Machine Learning

Neural Networks and Deep Learning

Dr. Edgar Acuna
Department of Mathematical Sciences

Universidad de Puerto Rico- Mayaguez

academic.uprm.edu/eacuna

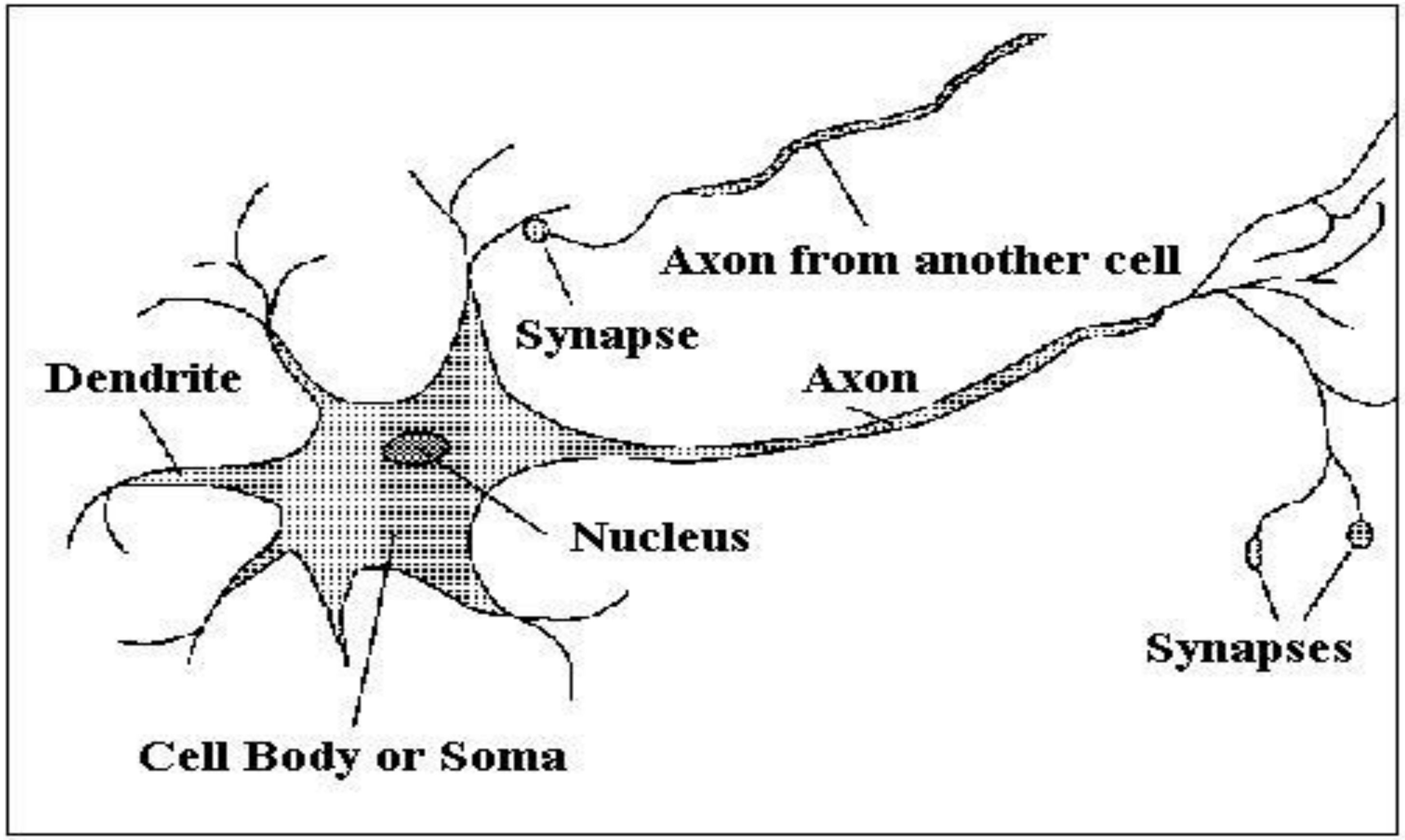
Introduction

The human brain has approximately 1.5 x 10^{10} neurons with a large number of connexions among them called synapses. The number of synapses connecting two neurons varies from 10 to 10^4 . The network of neurons form a system of information processing which is massively parallel.

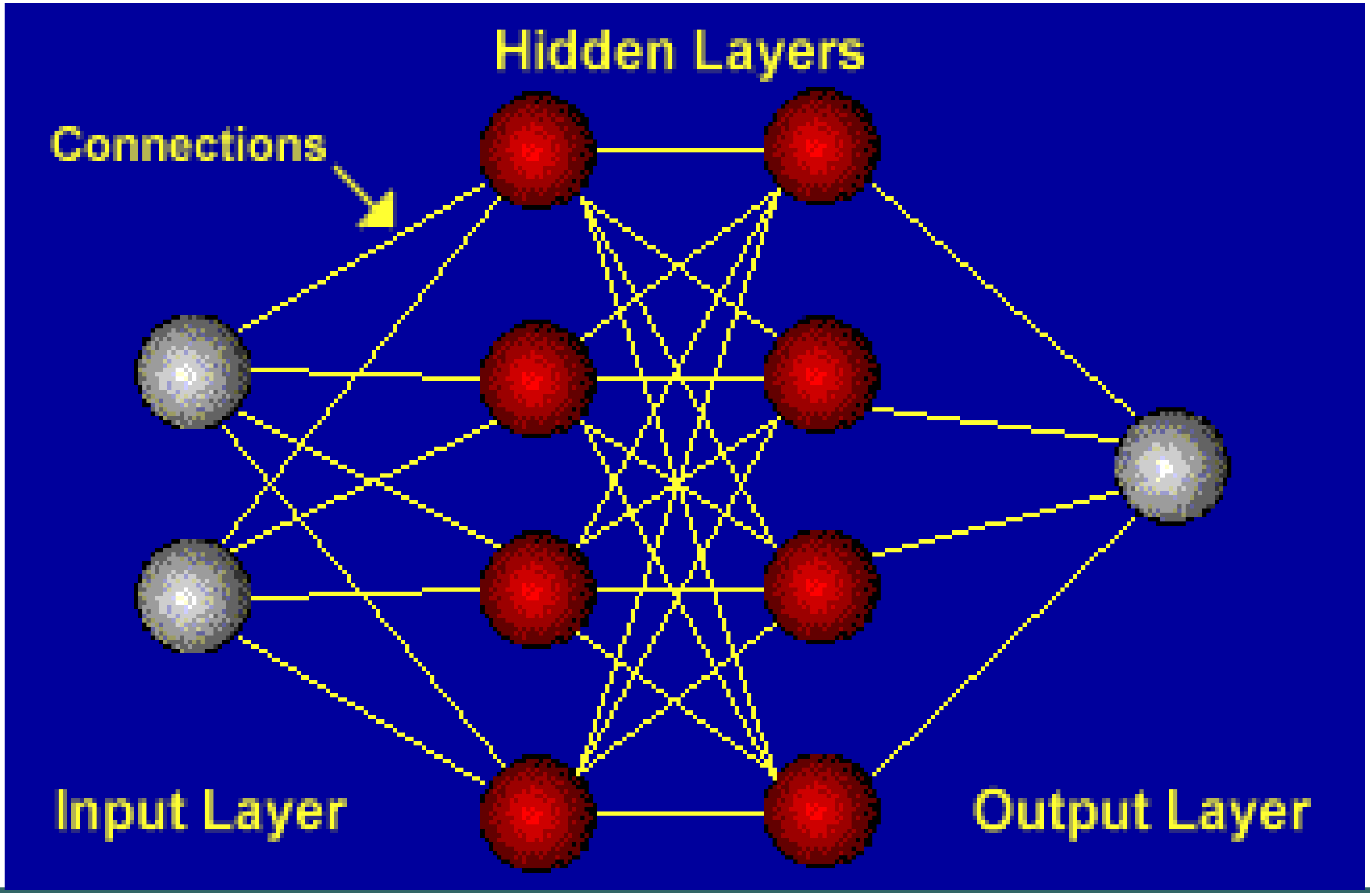
Neural Networks can be considered as an intent to emulate the human brain. In general a neural nets are representations of mathematical models where computational units are connected among them using a nechanism that learns from experience. That is, from the data was recollected.

The first ideas about Neural nets were introduced by McCulloch and Pitts (1943). Later, Rosenblatt (1958, 1962) introduced the concept of “Perceptron” (set of neurons) and he tried to apply them to classification. In 1974, Werbos, published a first gradient descendent algorithm for backpropagation. But, it was until 1986 when, Hinton, Rumellhart and Williams presented the Backpropagation algorithm for learning a neural nets that these started to be used in great demand.

A neuron


Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell Body or Soma

A neural network



The Human Brain vs. Computers

	processing elements	**element size**	**energy use**	**processing speed**	**style of computation**	learns	Intelligent, consciuous
	10^{15} synapses	$2x10^{-8}$ m	30 W	1000 Hz	parallel, distributed	yes	usually
	$2x10^{10}$ transistors	10^{-8} m	130 W (Core I7)	$3x10^{9}$ Hz	serial, centralized	It is trying	Not(yet)

Estimates are that computers will surpass the capability of human brains around the year 2040.

In Statistics, the research work done by Ripley (1993) and, Chen and Titterigton (1994) were fundamental to attract statisticians into neural network.

Artificial Neural Nets (ANN) have plenty of applications. Among them;

- a. Classification
- b. Regression
- c. Clustering
- d. Outlier Detection
- e. Density function estimation

Comparing Statistics and Neural Nets terms

Statistics	Neural networks
Independient variables	Inputs
Dependent values	Targets
Predicted values	Outputs
Estimacion, Ajuste	Learning, training
Parameters	Weights
Transformations	Functional Links
Outlier Detection	Novelty detection

Type of neural nets

a) For supervised Learning (Linear regression and classification)

- Multilayer Perceptron (MLP)
- Radial basis function Networks (RBF)
- Learning Vector Quantization (LVQ)

b) For unsupervised Learning (Clustering)

- Hopfield Networks
- Kohonen's Self-Organizing Maps
- Adaptive Resonance Theory

The single-layer perceptron

A perceptron is a single layer neural network. A perceptron calculates a linear combination of imputs adding also an intercept called Bias. Then, an ***activation function,*** which in general is nonlinear is applied to the linear combination to generate a output. Thus, the output y_j is given by

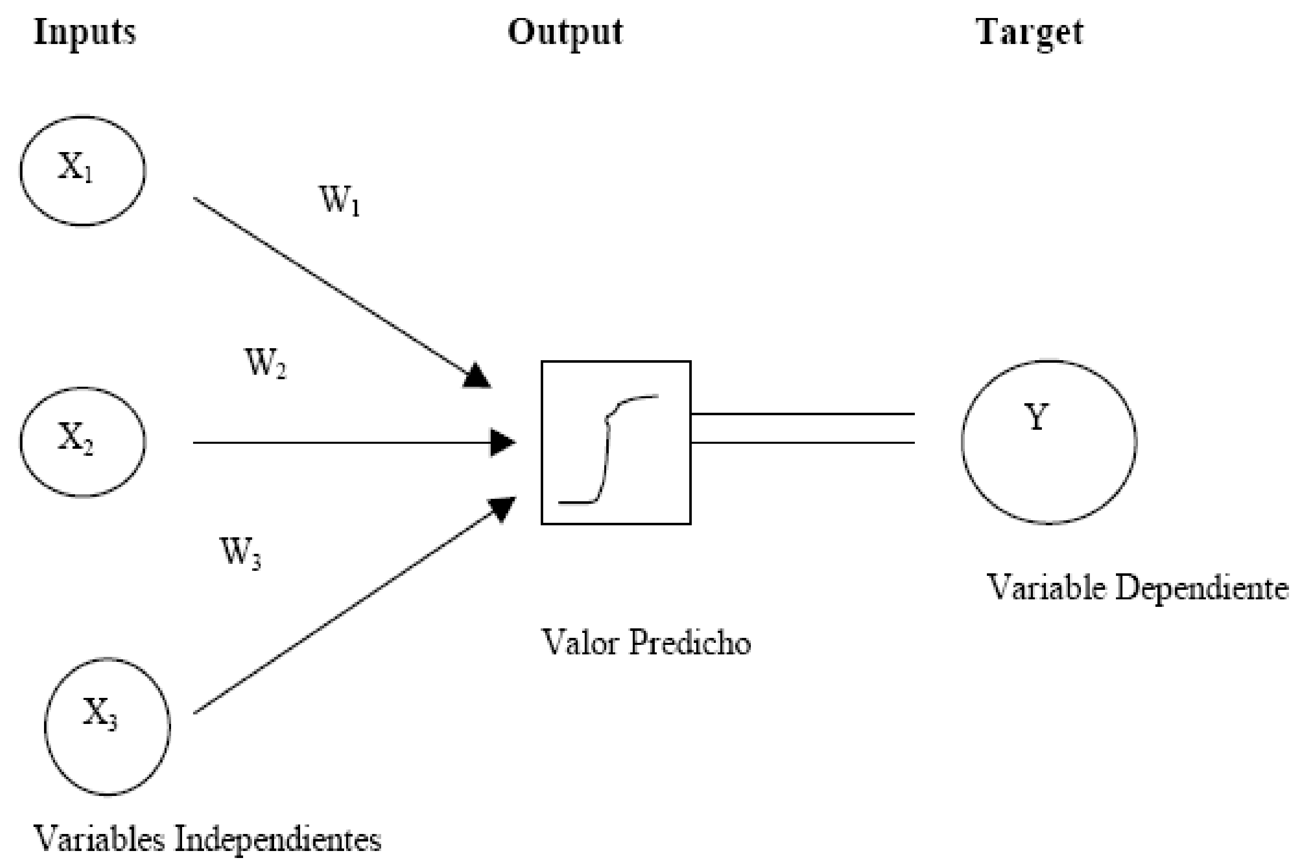
$$y_j = f_j\Big(\sum_{imputs:i} w_{ij} x_i\Big)$$

f_j represents the activation function and w_{ij} are the weights. The neural net learns the weights from the recollected data.

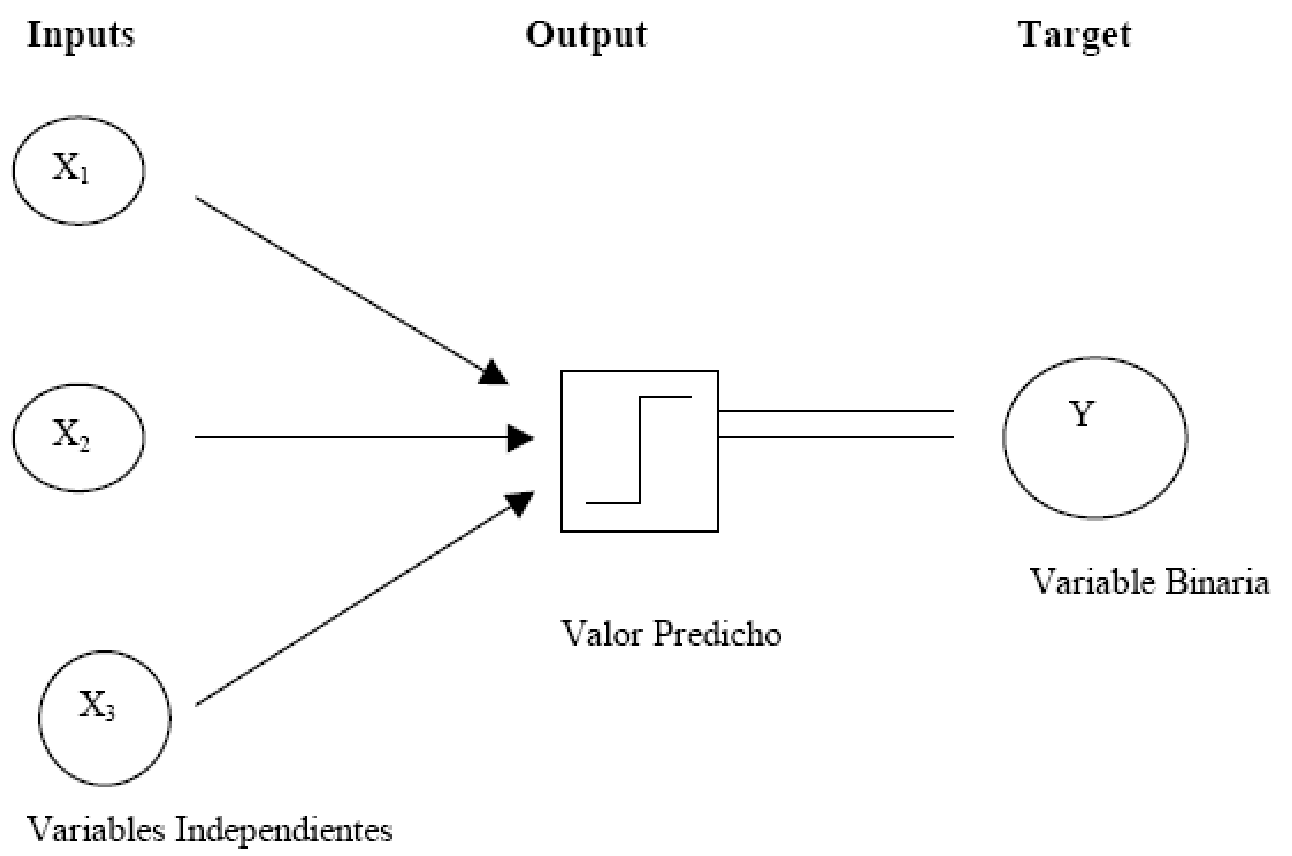
Activation Functions

Name	Function
Linear	A(x)=x
Logistic (Sigmoid)	$A(x)=(1+e^{-x})^{-1}$
Gaussian	$A(x)=\exp-x^2/2)$
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
Tanh	A (x)=2*logistic(2x)-1

Single Perceptron= Logistic Regression



Perceptron for two-class classification



The Multilayer Perceptron, MLP

A neural net of two layers can be written as a pair of equations

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

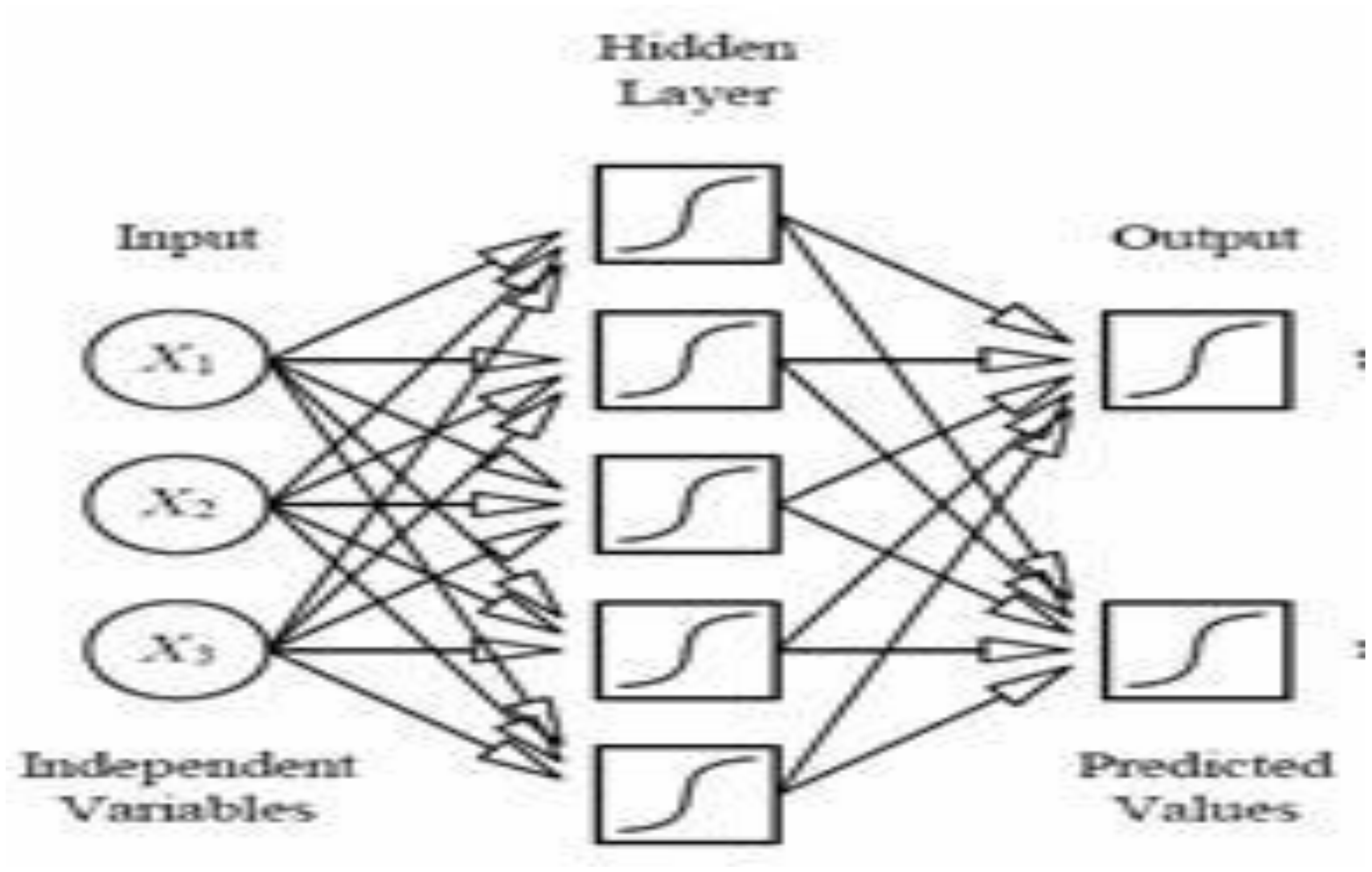
$$y_k = \phi_o\ (\alpha_k + \sum_j w_{jk} z_j)$$

Here z_j are the units in the hidden layer, y_k are the units in the output layer and ϕ_o and ϕ_h are activation functions. Usually ϕ_h is the logistic function, also known as the sigmoidal activation function. On the hand, ϕ_o can be either linear, logistic or threshold function. Replacing the first equation in the second equation, we obtain

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

where the weights w_{ij} and the biases α_j must be estimated using the data

A MLP Neural Net



The Multilayer Perceptron, MLP(cont)

In classification problems with more than two classes,say C classes, the softmax function is used as an output function. The softmax function, ϕ_o, is defined as

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

where

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

The MLP also is called a Feed Forward Neural Network (FFNN) or a Backpropagation net .

When a neural network has more than one hidden layers is called a "deep" Neural network.

Training of a Neural Net

- This is equivalent to model estimation in the statistics world. The weights w's of the NN are chosen in such way that some fit measure is minimized.

- **Regression:** The sum squares of errors given by

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

Is minimized with respect to the weights w=(α_j, w_{ij})

Training of a neural network(cont)

- **Classification**: In two-classes problems, the cross entropy given by

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

Is minimized with respect the vector of weights w. If the instance $\mathbf{x}_i$ belongs to class 1 then $y_i = 1$ and it is equal to 0 otherwise. Las $\hat{y}_i$ are estimations of the posterior probability to lie in the class 1 when $\mathbf{x}_i$ is observed.
This is equivalent to parameter estimation in a binary logistic regression.

Training of a neural network(cont)

For J classes the entropy function given by

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

must be minimized. E can be minimized using methods from numerical analysis and nonlinear models including:

Gradient Descent, Quasi-Newton methods (recommended if the number of weights is less than 1000), Gradient Conjugated method (recommended if there is a large amount of weights to be estimated), Simulated Annealing, Particle Swarm Optimization and Genetic Algorithms.

The greatest challenge to minimize E is the presence of multiple local minimum and there is a risk to choose one that is not the optimum. Frequently, it is necessary to restart the minimization process using different starting values for the iterative process.

The Backpropagation algorithm is used to compute the gradients.

Example 1. Neural Nets applied to the prediction of the final grade based on Ex1 and Ex2

The **MLPClassifier** function from the **scikit-learn** library allow us to perform training of a MLP neural net.

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])]
```

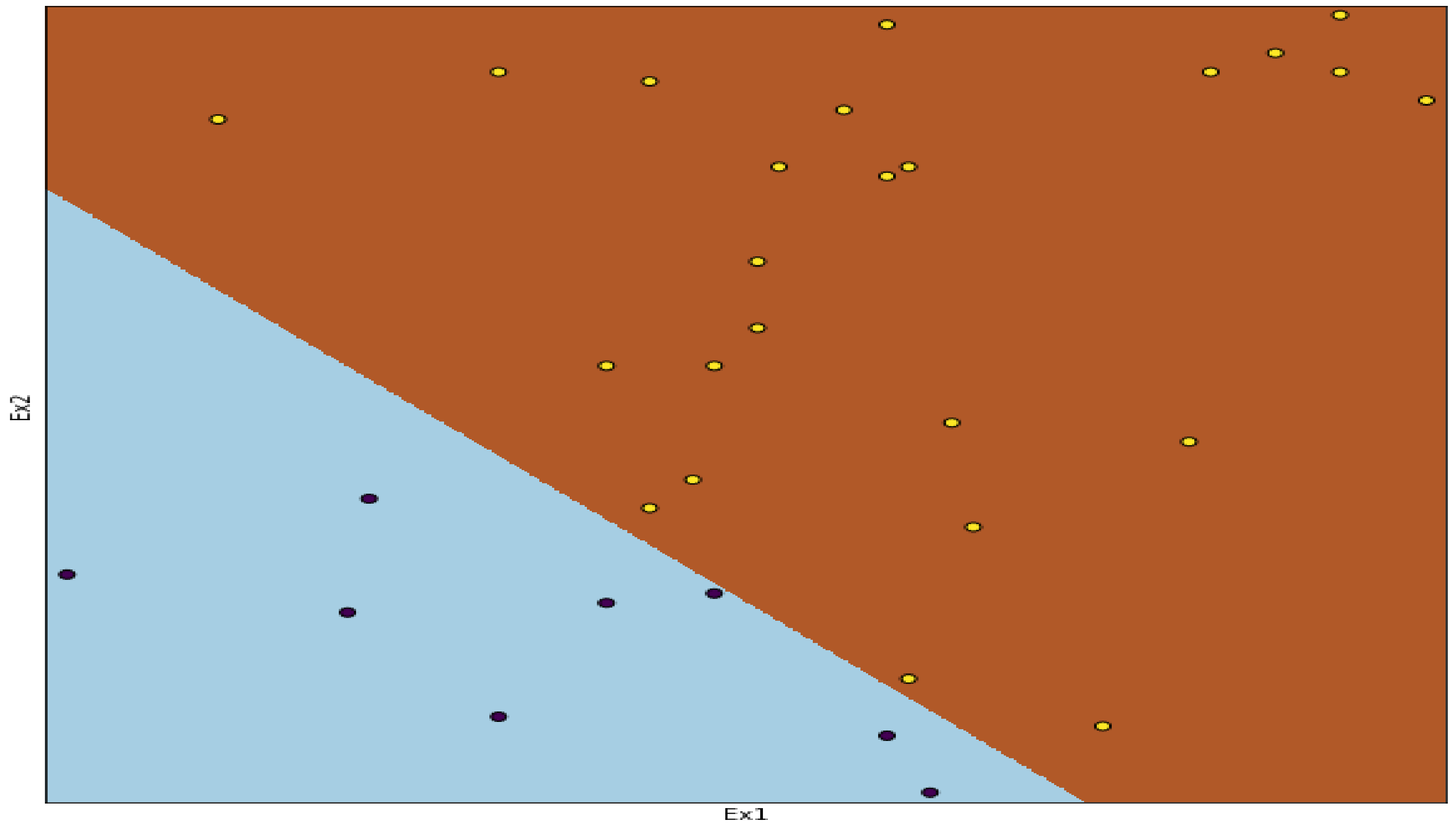
There are 21 parameters in the model

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

Example 1: Decision Boundary using Neural Nets



Example 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)
0.79427083333333337
```

Example 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)
0.8268229166666663
#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)
0.6458333333333337
```

Accuracy of MLP estimated by CV

```
 #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 resubtitution	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 penality is added to the function E(w) and then it is minimized. More specifically,

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

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

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

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 imput 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 taking 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 he use of decay

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

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

A large decay gives better accuracy

Radial basis functions Nets

Initially, the radial basis functions nets (RBF nets) were applied to approximate functions.

Roomhead and Lowe (1988) were the first researchers to ue RBF nets in classification. Mathematically, the RBF nets can be written as a linear combination of nonlinear radially symmetric functions. That is:

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

where c_j are pre-specified centers, α and β_j are weights to be estimated, the ϕ_j 's are the basis functions, usually they are the same for all j . The most used functions are the gaussian $\phi(r)=\exp(-r^2/2\sigma)$, the quadratic $\phi(r)= r^2+ar +b$ and the thin plate spline $\phi(r)=r^2\log(r)$. Here r represents radius.

Radial basis functions Nets(cont)

RBF nets are very similar to density estimation using Gaussian mixtures or kernel methods.

Most of the time, the centers of the model, are the centroids determined for any clustering algorithm, such as k-means.

The number of components, M, is chosen trying to avoid either underfitting or overfitting.

In terms of classification, RBF nets are used to estimated the posterior probability of each class, $P(C_j/x)$ and assign then the object x to the class j with the maximum posterior probability.

In scikit-learn, the function sklearn.gaussian_process.kernels.RBF trains a RBF net.

Radial basis functions Nets(cont)

- The training of RBF net is faster than the training MLP net. Also , an RBF is easier to be interpreted. However the MLP nets is much faster for making predictions.
- RBF nets are more sensitive than MLP nets to the curse of dimensionality problem. Therefore, it does not perform well with a large number of predictors.

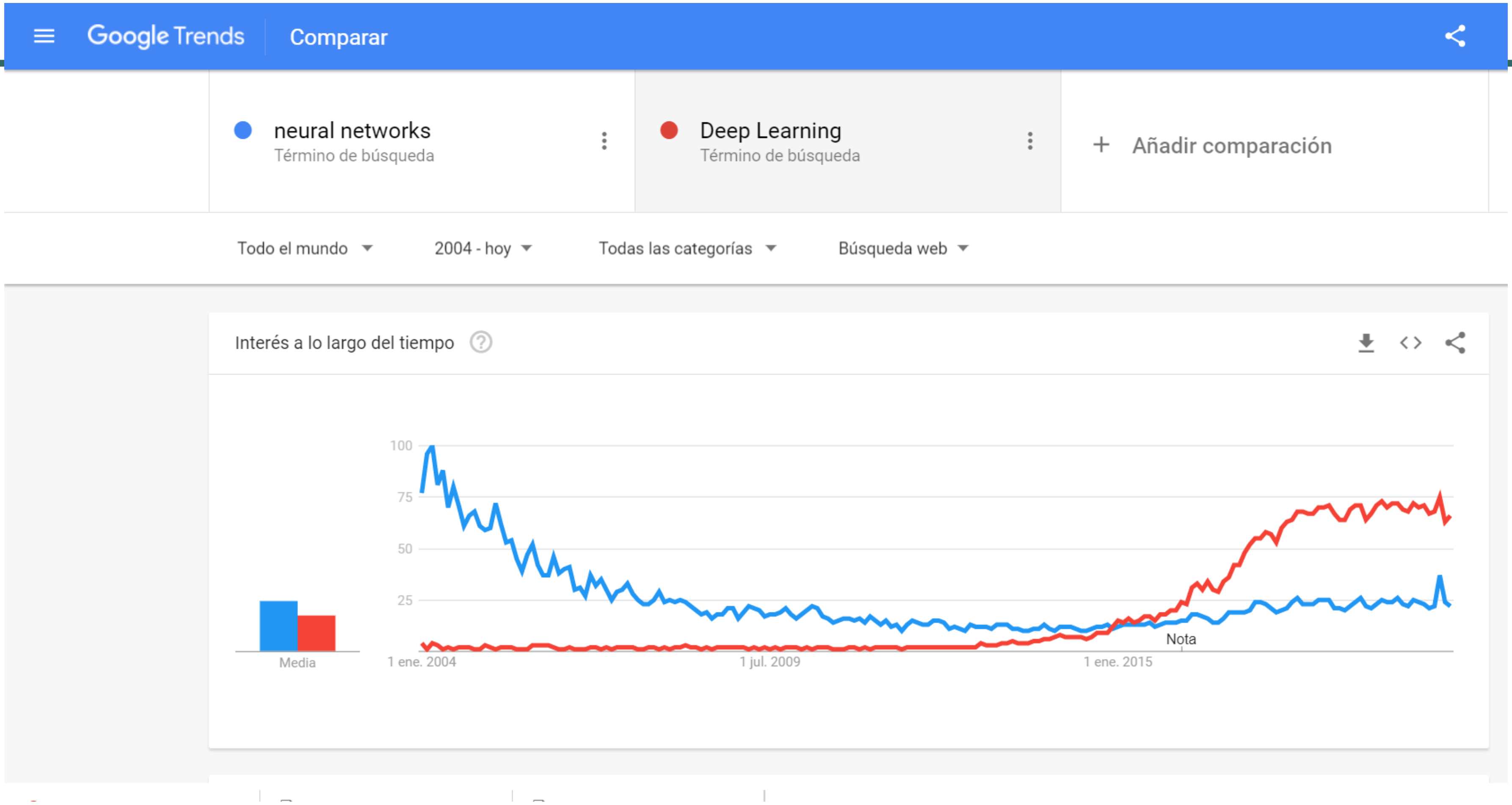
Advantages and disadvantages of Neural Nets

a) Neural nets are good for making predictions but they are hard to understand.

b) NN are good to analyze large and complex datasets.

c) Depending on the problem NN can perform better or worst than any other machine learning algorithm.

d) NN do not have good theory for feature selection or model selection in general.



Google Trends | Comparar

● neural networks
Término de búsqueda

● Deep Learning
Término de búsqueda

+ Añadir comparación

Todo el mundo ▾ | 2004 - hoy ▾ | Todas las categorías ▾ | Búsqueda web ▾

Interés a lo largo del tiempo

Deep Learning

In 2006, Geoffrey Hinton (now at Google) showed that a type of neural network called deep belief network, can be efficiently trained using a strategy called greedy layer-wise pretraining.
Hinton and his associates introduced the term “deep learning” to emphasize that now it would be possible to train neural nets more deeply than before.

At the present is claimed that deep neural networks have better performance than any other machine learning methods.

The success of deep learning has led to the appearance of several libraries to train deep neural networks. Most of these libraries are written in python. For instance, Keras, Theano, and Tensor Flow (Google Brain).

Deep Learning (cont)

The basic idea of Deep Learning is to simulate the array of a large number of neurons of the Brain’s Neocortex in a artificial neural network.
Thanks to improvements in the mathematical formulation and the constant increase in computer power, now scientist can model more layers than neurons than before.
 In June 2012, a Google’s Deep Learning system was capable to identify cats in 10 millions of Youtube videos with the double of efficiency of previous algorithms of image recognition (the team was led for A. Ng and J. Dean). Also, Google has used Deep Learning to reduce the error rate of it’s algorithm for voice recognition on its androids cell phones.

To extend deep learning to applications beyond voice and image recognition it requires more software development (CUDA, 2007) and processing power (GPU).

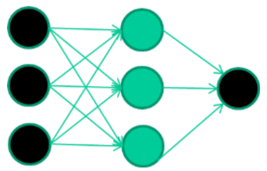
Limitations of Neural Networks

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

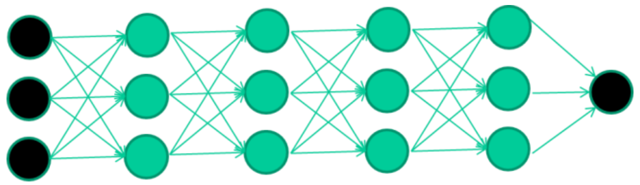
What is new in Deep Learning with respect to NN?

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

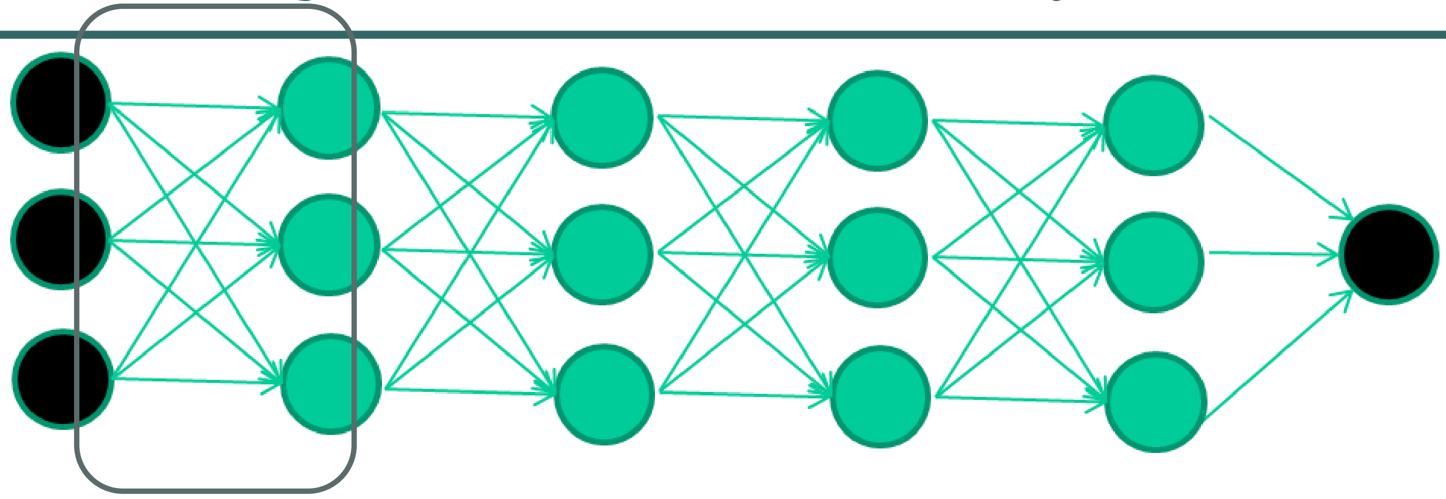


but these algorithms are not good at learning the weights for networks with more hidden layers

what's new is: <u>algorithms for training many-later networks</u>

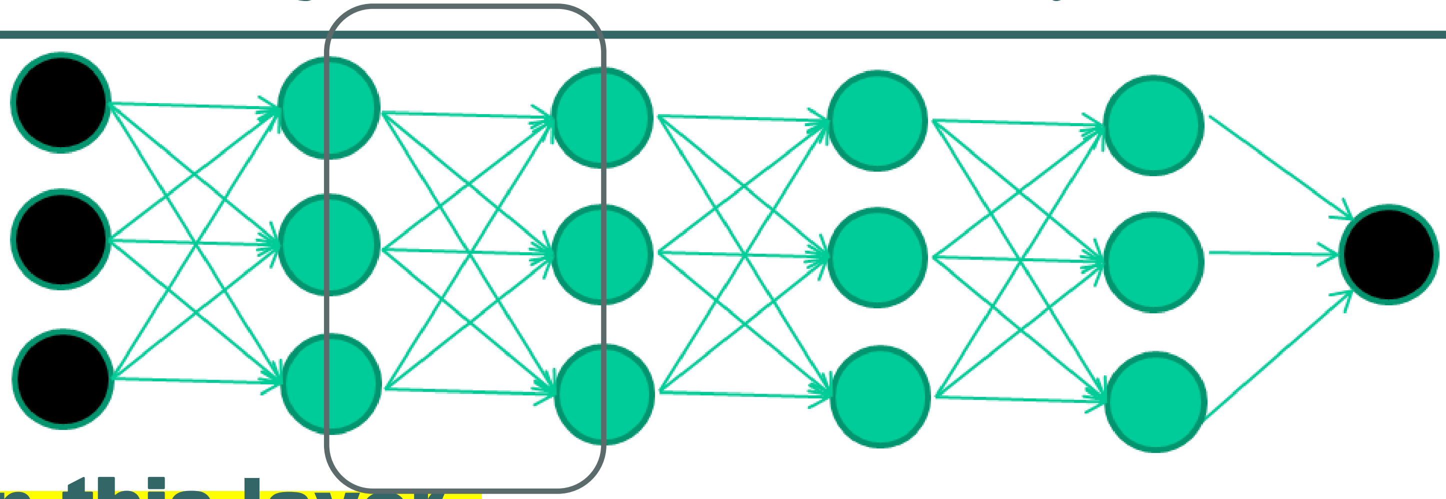


The new way to train multi-layer NNs…



Train this layer first

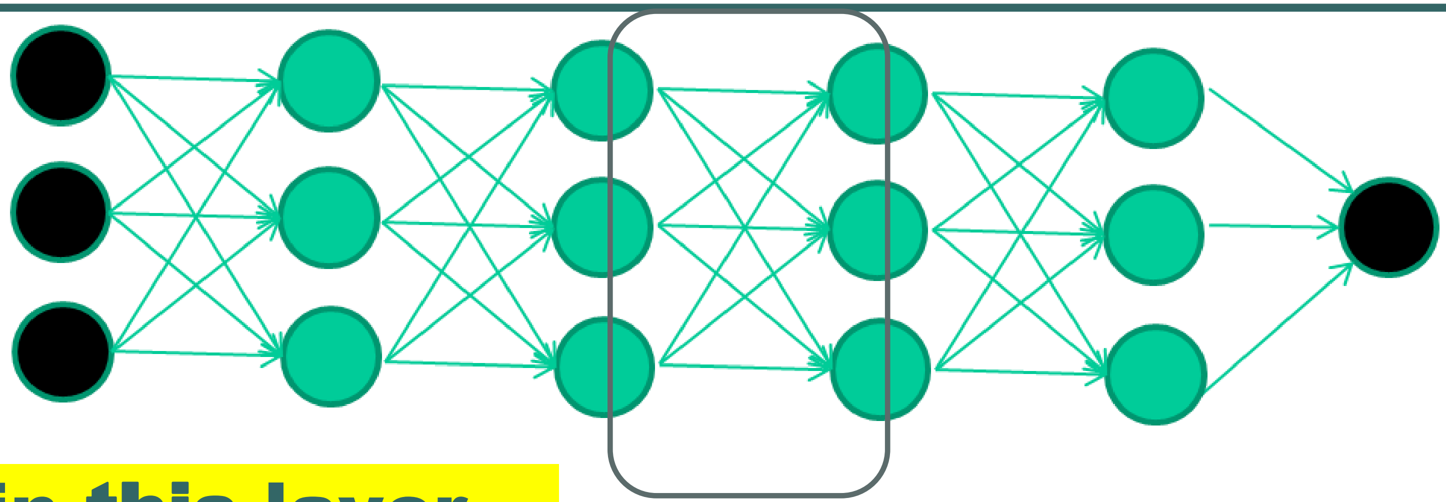
The new way to train multi-layer NNs…



Train this layer first

then this layer

The new way to train multi-layer NNs…

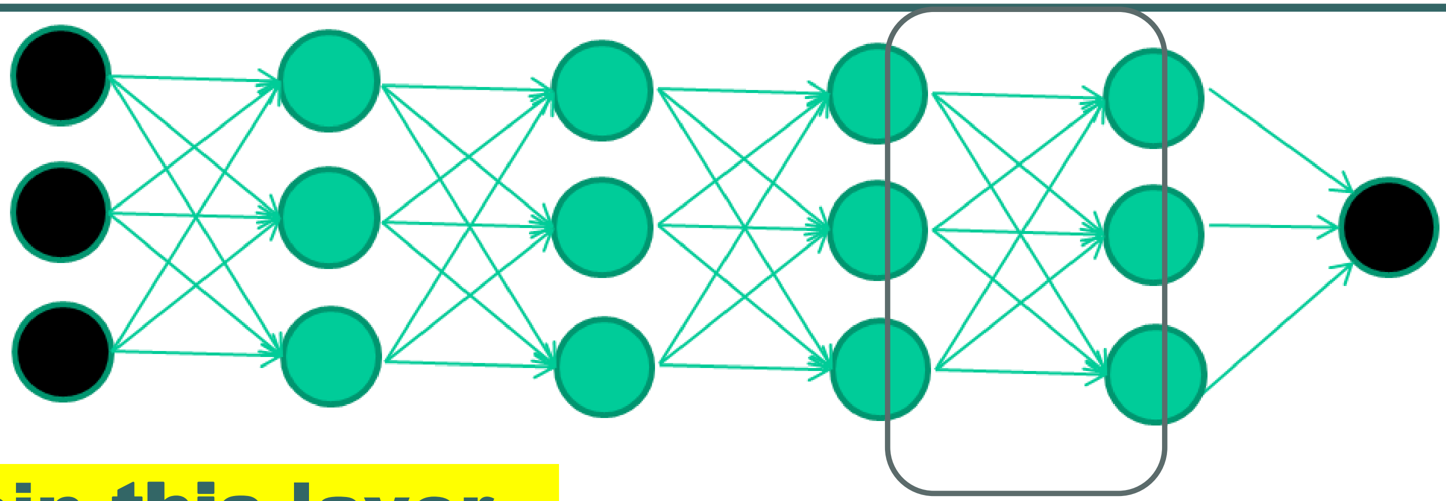


Train this layer first

then this layer

then this layer

The new way to train multi-layer NNs…



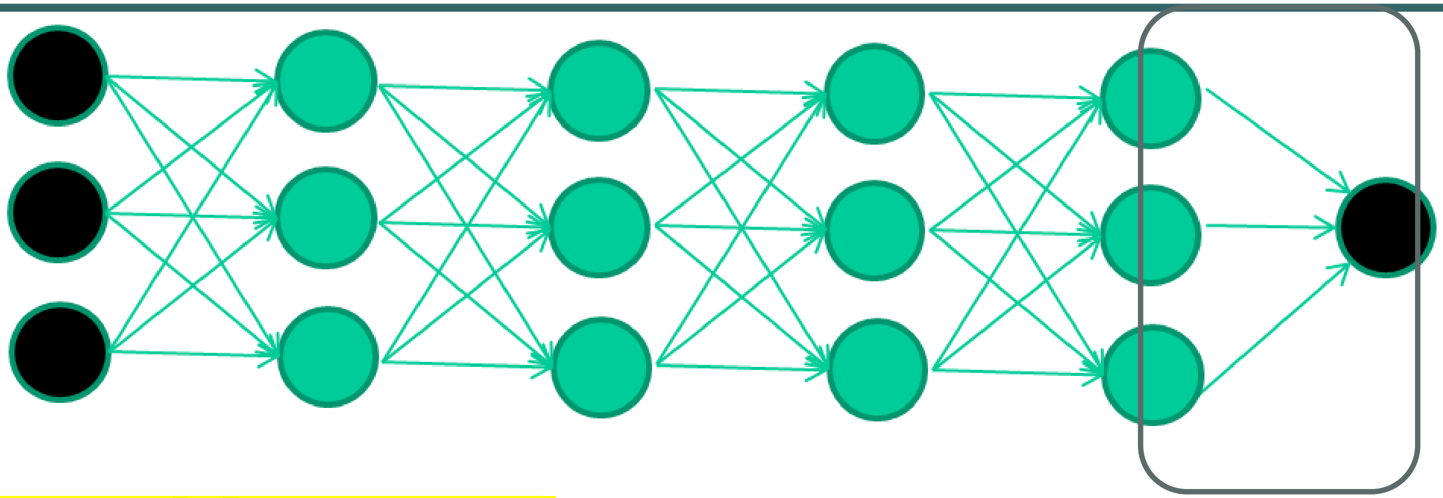
Train this layer first

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

Types of Deep Learning architectures

Multilayer Perceptrons (MLPs), useful for numerical data
Convolutional Neural Networks (CNNs), very popular to analyze data from images (Computer Vision).
Recurrent Neural Networks (RNNs), used for learning from sequential data including text, audio and video. Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.
Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data

Deep Learning using h20

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 using h20

```
 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 using h20

```
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 using h20

```
 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)
```

Data Mining and Machine Learning

Outlier Detection (Novelty Detection)
using autoencoders

Dr. Edgar Acuna
Departmento de Matematicas

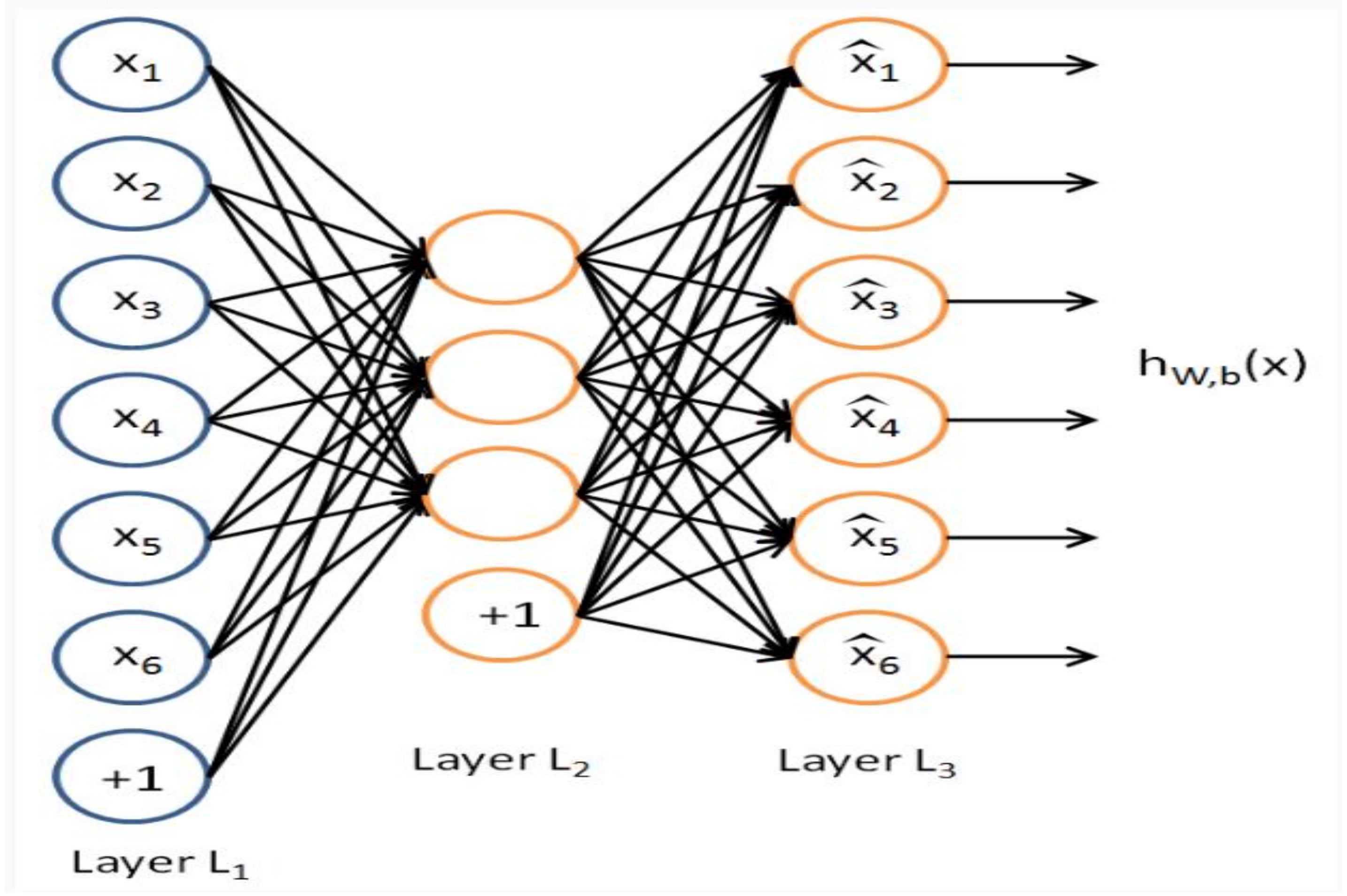
Universidad de Puerto Rico- Mayaguez

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!

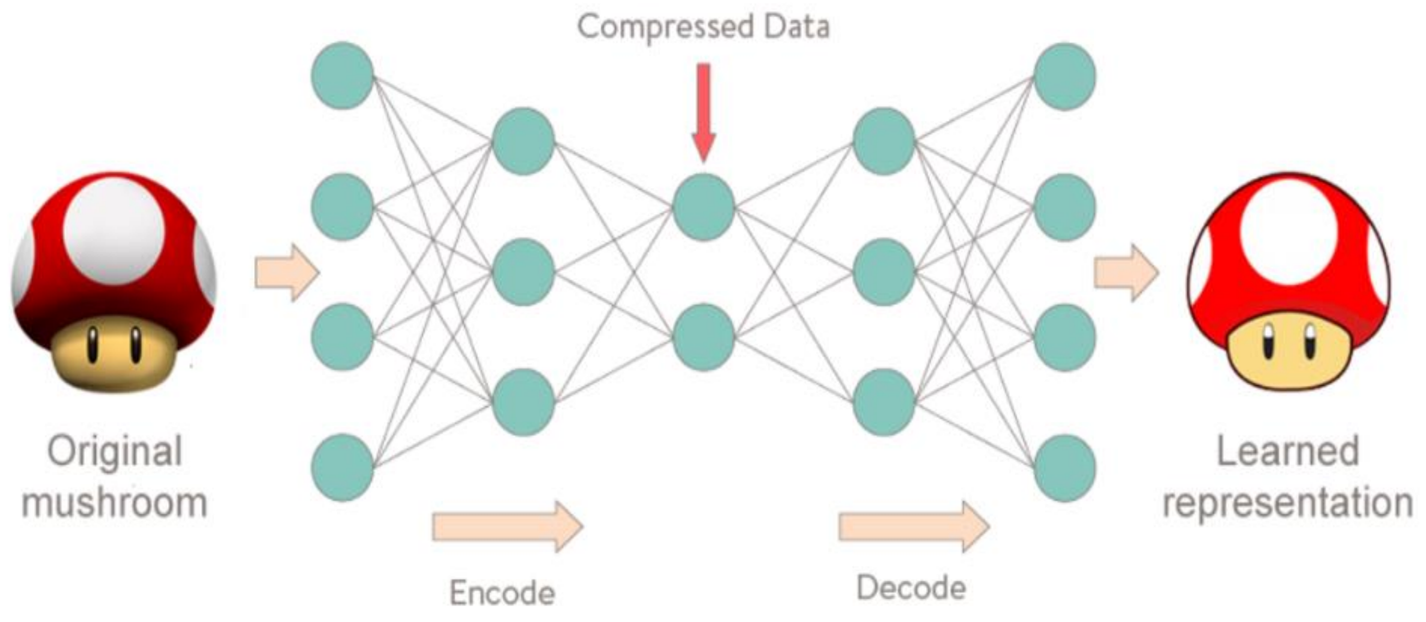
Example of autoencoder



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)

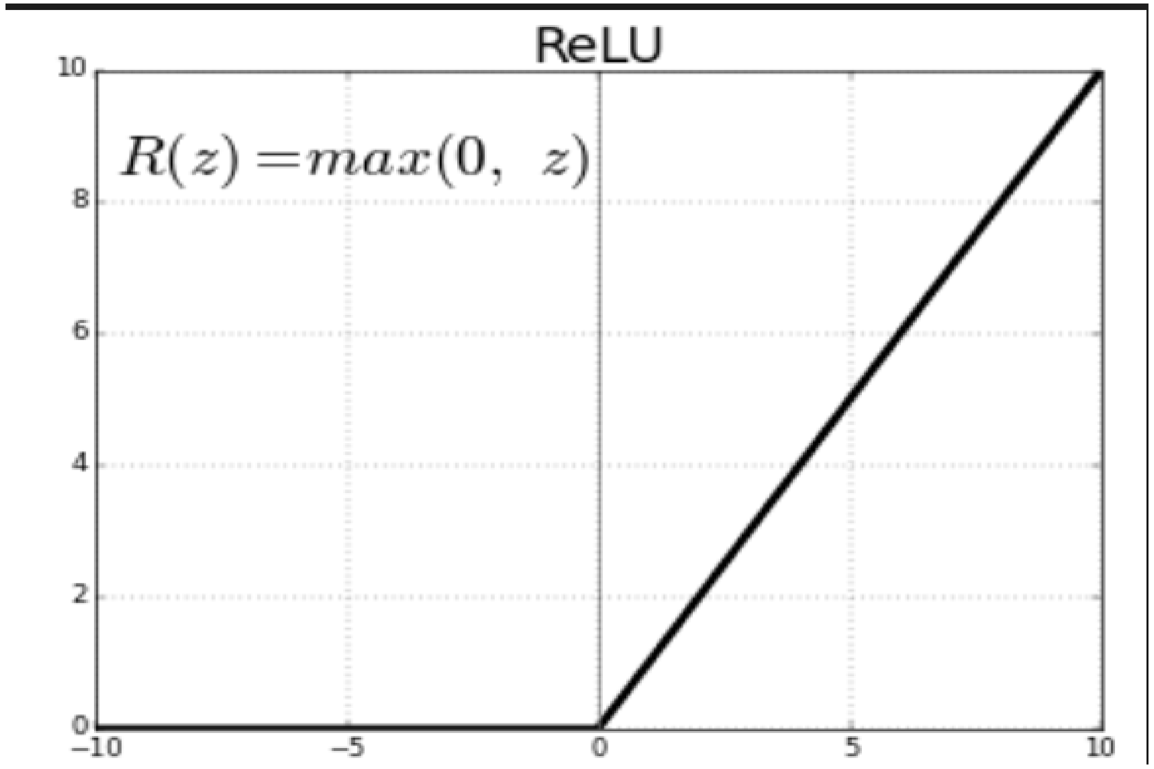
However one has to do tuning on the RE's threshold to detect outliers

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.

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

The Relu activation function



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
```