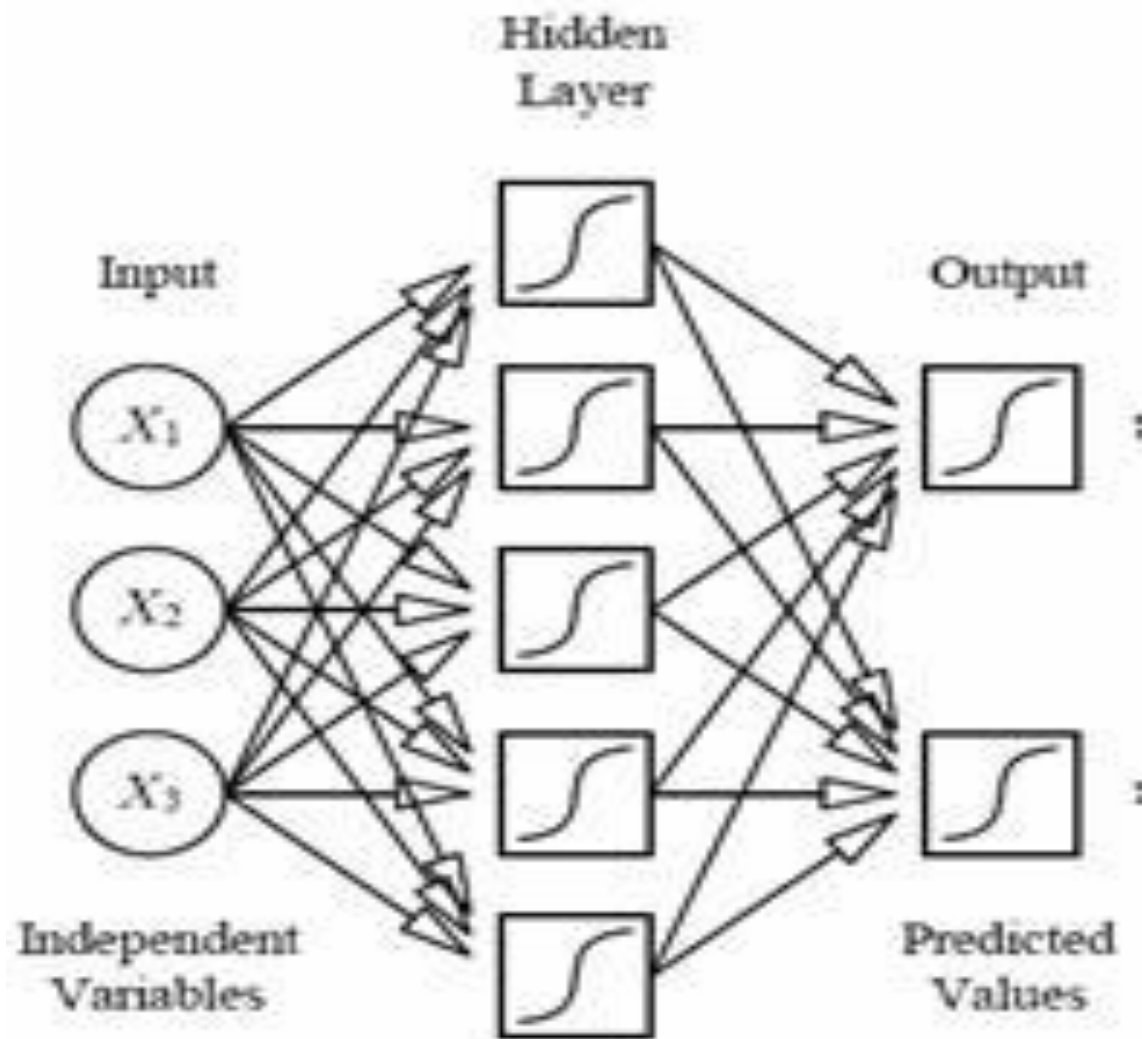
$$z_j = \phi_h(\alpha_j + \sum_i w_{ji} x_i)$$

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

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".

Ejemplo 1. Neural Nets aplicada a la prediccion de la nota final basado en dos examenes Ex1 y Ex2

La function **MLPClassifier** del modulo **scikit-learn** se usa para entrenar una red neuronal MLP .

For instance, to predict the final grade, we use the following commands:

```
df=pd.read_csv("http://academic.uprm.edu/eacuna/eje1dis.csv")
```

```
y=df['Nota']
```

```
X=df.iloc[:,0:2]
```

```
#creating a numerical column "pass" to represent the clases
```

```
lb_make = LabelEncoder()
```

```
df["pass"] = lb_make.fit_transform(df["Nota"])
```

```
y2=df['pass']
```

```
y1=y2.as_matrix()
```

```
X1=X.as_matrix()
```

Example 1 (cont)

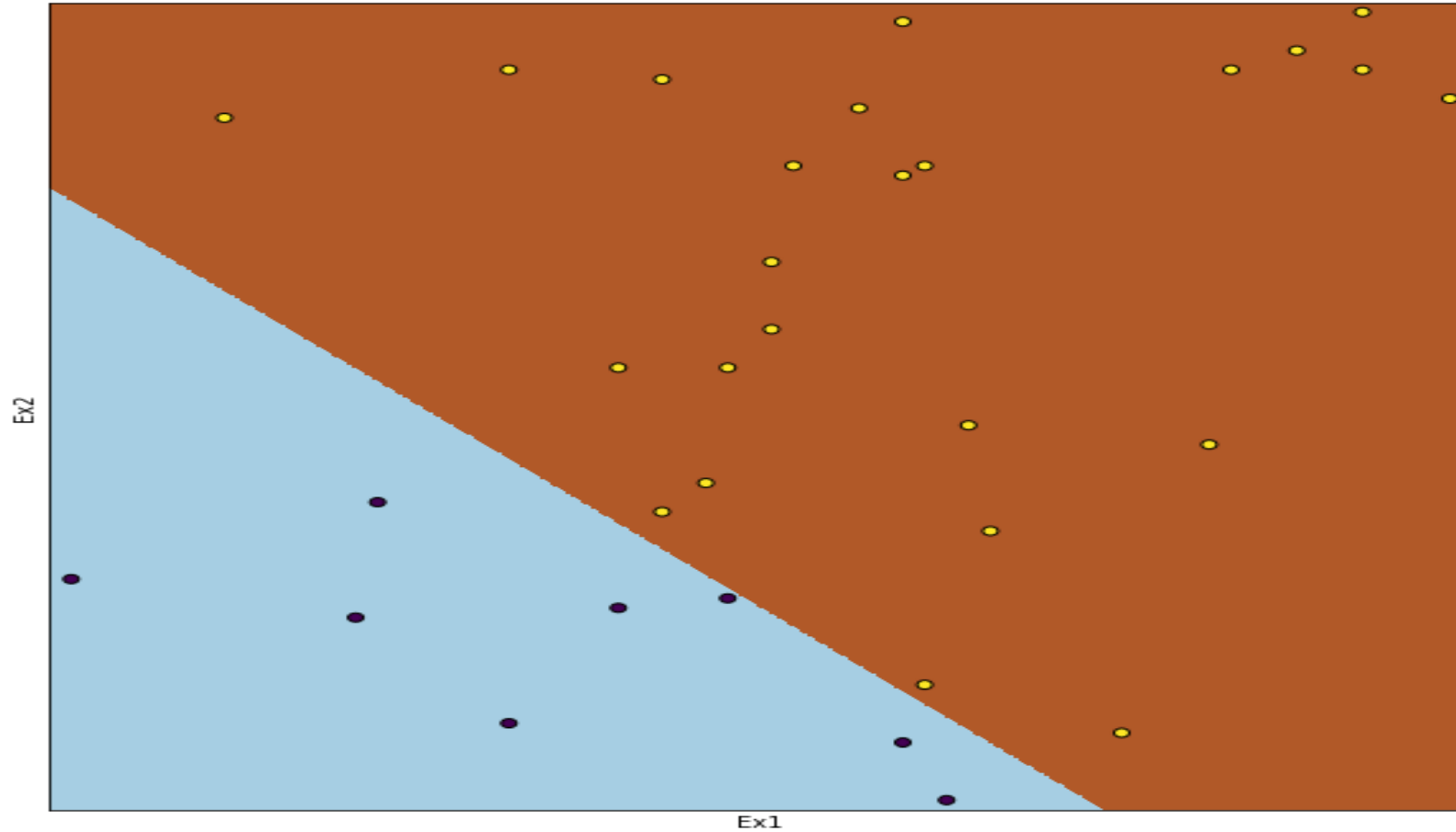
```
#Training a neural net with one hidden layer and five units on it
mlp = MLPClassifier(solver='lbfgs',hidden_layer_sizes=(5),max_iter=1000,random_state=99)
mlp.fit(X1, y1)
#Showing the weights
mlp.coefs_
[array([[ -1.93306393e+01, -2.19847626e-02, 4.67280387e+01, -1.21251200e+01, -
 1.67606515e+01], [ 9.31589043e+01, -3.73206703e-01, 4.15846672e+01, -
 1.23368759e+01, -3.21037745e+00]]), array([[ -7.01419931e-04], [ 9.44058482e-01], [
 1.10475826e-02], [ 1.96857243e+00], [ -1.35496680e+00]])]
#Showing the biases
mlp.intercepts_
[array([-7.44811572, 0.45692388, 1.39966161, -0.16317004, 0.57152039]), array([-
 51.31888278])]
There are 21 parameters in the model
```

Example 1 (cont)

```
#Calculating the posterior probabilities
mlp.predict_proba(X1)
array([[ 0.00000000e+00,  1.00000000e+00],
 [ 2.22044605e-16,  1.00000000e+00],
 [ 0.00000000e+00,  1.00000000e+00],
 [ 2.22044605e-16,  1.00000000e+00],
 [ 3.10862447e-15,  1.00000000e+00],
 [ 6.56805055e-10,  9.99999999e-01],
 [ 1.19992905e-12,  1.00000000e+00],
 [ 4.33475522e-09,  9.99999996e-01],
 [ 2.44449664e-08,  9.99999976e-01],
```

A record of X is assigned to class with the greatest posterior probability

Ejemplo 1: Frontera de Decision usando red neuronal



Ejemplo 2: Diabetes

```
url= "http://academic.uprm.edu/eacuna/diabetes.dat"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pd.read_table(url, names=names)
y=data['class']
X=data.iloc[:,0:8]
y1=y.as_matrix()
X1=X.as_matrix()
#Training a NN with one hidden layer and 20 units
mlp2=MLPClassifier(solver='lbfgs',hidden_layer_sizes=(20),max_iter=1000)
mlp2.fit(X1, y1)
#Estimating the accuracy
mlp2.score(X1, y1)
0.7942708333333337
```


Ejemplo 2(cont)

```
#training a NN with two hidden layer and 20 units in each of them
mlp22=MLPClassifier(solver='lbfgs',hidden_layer_sizes=(20,20),max_iter=5000)
mlp22.fit(X1, y1)
mlp22.score(X1, y1)
0.82682291666666663
#Using Training and Test sets
X_train, X_test, y_train, y_test = train_test_split(X, y)
#training a NN with one hidden layer and 20 units
mlp=MLPClassifier(hidden_layer_sizes=(20),max_iter=500)
mlp.fit(X_train, y_train)
pred=mlp.predict(X_test)
mlp.score(X_test, y_test)
0.64583333333333337
```

Exactitud de MLP estimada por Validacion Cruzada

```
#Estimating the accuracy using cross validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(mlp, X1, y1, cv=10)
print ('The accuracy estimated by CV is:', scores.mean())
```

The accuracy estimated by CV is: 0.690105946685

Example of overfitting with nnet

Let us consider the Diabetes dataset

Number of units in the hidden layer	5	20	50	100	200	500
Accuracy by resubstitution	77.08	79.81	80.46	86.71	99.21	100.0
Accuracy by cross-validation	72.4	74.6	73.04	72..00		

There is high variability on the estimation of the accuracy by resubstitution

Remedial measures for overfitting

Neural Nets tend to overfit the data. That is, NN tend to give accuracy of 100%.

The following are some remedial measures to fix this problem:

a) Stop the iterative process to estimate the minimum of E. It is assumed that a validation set is available and the iterative process is stopped when the performance of the neural net on the validation set begins to deteriorate.

b) Regularization: In this case a penalty is added to the function $E(w)$ and then it is minimized. More specifically,

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

here λ is the regularization constant and ϕ is the penalty function of the model. The most simple regularization method is the one known as weight decay defined by

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

This is very similar to ridge regression in statistics.

Remedial measures for overfitting(cont)

- c) **Average:** In this case several values are chosen as starting values of the iterative process of minimizing $E(w)$ and then the average of the predictions obtained is taken.
- d) **Add noise.** In this case we add noise to each input variable and then a NN is fitted. The process is repeated several times and then the predictions obtained are averaged.
- e) **Bagging** (“Bootstrap aggregating”). Several samples with replacement and of the same size are taken from the training sample. A NN is fitted for each of these samples. Finally, each instance of the training sample is assigned to the most voted class.

Effect of the use of decay

Let us consider the Diabetes dataset. The parameter alpha of the MLPClassifier determines the decay.

Decay	0	5	10	.1	.5
Error aparente	80.59	83.2	83.72	80.85	80.46

A large decay gives better accuracy

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.

Limitations of Neural Networks

Random initialization + densely connected networks lead to:

- High cost
 - Each neuron in the neural network can be considered as a logistic regression.
 - Training the entire neural network is to train all the interconnected logistic regressions.
- Difficult to train as the number of hidden layers increases
 - Recall that logistic regression is trained by gradient descent.
 - In backpropagation, gradient is progressively getting more dilute. That is, below top layers, the correction signal δ_n is minimal.
- Stuck in local optima
 - The objective function of the neural network is usually not convex.
 - The random initialization does not guarantee starting from the proximity of global optima.
- Solution:
 - Deep Learning/Learning multiple levels of representation

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 cualquier otro metodo 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, 2015), TensorFlow (Google Brain, 2015 coore en Python y R).

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

Only performs deep learning using MLP

```
> 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)
```

For Convolutional Networks and Recurrent Neural Networks, use Deep Water, Theano, Tensorflow, Lasagne, Keras, Mxnet. Caffe,

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

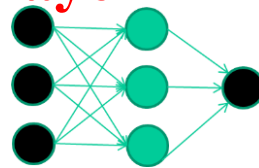
	metric	threshold	value	idx
1	max f1	0.281453	0.720661	229
2	max f2	0.088843	0.825949	322
3	max f0point5	0.544483	0.735849	137
4	max accuracy	0.488910	0.800781	157
5	max precision	0.986791	1.000000	0
6	max recall	0.031608	1.000000	364
7	max specificity	0.986791	1.000000	0
8	max absolute_mcc	0.281453	0.552705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

Deep Learning usando h2o

```
library(h2o)
> h2o.init(nthreads = -1, #Number of threads -1 means use all cores on your machine
+         max_mem_size = "8G") #max mem size is the maximum memory to allocate to
H2O
dl_fit2 <- h2o.deeplearning(x = x, y = y, training_frame = train,model_id = "dl_fit2",
epochs = 20, hidden= c(10,10),seed = 1)
dl_perf2 <- h2o.performance(model = dl_fit2,newdata = test)
```

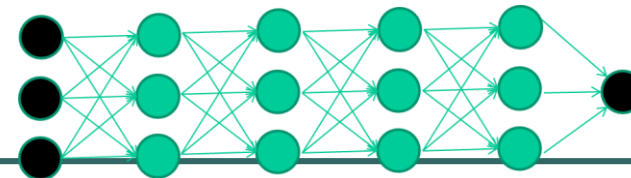

Cual es lo nuevo en Deep Learning con respecto a NN?

we have always had good algorithms for learning the weights in networks with 1 hidden layer

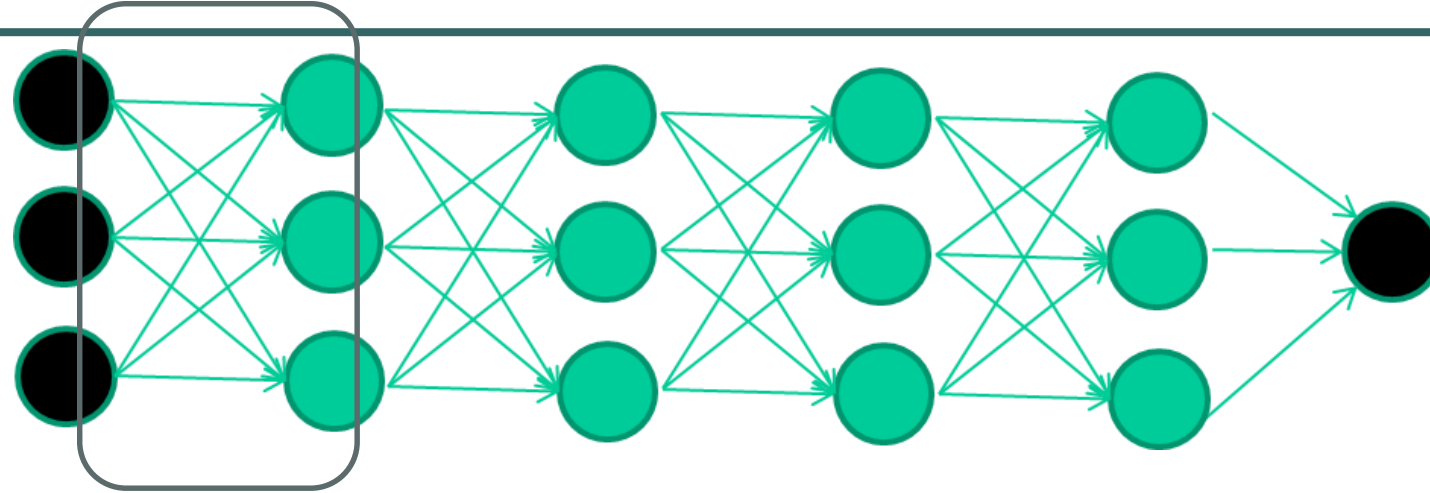


but these algorithms are not good at learning the weights for networks with more hidden layers

what's new is: algorithms for training many-layer networks

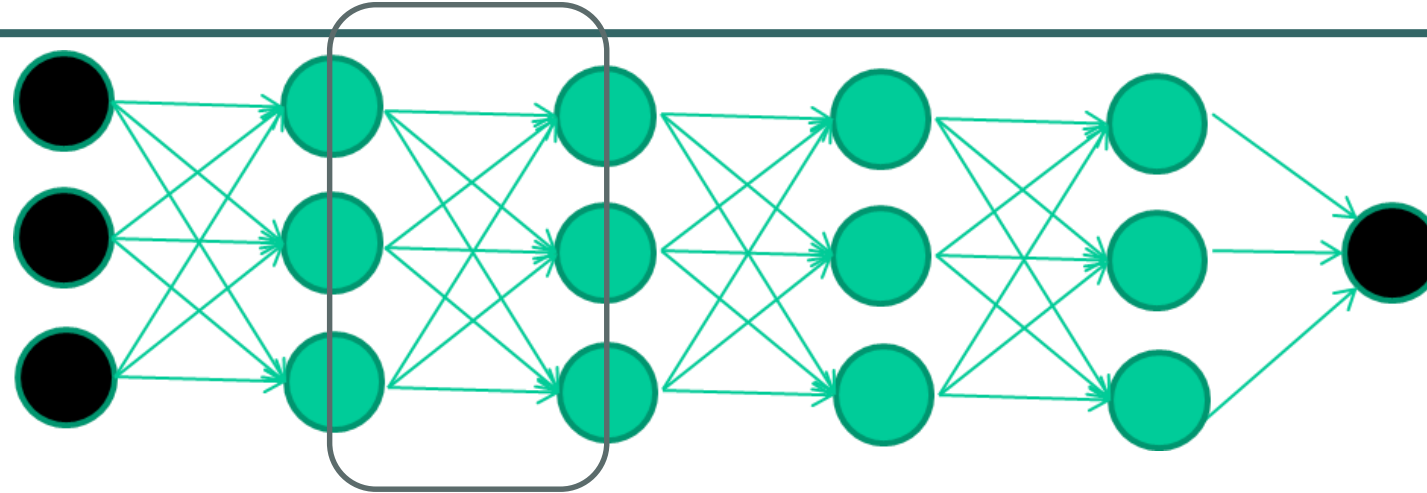


La nueva manera de entrenar NN de varias capas



**Entrenar esta
capa primero**

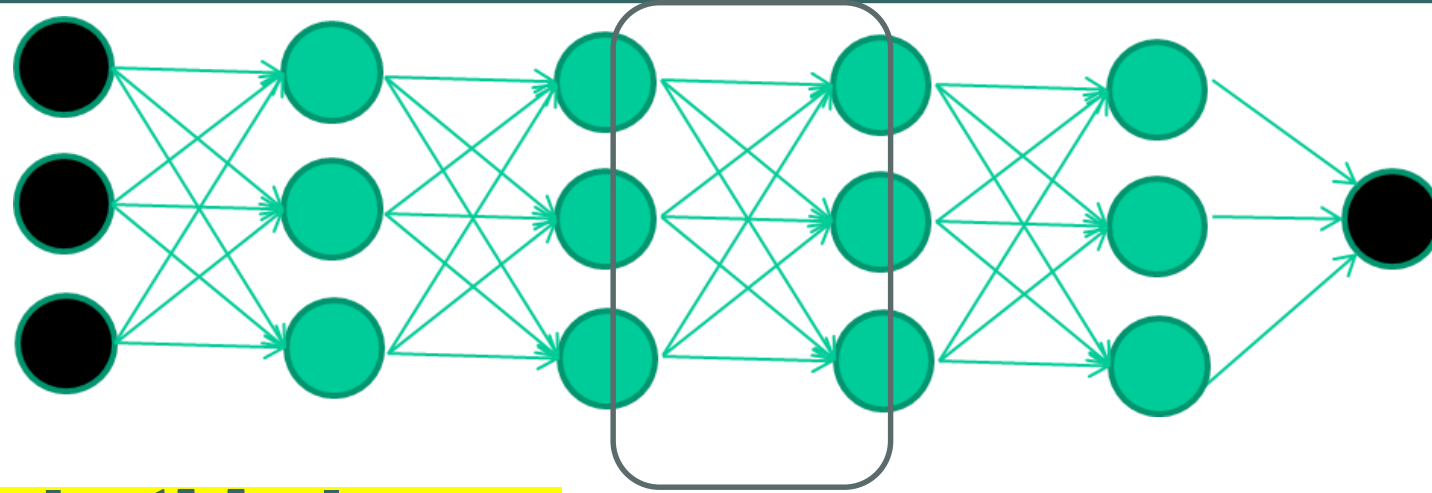
La nueva manera de entrenar NN de varias capas



Entrenara esta

**capa
Luego entrenra
esta capa**

The new way to train multi-layer NNs...

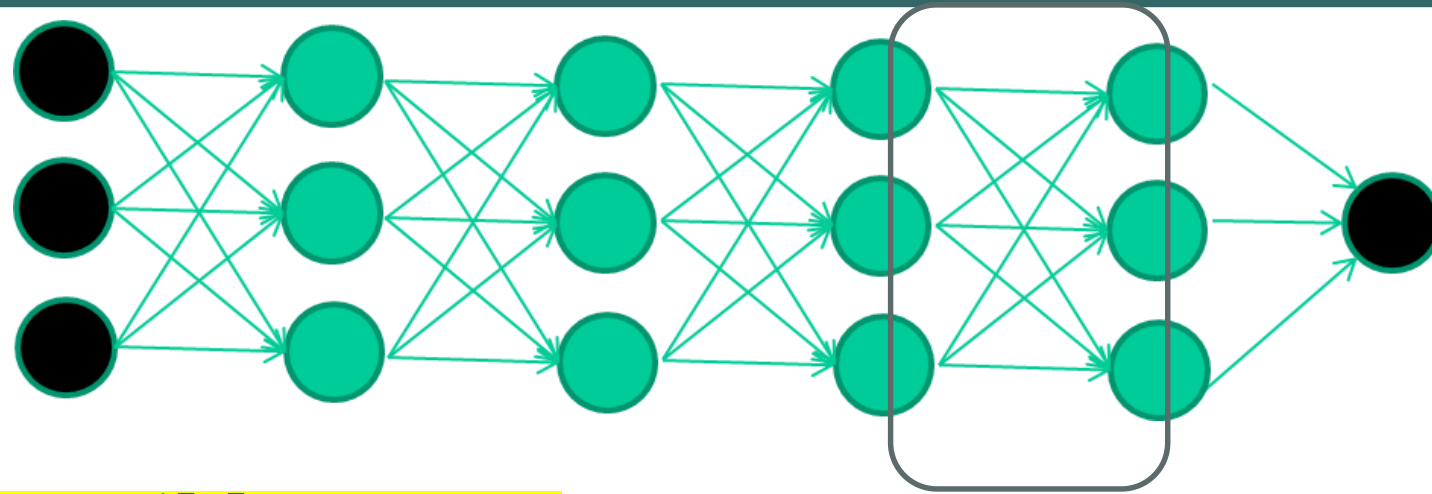


**Train this layer
first**

then this layer

then this layer

The new way to train multi-layer NNs...



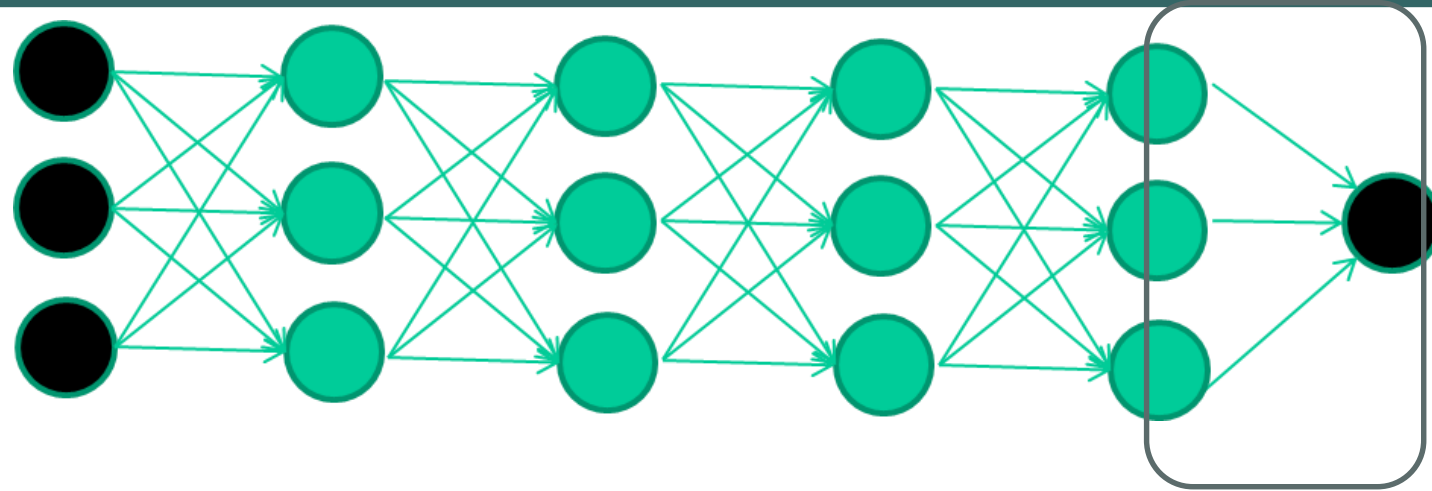
**Train this layer
first**

then this layer

then this layer

then this layer

The new way to train multi-layer NNs...



**Train this layer
first**

then this layer

then this layer

then this layer

finally this layer

Data Mining and Machine Learning

Outlier Detection (Novelty Detection)
using autoencoders

Dr. Edgar Acuna
Departamento de Matematicas

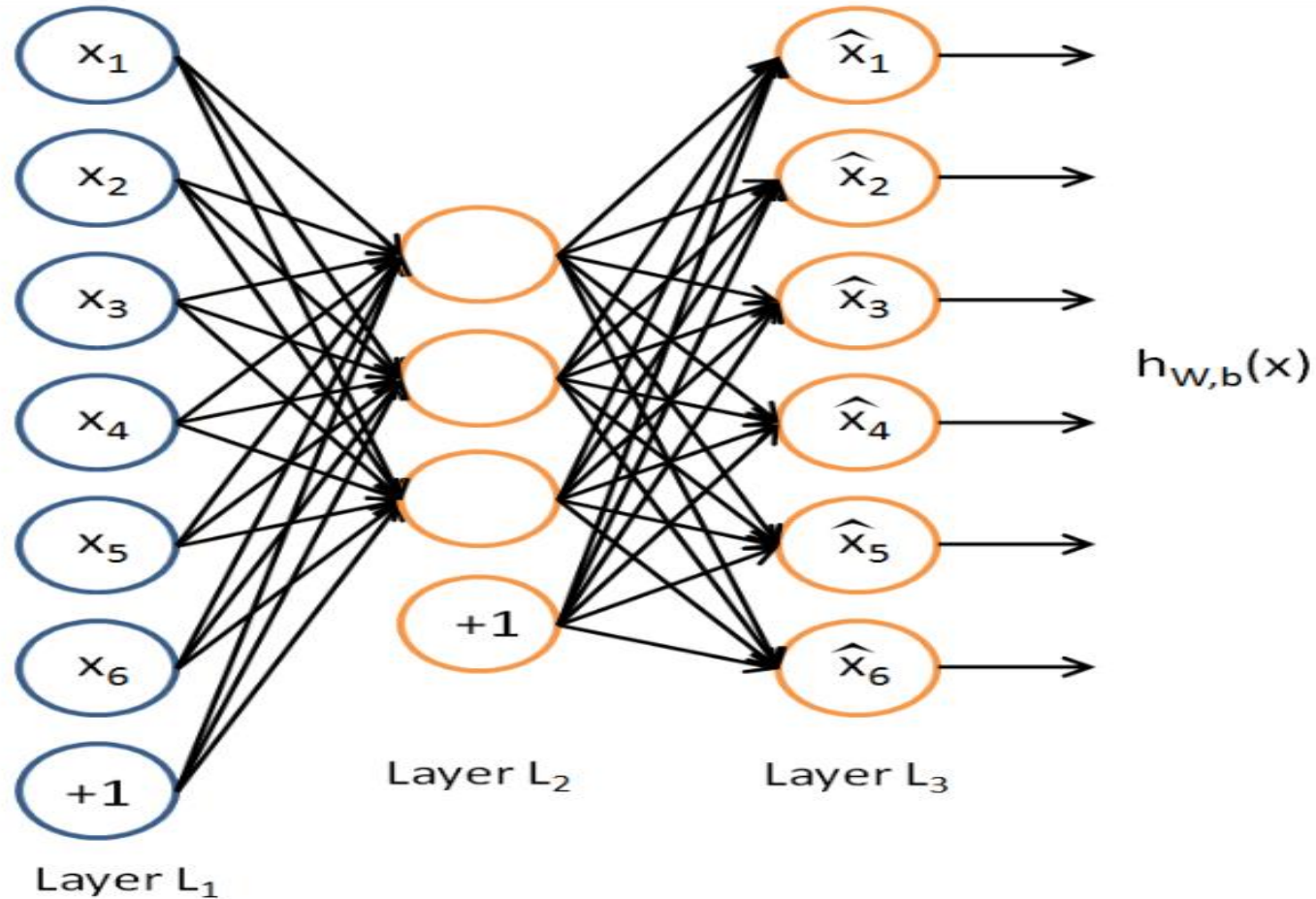
Universidad de Puerto Rico- Mayaguez

academic.uprm.edu/eacuna

Autoencoders

- Autoencoders are:
- feed Forward Neural Network models for unsupervised tasks (No Labels).
- Applies backpropagation, setting the target values to be equal to the inputs.
- simple to understand!

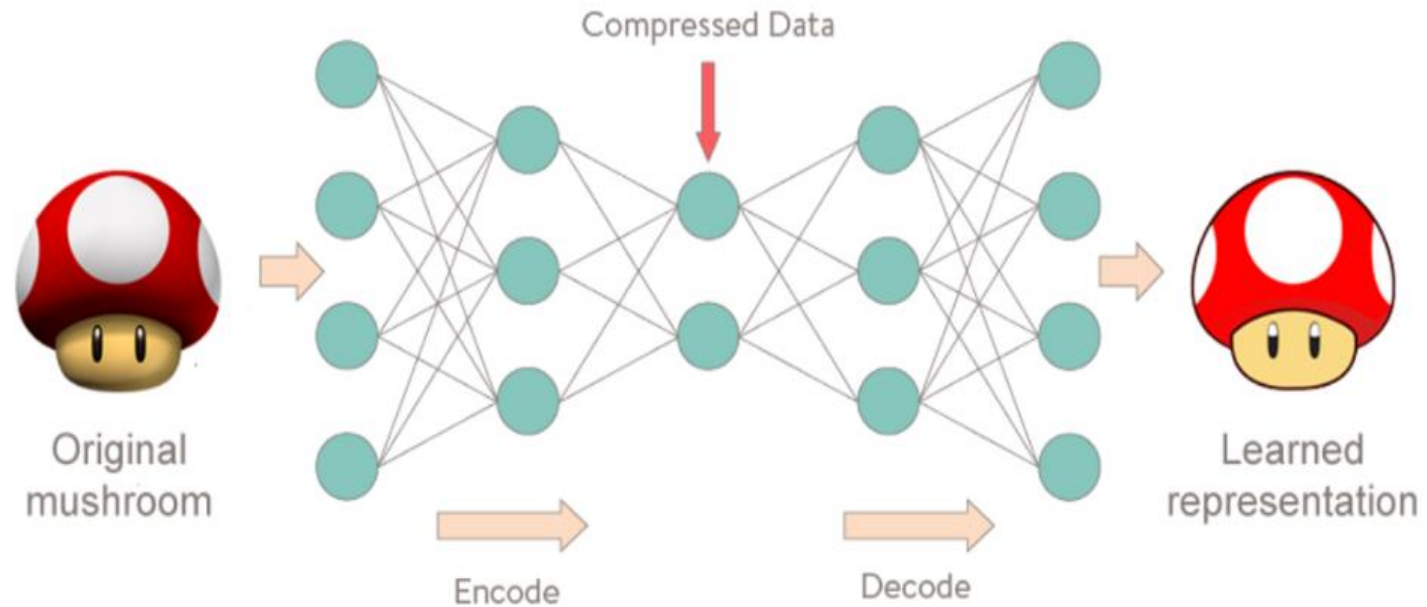
Example of autoencoder



Autoencoders are used for:
compressing data and learn some features, especially in non-structured data(e.g images).

anomaly (outlier) detections (Fraud detection is one area of application).

Autoencoder with 5 layers



The autoencoder model tries to minimize the reconstruction error (RE), which is the mean squared distance between input and output.

$$L(x, x') \approx ||x - x'||^2$$

Reconstruction Error

The model will train on the normal dataset by minimizing the RE. It is expected that the RE to be relatively high when it is tested on a new abnormal datapoint (outlier)

However one has to do tuning on the RE's threshold to detect outliers

Detecting outliers for Diabetes using autoencoders

The neural network model is built by Keras. The optimized Autoencoder model has 3 fully connected hidden layers, which has 4,2,4 neurons respectively. The input and out layers have 8 neurons each for the 8 features.

I used the tangent hyperbolic (tanh) activation function in the first and third layer and the relu activation function in the second and fourth layer. I train the model for 20 epochs and used a batch size of 50, and 10 % of the training data is used for validation during the training process.

Outliers detected on Diabetes using autoencoders

Using all the data

4, 12, 13, 39, 43, 45, 86, 88, 159, 177, 186, 193, 215,
228, 247, 259, 270, 298, 357, 362, 370, 375, 445, 453,
458, 459, 487, 542, 558, 579, 590, 674, 691, 740, 744, 763

Using train and test datasets

445, 428, 123, 228, 274, 660, 517, 323, 339, 100, 588, 708,
691, 582, 614, 84, 24, 459, 509, 319, 12, 612, 672, 298