|  |  |
| --- | --- |
|  | DSNBoost |
|  |  |
| V.2 | Report 12/11/2015 |
|  | Author:  MANUEL MONTOYA CATALÁ |

[1. Introduction 3](#_Toc435108336)

[1.1 Estructura DSNBoost 3](#_Toc435108337)

[1.2 Estructura DSN 4](#_Toc435108338)

[1.3 Algoritmo de Aprendizaje de cada capa 5](#_Toc435108339)

[1.4 Parámetros de los experimentos 5](#_Toc435108340)

[2. Abalone DSNBoost 6](#_Toc435108341)

[2.1 Resultados experimentales totales 6](#_Toc435108343)

[2.1.1 Tabla de Resultados absolutos 6](#_Toc435108344)

[2.1.2 Resultados para Nepoch = 100 7](#_Toc435108345)

[2.1.3 Resultados para diversos Nepoch 7](#_Toc435108346)

[2.2 Evolución con el número de capas 8](#_Toc435108347)

[2.2.1 Realizaciones para Nepoch = 100 y Nh = 26 8](#_Toc435108348)

[2.2.2 Accuracy y nL en función del criterio de parada. 8](#_Toc435108349)

[2.2.3 Estadísticos en función del número de neuronas y capas 9](#_Toc435108350)

[2.2.4 Estadísticos en función del número de neuronas 9](#_Toc435108351)

[3. Kwok DSNBoost 10](#_Toc435108352)

[3.1 Resultados experimentales totales 10](#_Toc435108354)

[3.1.1 Tabla de Resultados absolutos 10](#_Toc435108355)

[3.1.2 Resultados para Nepoch = 100 11](#_Toc435108356)

[3.1.3 Resultados para diversos Nepoch 11](#_Toc435108357)

[3.2 Evolución con el número de capas 12](#_Toc435108358)

[3.2.1 Realizaciones para Nepoch = 100 y Nh = 26 12](#_Toc435108359)

[3.2.2 Accuracy y nL en función del criterio de parada. 12](#_Toc435108360)

[3.2.3 Estadísticos en función del número de neuronas y capas 13](#_Toc435108361)

[3.2.4 Estadísticos en función del número de neuronas 13](#_Toc435108362)

[3.3 Comentarios generales 13](#_Toc435108363)

[4. Ima DSNBoost 14](#_Toc435108364)

[4.1 Resultados experimentales totales 14](#_Toc435108366)

[4.1.1 Tabla de Resultados absolutos 14](#_Toc435108367)

[4.1.2 Resultados para Nepoch = 100 15](#_Toc435108368)

[4.1.3 Resultados para diversos Nepoch 15](#_Toc435108369)

[4.2 Evolución con el número de capas 16](#_Toc435108370)

[4.2.1 Realizaciones para Nepoch = 100 y Nh = 26 16](#_Toc435108371)

[4.2.2 Accuracy y nL en función del criterio de parada. 16](#_Toc435108372)

[4.2.3 Estadísticos en función del número de neuronas y capas 17](#_Toc435108373)

[4.2.4 Estadísticos en función del número de neuronas 17](#_Toc435108374)

[4.3 Comentarios generales 17](#_Toc435108375)

[5. Wav DSNBoost 18](#_Toc435108376)

[5.1 Resultados experimentales totales 18](#_Toc435108378)

[5.1.1 Tabla de Resultados absolutos 18](#_Toc435108379)

[5.1.2 Resultados para Nepoch = 100 19](#_Toc435108380)

[5.1.3 Resultados para diversos Nepoch 19](#_Toc435108381)

[5.2 Evolución con el número de capas 20](#_Toc435108382)

[5.2.1 Realizaciones para Nepoch = 100 y Nh = 26 20](#_Toc435108383)

[5.2.2 Accuracy y nL en función del criterio de parada. 20](#_Toc435108384)

[5.2.3 Estadísticos en función del número de neuronas y capas 21](#_Toc435108385)

[5.2.4 Estadísticos en función del número de neuronas 21](#_Toc435108386)

[5.3 Comentarios generales 21](#_Toc435108387)

[6. DSN 22](#_Toc435108388)

[7. Resultados DSN L = 10 23](#_Toc435108389)

[5.4 Abalone 23](#_Toc435108390)

[5.5 Kwok 24](#_Toc435108391)

[5.6 Ima 25](#_Toc435108392)

[5.7 Wav 26](#_Toc435108393)

[8. Evolución DSN L = 10 27](#_Toc435108394)

[5.8 Evolución de gamma 27](#_Toc435108395)

[5.9 Evolución de la accuracy 27](#_Toc435108396)

[9. Conclusiones finales DSNboost 29](#_Toc435108397)

[5.9.1 Cambiar forma de entrenamiento de las capas 30](#_Toc435108398)

[5.9.2 Cambiar inicialización de las capas 30](#_Toc435108399)

[5.9.3 Cambiar estructura 30](#_Toc435108400)

[5.9.4 Aplicar función de énfasis sólo con alpha. 32](#_Toc435108401)

[10. Resultados Finales 33](#_Toc435108402)

# Introduction

En este report se analizarán cuantitativa y cualitativamente las propiedades de 2 algoritmos de entrenamiento para Redes Neuronales de tipo profundo:

* DSN: Deep Stacked Networks
* DSNBoost: Deep Stacked Networks Boosting

## Estructura DSNBoost

En esta estructura, cada capa es entrenada con:

* Toda la entrada inicial .
* Una inyección que va a todas las neuronas ocultas.
* Una función de énfasis de cada muestra es:

Donde son parámetros de validar y es la salida total del sistema normalizada para la muestra

es el valor deseado a la salida para la muestra .

**Nota:** Este report está realizado para , es decir, no se está aplicando realmente una función de énfasis.

La **inyección** utilizada en este report en la capa es la salida total del sistema para la capa anterior normalizada, es decir:

La salida para cada capa , es el sumatorio ponderado de las salidas individuales de cada capa, . La cosntante de agregación utilizada por cada capa,, se obtiene utilizando la ecuación del RAB:

Tenemos que la salida total para la capa es:

**Nota**: Como se puede apreciar en la ecuación de la salida total , conforme avance el número de capas, las nuevas salidas tendrán menos efecto en la salida global ya que aumenta conforme apilamos capas.

El **criterio de parada** para dejar de apilar capas está basado en la estabilidad de :

* Definiendo una ventana de gammas, , se calcula la diferencia entre la media de las últimas y las penúltimas normalizado por la media. Si esta diferencia es menor que un C (porcentaje de variación de ), entonces se para de apilar capas. Es decir, el algoritmo se para en la capa L si:

En la práctica se ha utilizado y C = 0.001. Cuanto mayor sea ng y menor sea C, más estabilidad de gamma será requerida para parar el aprendizaje (aumenta el número medio de capas).

La siguiente figura muestra la estructura general del DSNBoost

## Estructura DSN

En esta estructura, cada capa es entrenada con:

* Toda la entrada inicial .
* Todas las activaciones de capas anteriores normalizadas
* La misma función de énfasis

La **inyección** utilizada en este report en la capa es la acticación de la capa de salida de todas las capas anteriores

La salida para cada capa , es simplemente la salida de dicha capa:

El **criterio de parada** para dejar de apilar capas también está basado en la estabilidad de , para que ambas estructuras tuvieran un criterio común. Si bien se calcula de la misma manera que en DSNBoost, no se utiliza para nada en el algoritmo.

## Algoritmo de Aprendizaje de cada capa

El algoritmo de aprendizaje **para cada capa** es el típico Backpropagation (sin Batch-Mode para asegurarnos). Los parámetros utilizados son:

* Inicialización de pesos: Variable aleatoria uniforme entre -ro y ro donde:

En deep learning se suele utilizar este criterio multiplicando esta ro por , nosotros la bajamos y nos curamos en salud pero podríamos probar con una ro mayor.

* Constante de aprendizaje Inicial: La constante de aprendizaje inicial es
* Criterio de parada: Se utilizado un criterio de parada que no requiera utilizar error de validación (como se pidió). El algoritmo tiene un **número de épocas fijo**:

Aunque se han realizado experimentos para diferentes valores de

La constante de aprendizaje **decae linealmente** **a 0** durante el número de épocas.

* La función de coste a minimizar es el error cuadrático medio, MSE.
* La función de activación de las **neuronas ocultas** es tanh.
* La función de activación de la **neurona de salida** es tanh.

## Parámetros de los experimentos

Para el cálculo del **error omnisciente** se han llevado a cabo **40 realizaciones** para cada conjunto de parámetros.

Para el cálculo del **error de validación**, se han utilizado **10 realizaciones** de un **5-fold**.

En las gráficas se visualiza la **accuracy** del algoritmo, el porcentaje de acierto.

Siempre que hablemos de error de entrenamiento o de test nos estamos refiriendo a los errores omnisciente. Cuando hablemos de error de validación, nos referimos al error obtenido la cross-validación.

En los experimentos se han validado fundamentalmente el número de neuronas y el número de épocas , si bien también se realizan análisis a nivel de número de capas.

# Abalone DSNBoost

A continuación se muestran los diversos resultados para la base de datos Abalone utilizando la estructura DSNBOOST. Se mostrarán 2 tipos de resultados, primero, los resultados experimentales del algoritmo descrito anteriormente utilizando el criterio de parada mencionado anteriormente. Posteriormente se analizan resultados del algoritmo capa a capa con el objetivo de visualizar su comportamiento.



## Resultados experimentales totales

En esta sección se mostrarán los errores de train, validación y test para para diferente número de neuronas y épocas. Mostrándose inicialmente los mejores valores de [Nepoch, Nh] tanto omniscientes como crossvalidados.

### Tabla de Resultados absolutos

Empezaremos dando los resultados finales del algoritmo, indicando tanto para el caso OMN, como para el CV, cuales son los mejores resultados en función de la media del error. Se muestra para combinación [Nepoch, Nh], el error de tr, val y tst y la standard deviation del error de test.

**Error CV:**

[Nepoch = 100 Nh: 34] Tr: 0.195542077219 Val: 0.20138992835 Tst: 0.190434131737 std Tst: 0.00196096356572

[Nepoch = 100 Nh: 38] Tr: 0.196100645857 Val: 0.201431125001 Tst: 0.190104790419 std Tst: 0.00147278306746

[Nepoch = 100 Nh: 44] Tr: 0.196738726766 Val: 0.201789530421 Tst: 0.18997005988 std Tst: 0.00126848623738

[Nepoch = 100 Nh: 41] Tr: 0.196339858503 Val: 0.201790010736 Tst: 0.190269461078 std Tst: 0.00101752368072

[Nepoch = 100 Nh: 35] Tr: 0.195921035133 Val: 0.202188415201 Tst: 0.190359281437 std Tst: 0.00178036751465

[Nepoch = 100 Nh: 24] Tr: 0.194943757371 Val: 0.202307935841 Tst: 0.19005988024 std Tst: 0.00196695421625

[Nepoch = 100 Nh: 32] Tr: 0.195791309602 Val: 0.202468334725 Tst: 0.19004491018 std Tst: 0.00124665639801

[Nepoch = 100 Nh: 18] Tr: 0.195083358562 Val: 0.202504512568 Tst: 0.18994011976 std Tst: 0.00134697268014

[Nepoch = 100 Nh: 23] Tr: 0.195212875328 Val: 0.202546502851 Tst: 0.18994011976 std Tst: 0.00164098079114

**Error OMN:**

[Nepoch = 200 Nh: 10] Tr: 0.192131113084 Val: 0.206295107633 Tst: 0.188937125749 std Tst: 0.00255804115049

[Nepoch = 200 Nh: 12] Tr: 0.192111401516 Val: 0.206174627319 Tst: 0.188997005988 std Tst: 0.00227894271671

[Nepoch = 200 Nh: 14] Tr: 0.191622747737 Val: 0.205738850586 Tst: 0.189206586826 std Tst: 0.00243414278452

[Nepoch = 200 Nh: 11] Tr: 0.191273550973 Val: 0.20665534461 Tst: 0.189251497006 std Tst: 0.00199028621013

[Nepoch = 200 Nh: 16] Tr: 0.191692220271 Val: 0.205139568512 Tst: 0.189326347305 std Tst: 0.00194904200723

[Nepoch = 200 Nh: 13] Tr: 0.191881691802 Val: 0.205617018075 Tst: 0.189356287425 std Tst: 0.00256329218989

[Nepoch = 200 Nh: 33] Tr: 0.1928291983 Val: 0.203145386359 Tst: 0.189356287425 std Tst: 0.00226236190767

[Nepoch = 200 Nh: 38] Tr: 0.19378653097 Val: 0.203584509388 Tst: 0.189401197605 std Tst: 0.00199590818114

[Nepoch = 200 Nh: 47] Tr: 0.193846381359 Val: 0.20457940915 Tst: 0.189401197605 std Tst: 0.00182708338951

[Nepoch = 200 Nh: 9] Tr: 0.192011422311 Val: 0.207370962299 Tst: 0.189491017964 std Tst: 0.00240249640088

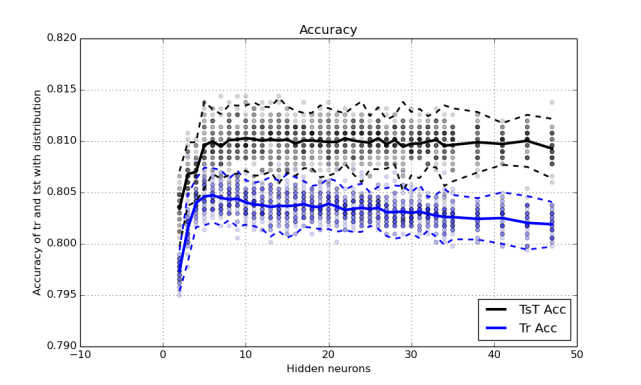
[Nepoch = 200 Nh: 41] Tr: 0.19346736879 Val: 0.203744111458 Tst: 0.189505988024 std Tst: 0.00216310890233

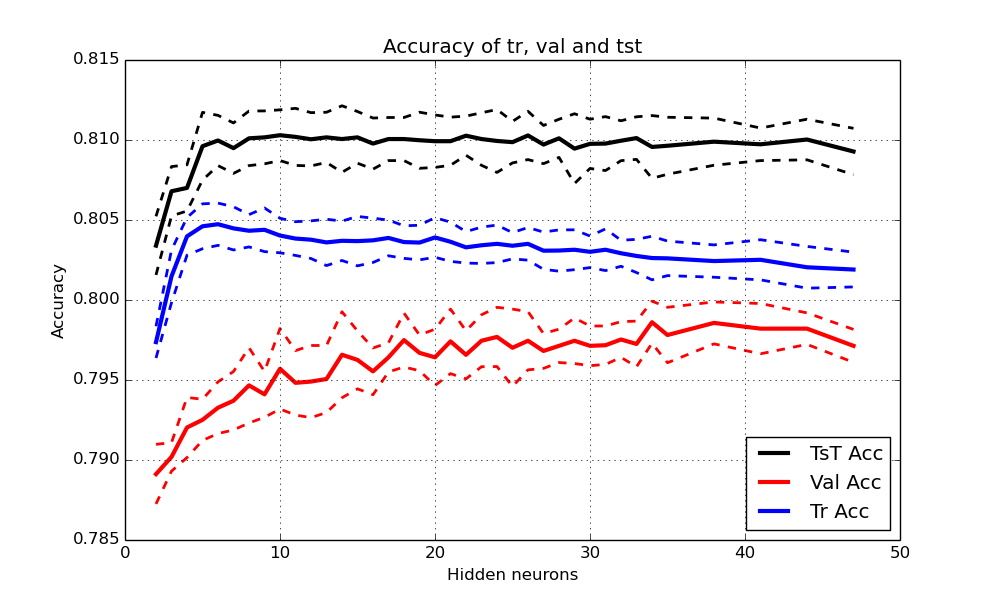
Observaciones:

* Obviamente el error OMN es menor que el error CV pero también suele tener una menor varianza.
* El error OMN mejora conforme aumenta el número de épocas mientras que el CV es mejor con menos épocas.
* La variación entre los mejores OMN y CV es de 0.14% de error.

### Resultados para Nepoch = 100

En esta sección se mostrarán diversas cualidades del algoritmo en función del número de neuronas ocultas para 100 épocas:





**Primera figura**: Muestra la evolución de los errores de train, validación y test para 100 épocas en función del número de neuronas.

**Segunda figura**: Muestra la evolución de los errores de train y test para 100 épocas en función del número de neuronas mostrando además los valores de las 40 realizaciones para ver cómo se distribuyen.

El error omnisciente se satura en Nh = 5, ni sube ni baja.

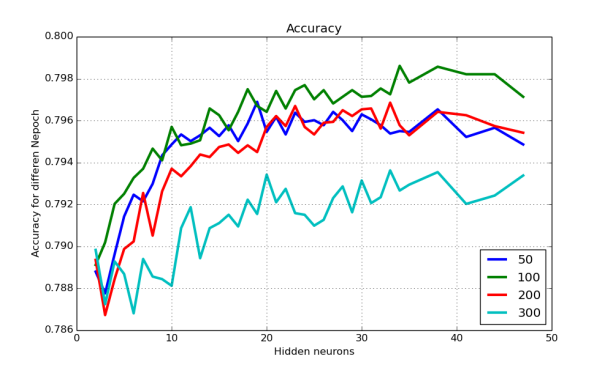
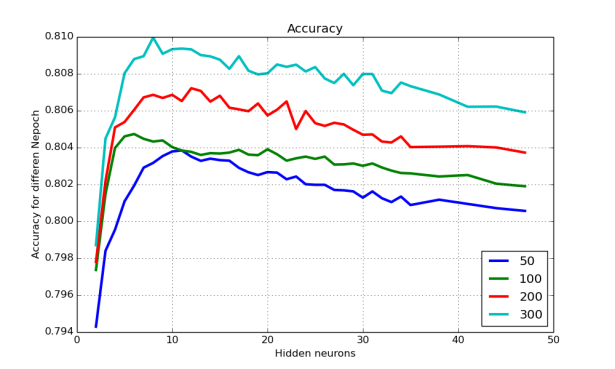
El error de entrenamiento aumenta con Nh: Underfitting ó menos overfitting ?

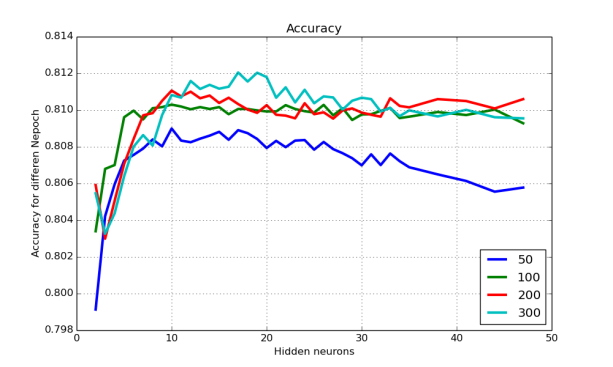
El error de validación disminuye con Nh pero parece que a partir de 40 Nh baja.

La distribución de las realizaciones parece correcta, no se aprecia nada raro. Distribución gaussiana, Teorema Central del Límite.

### Resultados para diversos Nepoch

Las siguientes gráficas muestran la evolución de la accuracy de tr, val y tst en función del número de neuronas y el número de épocas:





El error de tr aumenta con Nh.

El error de val disminuye con Nh. Falta hacer estudio para mayor Nh.

El error de tst disminuye con Nh y si el entrenamiento es suficiente, >50 epoch, se satura en 19.

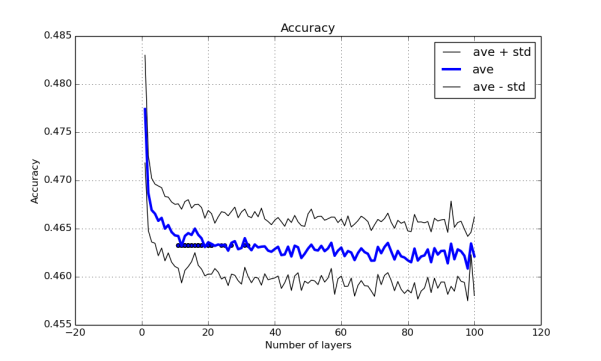
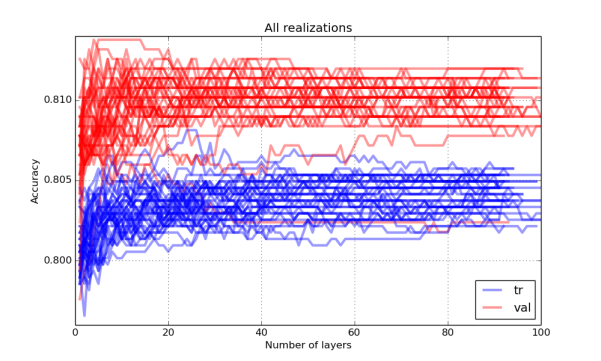
Mejor error CV alcanzado para Nepoch = 100 y mejor error OMN para Nepoch = 300

## Evolución con el número de capas

Con el objetivo de saber cómo se comporta el algoritmo conforme aumenta el número de capas, se han realizado 40 realizaciones para Nh entre 17 y 47 en pasos de 3 y Nepoch = 50,100,200 del **error omnisciente**. Si bien estos resultados no nos sirven para validar los parámetros, nos dan una idea de las capacidades del algoritmo y de cómo evoluciona para ver cómo deberíamos modificarlo.

### Realizaciones para Nepoch = 100 y Nh = 26

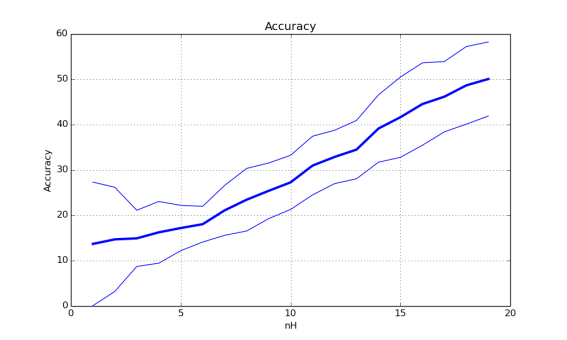
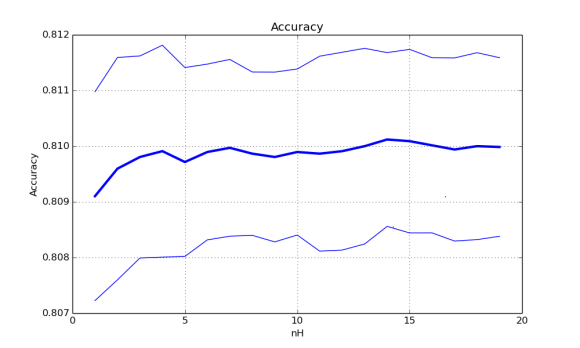
Como no podemos ver cómo evoluciona el error para todo número de neuronas en una sola gráfica, se ha optamos por esos parámetros que están en un punto intermedio del barrido para que sean representativos a nivel cualitativo. La primera gráfica muestra el accuracy de tr y tst para las 40 realizaciones y la segunda muestra la media y la varianza de gamma, ambas en función del número de capas. La nube de puntos que se ve en la segunda gráfica señala el número de capas donde pararía el criterio de parada utilizado inicialmente para las realizaciones.



Mirando las realizaciones una a una se puede observar que el resultado final de la realización depende mucho de las primeras capas. Las **realizaciones son estables y consistentes**. El número de cambios disminuye conforme aumenta nL. El valor de **gamma** disminuye un poco con el tiempo pero no significativamente. A partir del nL = 18 se estanca.

### Accuracy y nL en función del criterio de parada.

Dado que disponemos del resultado de todas las realizaciones capa a capa, podemos aplicar “offline” el criterio de parada que deseemos y ver las propiedades del algoritmo respecto a este. Las siguiente gráficas muestra la accuracy y el número de capas obtenido para diferentes tamaño de la ventana .



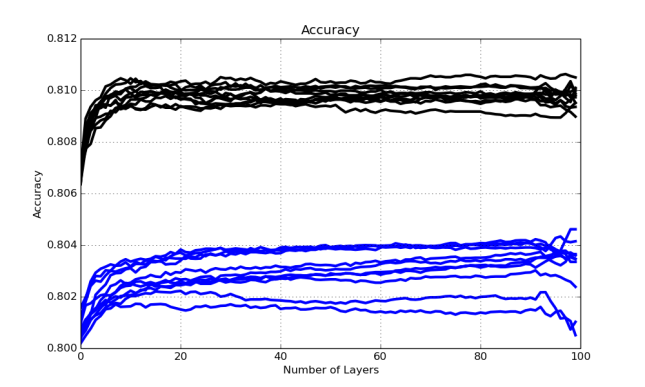
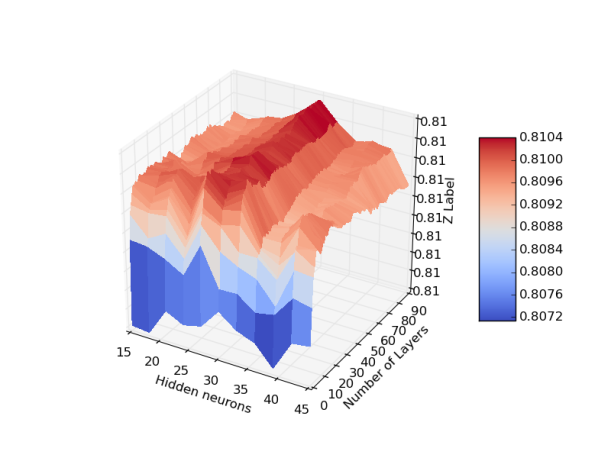
El accuracy no sufre grandes cambios con respecto al tamaño de la ventana. Es un poco menor al principio ya que algunas realizaciones paran en la etapa inicial 0-20 capas donde aún no se ha estabilizado.

El número de capas medio aumenta linealmente con el tamaño de ventana.

### Estadísticos en función del número de neuronas y capas

A falta de una mejor representación de las características del algoritmo en función del número de capas y de neuronas, presentamos 2 gráficas:

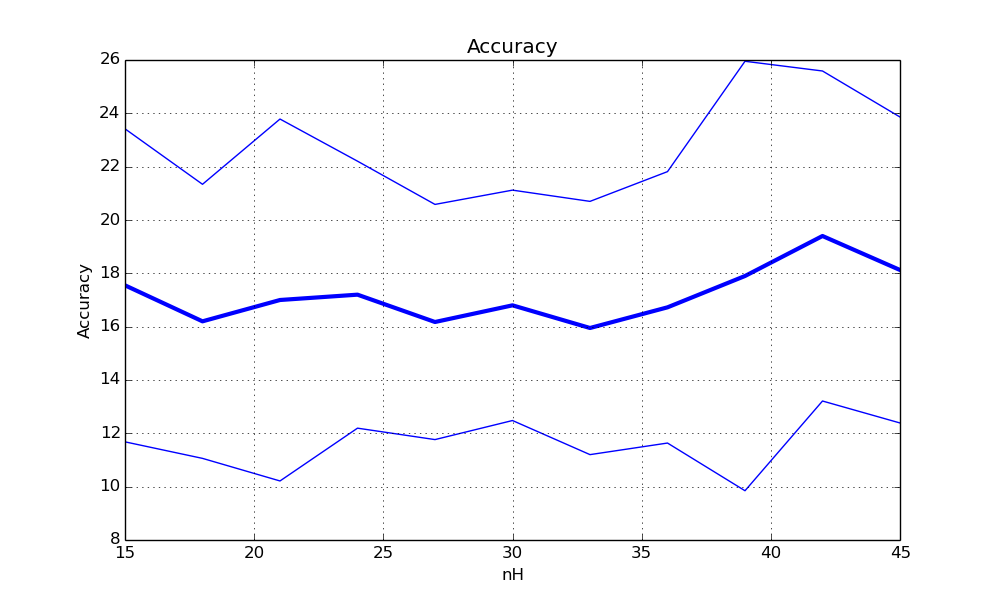
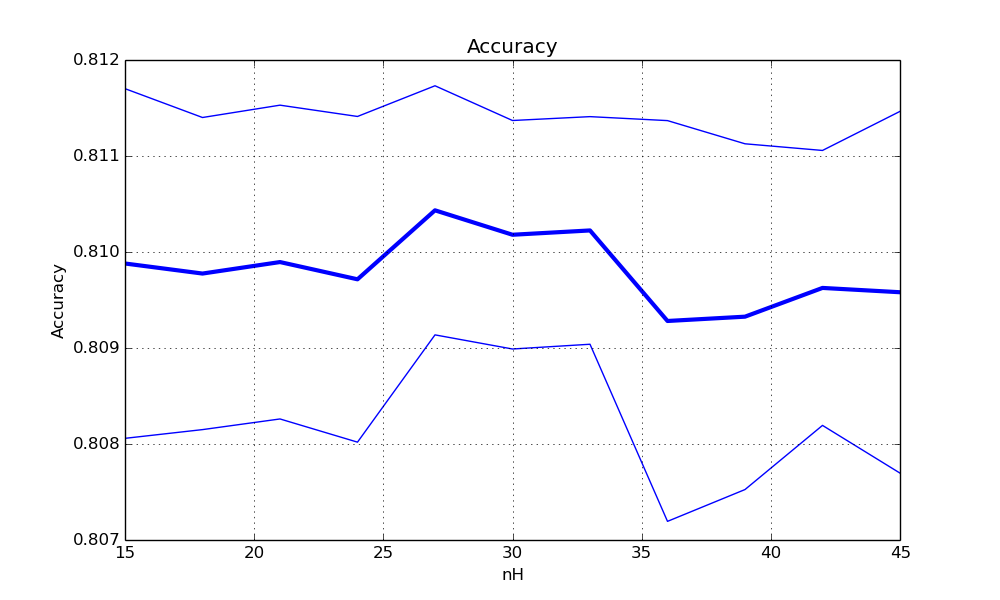
La primera muestra el accuracy de train y test para diferentes número de neuronas ocultas en función del número de capas pero no podemos distinguir claramente cual es cual (aún no está señalizado) pero la segunda gráfica nos proporciona esta información.

El comportamiento del algoritmo es muy similar independientemente del número de neuronas y los errores de tr y tst parecen estables a lo largo del número de capas, es decir, las curvas no se están entrecruzando todo el rato.

### Estadísticos en función del número de neuronas

Si aplicamos el criterio de parada inicial (ventana = 5) podemos ver las propiedades del algoritmo en función del número de neuronas, quitándonos la dimensión de las capas. Las siguientes gráficas muestran la accuracy y número medio de capas en función del número de neuronas ocultas utilizando dicho criterio de parada.



No se aceptan grandes diferencias con el número de neuronas pero el punto óptimo sería alrededor de 30 neuronas.

# Kwok DSNBoost

A continuación se muestran los diversos resultados para la base de datos Kwok utilizando la estructura DSNBOOST. Se mostrarán 2 tipos de resultados, primero, los resultados experimentales del algoritmo descrito anteriormente utilizando el criterio de parada mencionado anteriormente. Posteriormente se analizan resultados del algoritmo capa a capa con el objetivo de visualizar su comportamiento.



## Resultados experimentales totales

En esta sección se mostrarán los errores de train, validación y test para para diferente número de neuronas y épocas. Mostrándose inicialmente los mejores valores de [Nepoch, Nh] tanto omniscientes como crossvalidados.

### Tabla de Resultados absolutos

Empezaremos dando los resultados finales del algoritmo, indicando tanto para el caso OMN, como para el CV, cuales son los mejores resultados en función de la media del error. Se muestra para combinación [Nepoch, Nh], el error de tr, val y tst y la standard deviation del error de test.

**Error CV:**

[Nepoch = 100 Nh: 20] Tr: 0.0944 Val: 0.0956 Tst: 0.115406862745 std Tst: 0.000656039138329

[Nepoch = 100 Nh: 18] Tr: 0.0927 Val: 0.0964 Tst: 0.115404411765 std Tst: 0.000845410609363

[Nepoch = 100 Nh: 16] Tr: 0.0934 Val: 0.0976 Tst: 0.115286764706 std Tst: 0.000846603539364

[Nepoch = 100 Nh: 12] Tr: 0.0918 Val: 0.0976 Tst: 0.11524754902 std Tst: 0.000612642148214

[Nepoch = 100 Nh: 17] Tr: 0.0928 Val: 0.098 Tst: 0.115480392157 std Tst: 0.00132276666475

[Nepoch = 200 Nh: 38] Tr: 0.0914 Val: 0.0982 Tst: 0.115232843137 std Tst: 0.000573628908748

**Error OMN:**

[Nepoch = 200 Nh: 41] Tr: 0.0914 Val: 0.0992 Tst: 0.114960784314 std Tst: 0.000551900703727

[Nepoch = 200 Nh: 35] Tr: 0.0913 Val: 0.0994 Tst: 0.115112745098 std Tst: 0.000559575517431

[Nepoch = 200 Nh: 32] Tr: 0.09115 Val: 0.099 Tst: 0.115115196078 std Tst: 0.000418759163281

[Nepoch = 200 Nh: 27] Tr: 0.0906 Val: 0.1014 Tst: 0.115120098039 std Tst: 0.000373635863067

[Nepoch = 200 Nh: 47] Tr: 0.09215 Val: 0.0996 Tst: 0.115151960784 std Tst: 0.000651407696989

[Nepoch = 200 Nh: 33] Tr: 0.09125 Val: 0.1002 Tst: 0.115174019608 std Tst: 0.000488962909665

[Nepoch = 200 Nh: 44] Tr: 0.0917 Val: 0.1002 Tst: 0.115178921569 std Tst: 0.000526082259017

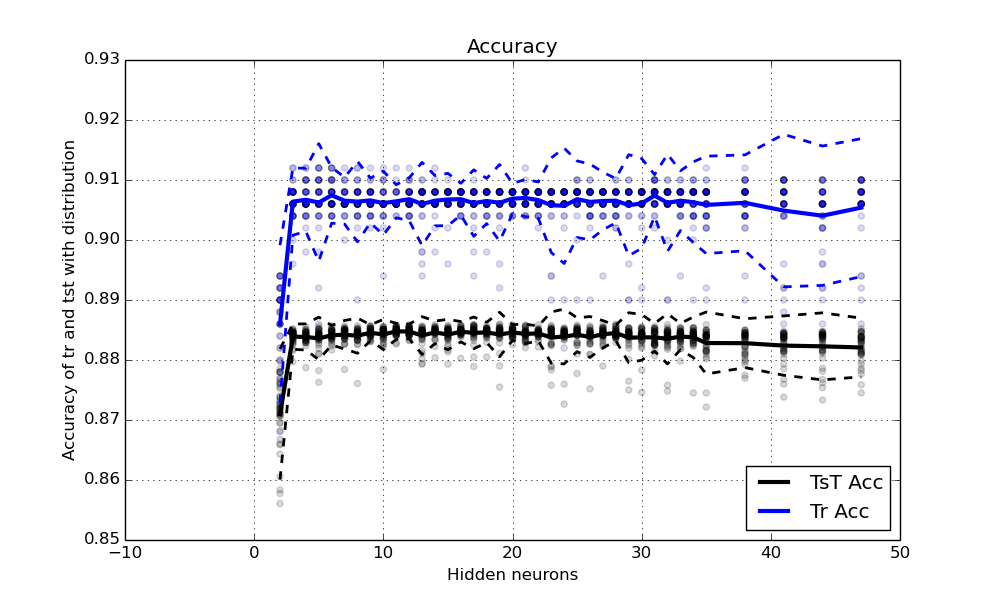
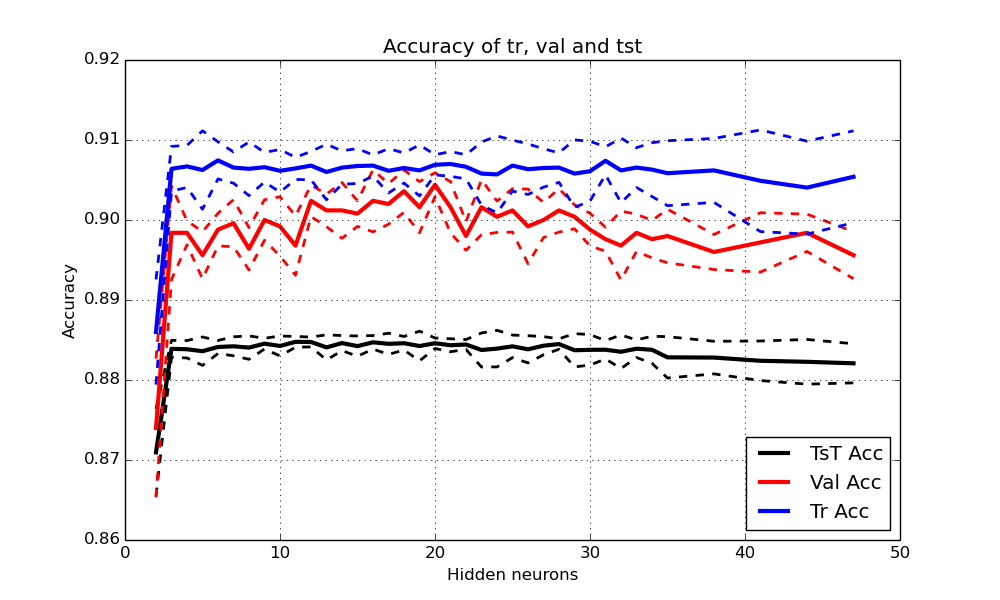
[Nepoch = 200 Nh: 34] Tr: 0.09115 Val: 0.0996 Tst: 0.115227941176 std Tst: 0.000496715229119

Observaciones:

* Obviamente el error OMN es menor que el error CV.
* El error OMN mejora conforme aumenta el número de épocas mientras que el CV es mejor con menos épocas.
* La variación entre los mejores OMN y CV es de 0.04% de error.

### Resultados para Nepoch = 100

En esta sección se mostrarán diversas cualidades del algoritmo en función del número de neuronas ocultas para 100 épocas:



**Primera figura**: Muestra la evolución de los errores de train, validación y test para 100 épocas en función del número de neuronas.

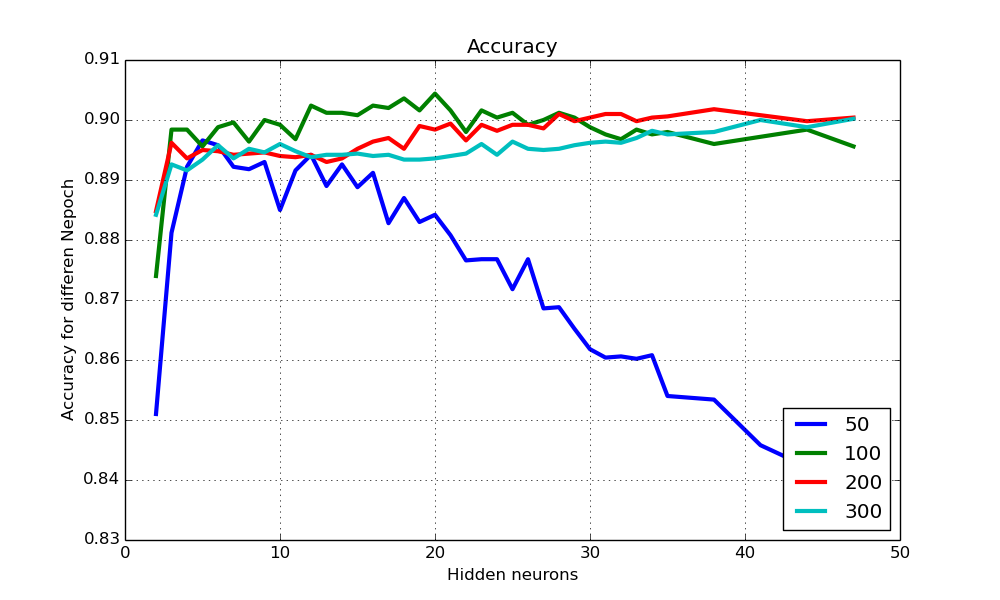
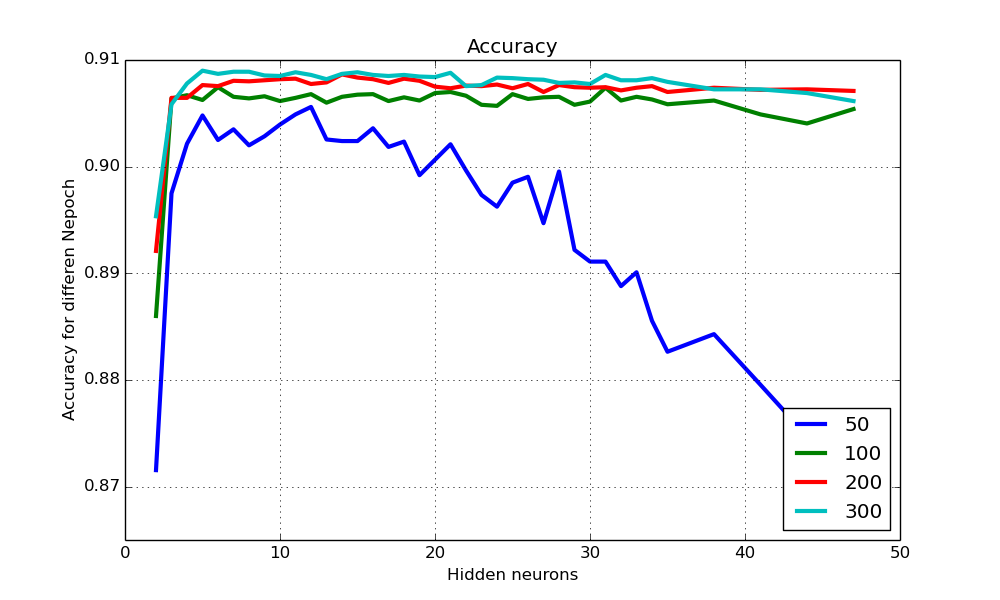
**Segunda figura**: Muestra la evolución de los errores de train y test para 100 épocas en función del número de neuronas mostrando además los valores de las 40 realizaciones para ver cómo se distribuyen.

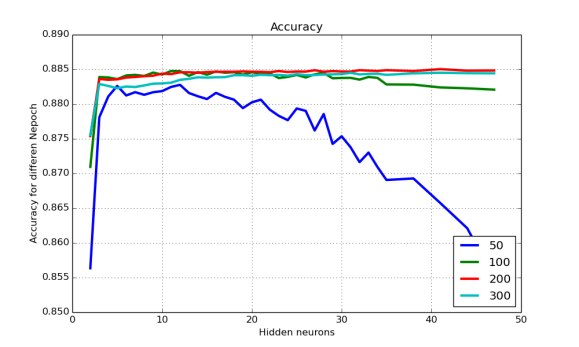
El error omnisciente se satura en Nh = 5.

La distribución de las realizaciones parece correcta. Conforme aumenta el número de neuronas la **varianza aumenta** y hay más outliers respecto a una distribución gaussiana.

### Resultados para diversos Nepoch

Las siguientes gráficas muestran la evolución de la accuracy de tr, val y tst en función del número de neuronas y el número de épocas:



El error de tr está saturado execto para Nepoch = 50 donde la red no se entrena lo suficiente y acaba haciendo underfitting conforme aumenta Nh

El error de val disminuye con Nh. Falta hacer estudio para mayor Nh.

El error de tst disminuye con Nh si el entrenamiento no es suficiente después de saturacion

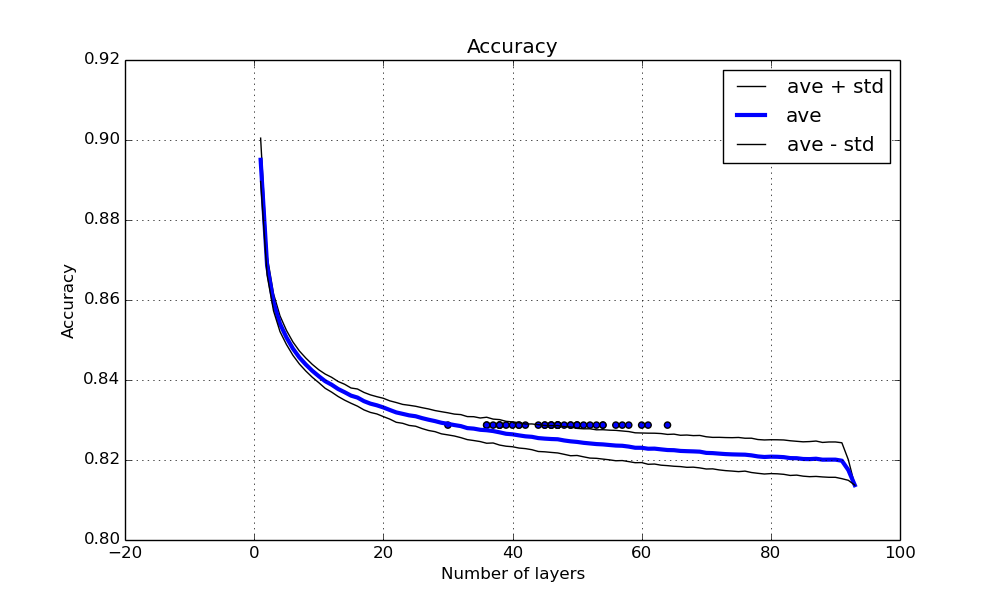
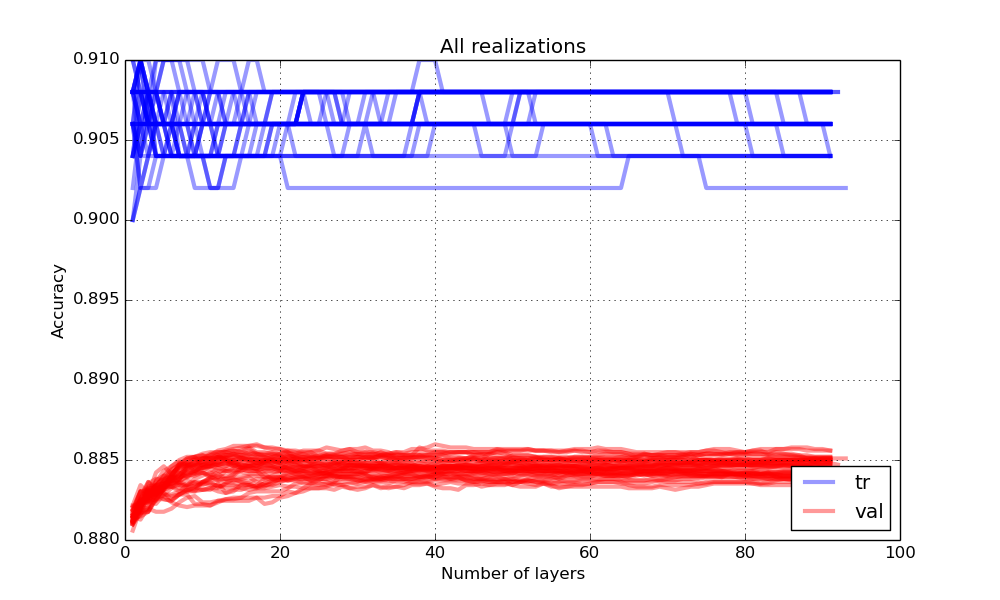
Mejor error CV alcanzado para Nepoch = 100 y mejor error OMN para Nepoch = 200.

## Evolución con el número de capas

Con el objetivo de saber cómo se comporta el algoritmo conforme aumenta el número de capas, se han realizado 40 realizaciones para Nh entre 17 y 47 en pasos de 3 y Nepoch = 50,100,200 del **error omnisciente**. Si bien estos resultados no nos sirven para validar los parámetros, nos dan una idea de las capacidades del algoritmo y de cómo evoluciona para ver cómo deberíamos modificarlo.

### Realizaciones para Nepoch = 100 y Nh = 26

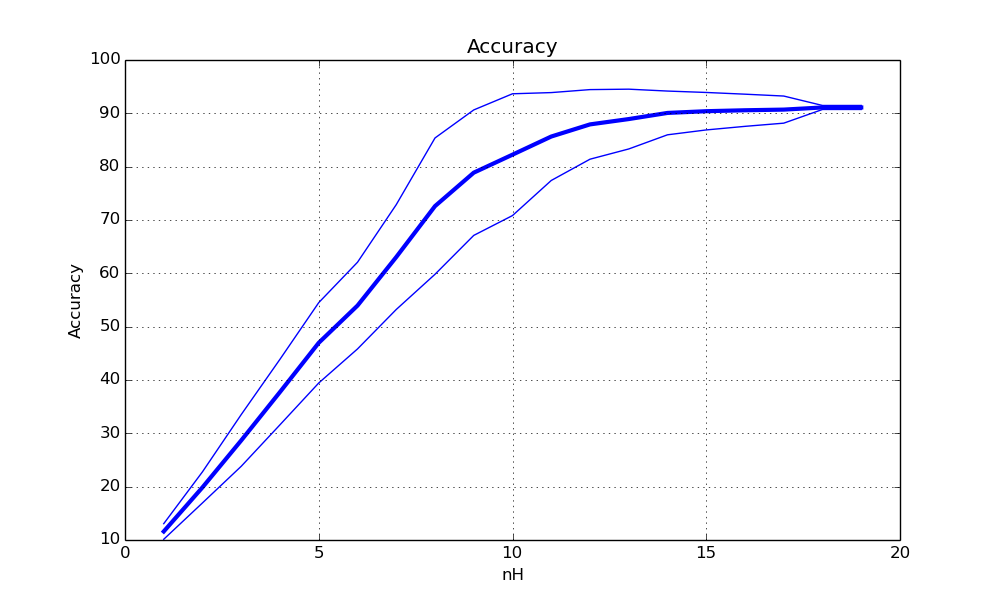
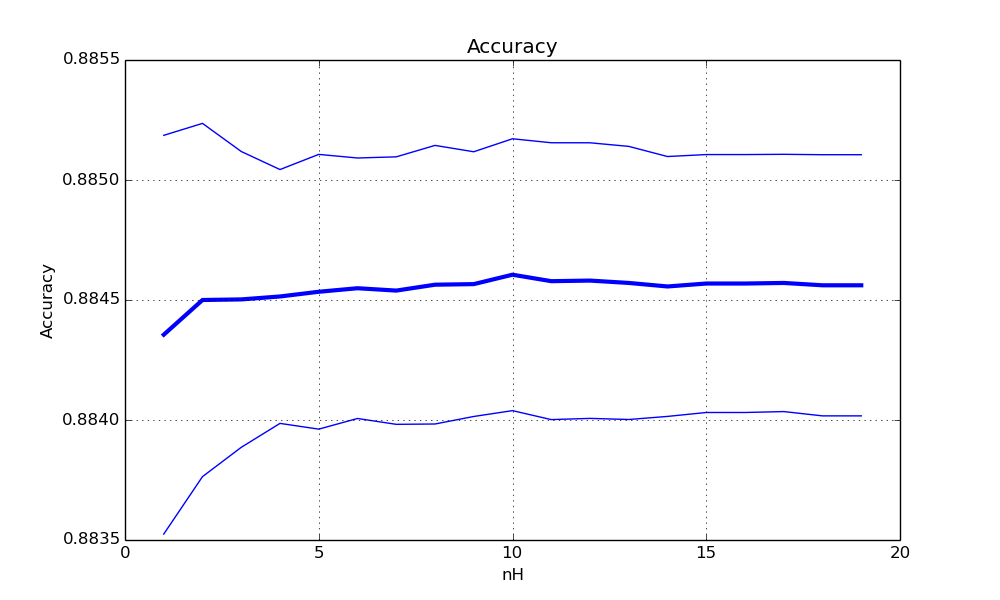
Como no podemos ver como evoluciona el error para todo número de neuronas en una sola gráfica, se ha optamos por esos parámetros que están en un punto intermedio del barrido para que sean representativos a nivel cualitativo. La primera gráfica muestra el accuracy de tr y tst para las 40 realizaciones y la segunda muestra la media y la varianza de gamma, ambas en función del número de capas. La nube de puntos que se ve en la segunda gráfica señala el número de capas donde pararía el criterio de parada utilizado inicialmente para las realizaciones.



Mirando las realizaciones una a una se puede observar que el resultado final de la realización depende mucho de las primeras capas. Las **realizaciones son estables y consistentes**. El número de cambios disminuye conforme aumenta nL. El valor de **gamma** disminuye con el tiempo y se satura, hace overfitting al principio ? A partir del nL = 18 se estanca el error de tst.

### Accuracy y nL en función del criterio de parada.

Dado que disponemos del resultado de todas las realizaciones capa a capa, podemos aplicar “offline” el criterio de parada que deseemos y ver las propiedades del algoritmo respecto a este. Las siguiente gráficas muestra la accuracy y el número de capas obtenido para diferentes tamaño de la ventana ngamma = 1-19.



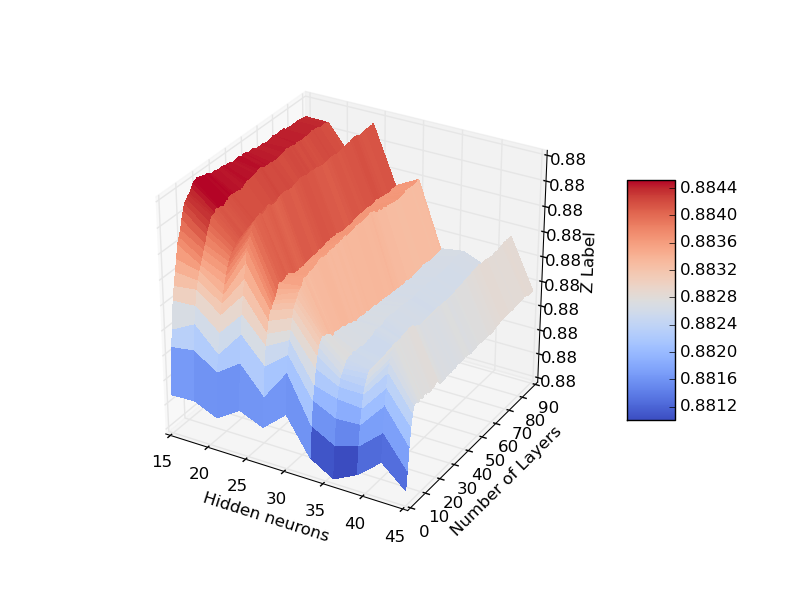
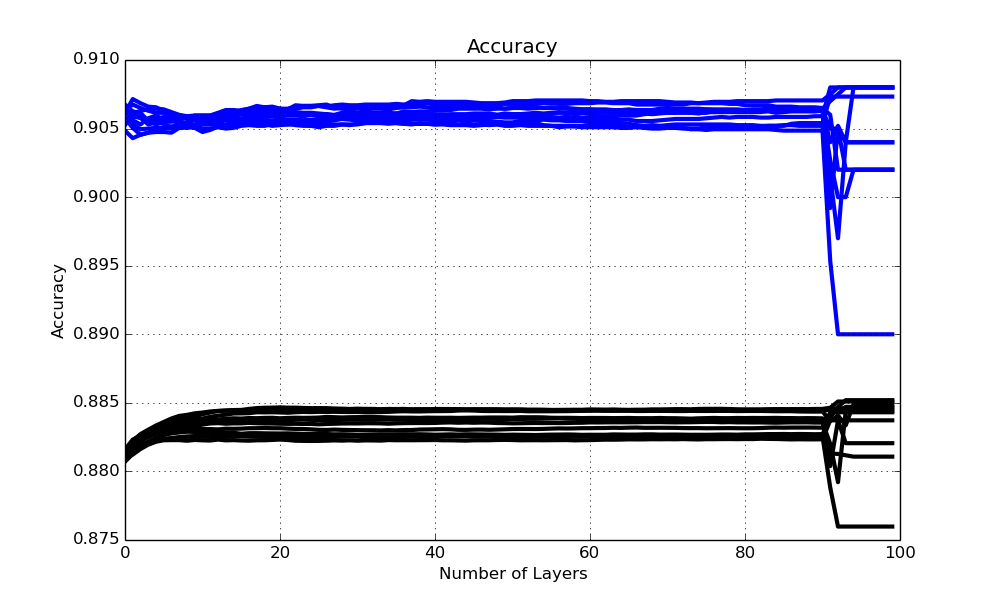
El accuracy no sufre grandes cambios con respecto al tamaño de la ventana. Es un poco menor al principio ya que algunas realizaciones paran en la etapa inicial 0-20 capas donde aún no se ha estabilizado.

El número de capas medio aumenta linealmente con el tamaño de ventana hasta saturarse alrededor de 100 que es el número máximo establecido.

### Estadísticos en función del número de neuronas y capas

A falta de una mejor representación de las características del algoritmo en función del número de capas y de neuronas, presentamos 2 gráficas:

La primera muestra el accuracy de train y test para diferentes número de neuronas ocultas en función del número de capas pero no podemos distinguir claramente cual es cual (aún no está señalizado) pero la segunda gráfica nos proporciona esta información.

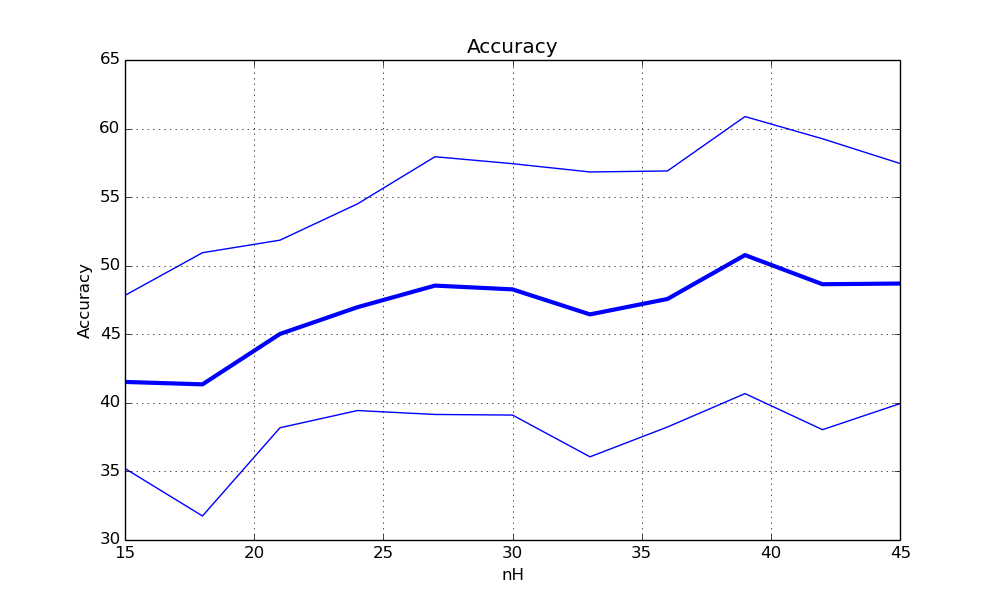
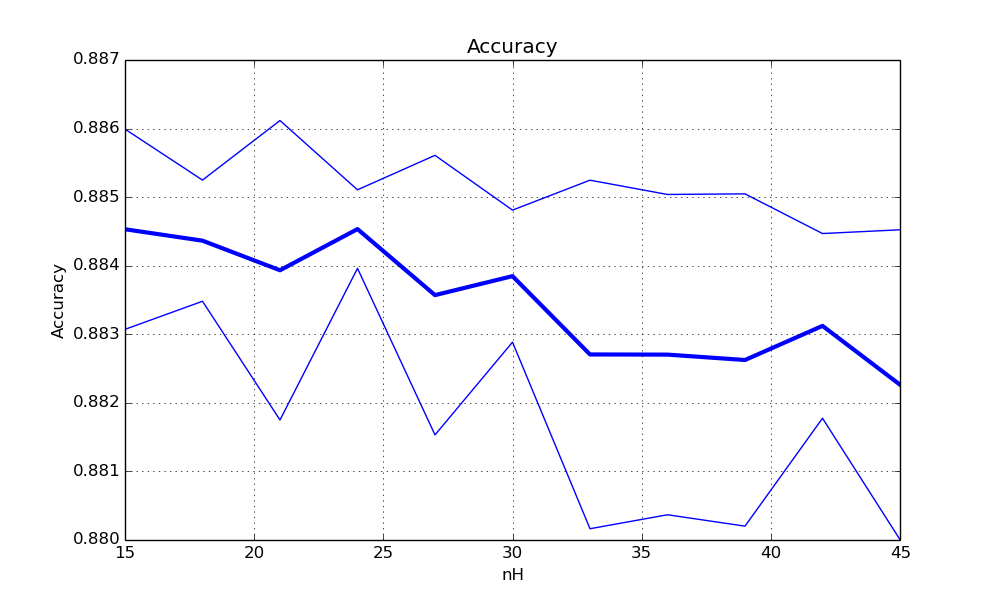


El comportamiento del algoritmo es muy similar independientemente del número de neuronas y los errores de tr y tst parecen estables a lo largo del número de capas, es decir, las curvas no se están entrecruzando todo el rato.

**Nota**: La variación rara que se ve al final es debida a que establecí el número de capas mínimo a 90 y con posibilidad de llegar a 100 si se cumplía el criterio de parada. Debe haber o bien algún error en el código respecto a eso o que las realizaciones inestables tienen mayor varianza.

### Estadísticos en función del número de neuronas

Si aplicamos el criterio de parada inicial (ventana = 5) podemos ver las propiedades del algoritmo en función del número de neuronas, quitándonos la dimensión de las capas. Las siguientes gráficas muestran la accuracy y número medio de capas en función del número de neuronas ocultas utilizando dicho criterio de parada.



Vemos que conforme aumenta el número de neuronas aumenta un poco el error y aumenta la varianza pero no mucho. El número de capas aumenta un poco con el número de neuronas.

## Comentarios generales

# Ima DSNBoost

A continuación se muestran los diversos resultados para la base de datos Image utilizando la estructura DSNBOOST. Se mostrarán 2 tipos de resultados, primero, los resultados experimentales del algoritmo descrito anteriormente utilizando el criterio de parada mencionado anteriormente. Posteriormente se analizan resultados del algoritmo capa a capa con el objetivo de visualizar su comportamiento.



## Resultados experimentales totales

En esta sección se mostrarán los errores de train, validación y test para para diferente número de neuronas y épocas. Mostrándose inicialmente los mejores valores de [Nepoch, Nh] tanto omniscientes como crossvalidados.

### Tabla de Resultados absolutos

Empezaremos dando los resultados finales del algoritmo, indicando tanto para el caso OMN, como para el CV, cuales son los mejores resultados en función de la media del error. Se muestra para combinación [Nepoch, Nh], el error de tr, val y tst y la standard deviation del error de test.

**Error CV:**

[Nepoch = 200 Nh: 16] Tr: 0.00553811257151 Val: 0.036160822816 Tst: 0.0339603960396 std Tst: 0.00558858051155

[Nepoch = 200 Nh: 25] Tr: 0.00542259860703 Val: 0.0366250388319 Tst: 0.0340841584158 std Tst: 0.00420580902164

[Nepoch = 200 Nh: 24] Tr: 0.00542261697372 Val: 0.0367007739077 Tst: 0.0336633663366 std Tst: 0.00408828174694

[Nepoch = 200 Nh: 26] Tr: 0.00592259867815 Val: 0.0367034446345 Tst: 0.0338118811881 std Tst: 0.00441427623989

[Nepoch = 200 Nh: 18] Tr: 0.00544201403876 Val: 0.03685551789 Tst: 0.0344059405941 std Tst: 0.00520508713332

[Nepoch = 200 Nh: 38] Tr: 0.00590314622859 Val: 0.036855819442 Tst: 0.0344059405941 std Tst: 0.0049144729033

[Nepoch = 200 Nh: 41] Tr: 0.00588428513994 Val: 0.0368599592393 Tst: 0.0344554455446 std Tst: 0.00498601121268

[Nepoch = 200 Nh: 30] Tr: 0.00607631555403 Val: 0.0370093594921 Tst: 0.0344059405941 std Tst: 0.00406875282296

**Error OMN:**

[Nepoch = 200 Nh: 12] Tr: 0.00576878939991 Val: 0.0396954643162 Tst: 0.0321287128713 std Tst: 0.00458529165045

[Nepoch = 200 Nh: 5] Tr: 0.00744212580348 Val: 0.0423969790177 Tst: 0.0328465346535 std Tst: 0.00536341075108

[Nepoch = 200 Nh: 27] Tr: 0.00607662992121 Val: 0.0378575832369 Tst: 0.0328712871287 std Tst: 0.0051504185361

[Nepoch = 200 Nh: 28] Tr: 0.00582659269001 Val: 0.0384694402625 Tst: 0.0331435643564 std Tst: 0.00465445270755

[Nepoch = 200 Nh: 20] Tr: 0.00555723250057 Val: 0.0387028506339 Tst: 0.0332425742574 std Tst: 0.00465918912785

[Nepoch = 200 Nh: 29] Tr: 0.00586490637046 Val: 0.0385490135835 Tst: 0.0333910891089 std Tst: 0.00414180735665

[Nepoch = 200 Nh: 34] Tr: 0.00592267267828 Val: 0.0375498932051 Tst: 0.0334405940594 std Tst: 0.00324993496048

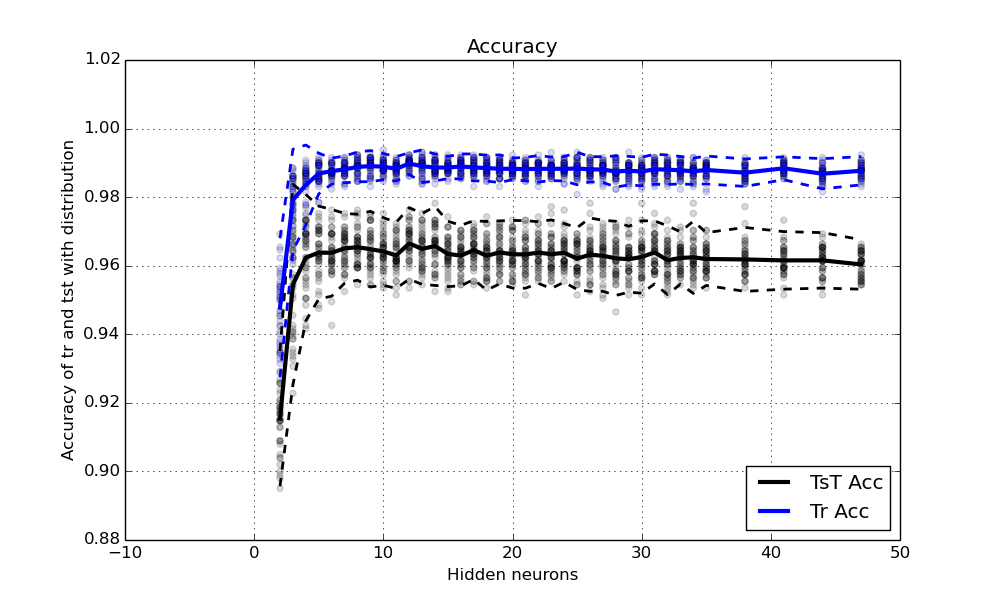
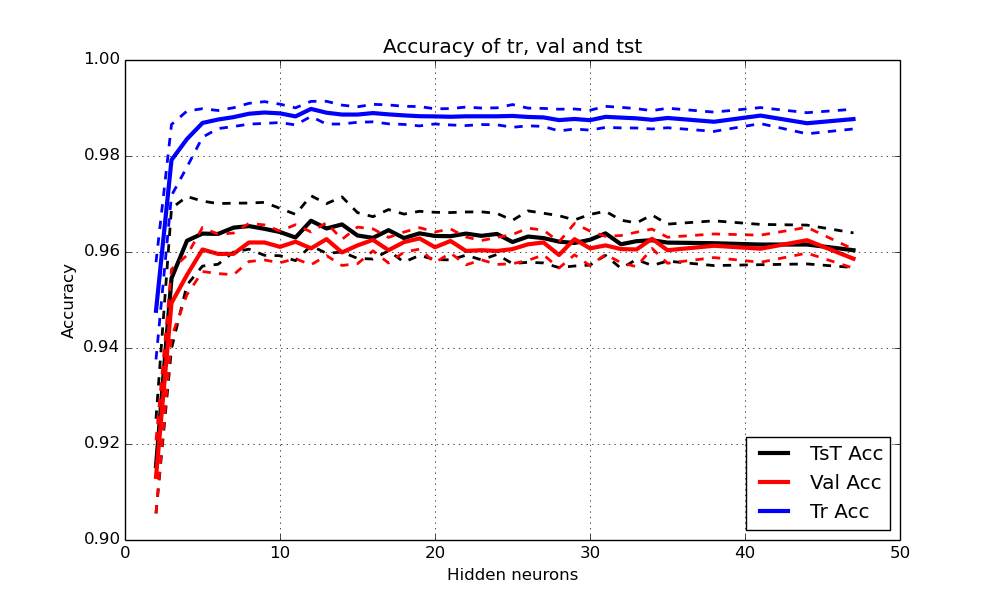
[Nepoch = 200 Nh: 31] Tr: 0.00588413721081 Val: 0.0377815557126 Tst: 0.0334900990099 std Tst: 0.00424640237658

Observaciones:

* Obviamente el error OMN es menor que el error CV.
* El error OMN mejora conforme aumenta el número de épocas mientras que el CV es mejor con menos épocas.
* La variación entre los mejores OMN y CV es de 0.17% de error.

### Resultados para Nepoch = 100

En esta sección se mostrarán diversas cualidades del algoritmo en función del número de neuronas ocultas para 100 épocas:



**Primera figura**: Muestra la evolución de los errores de train, validación y test para 100 épocas en función del número de neuronas.

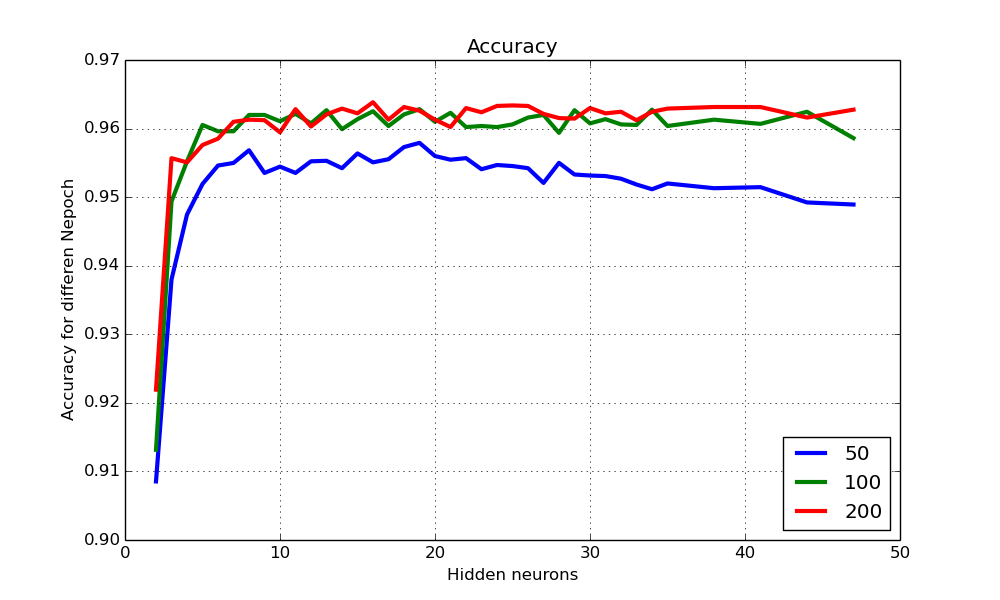
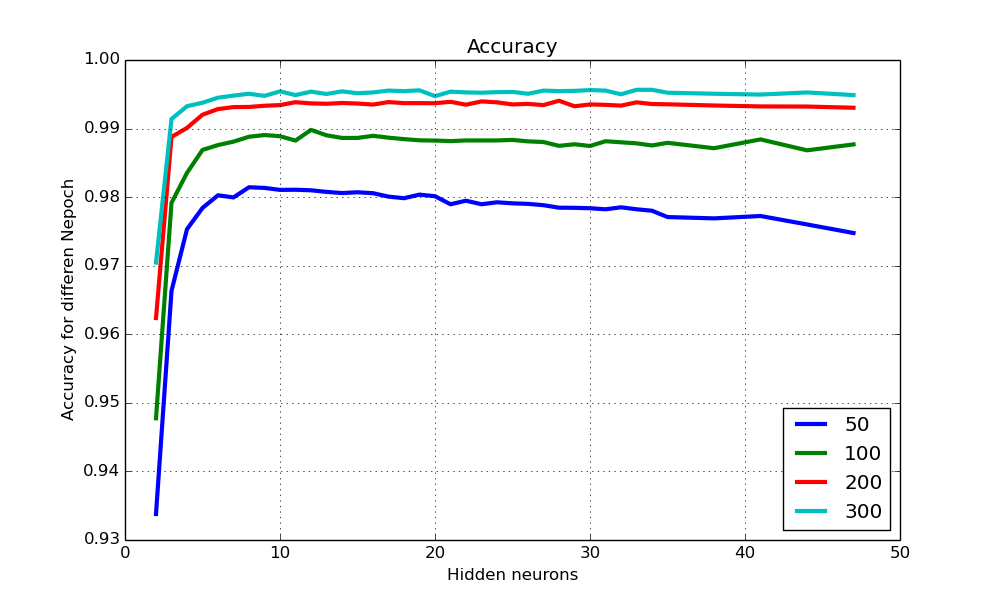
**Segunda figura**: Muestra la evolución de los errores de train y test para 100 épocas en función del número de neuronas mostrando además los valores de las 40 realizaciones para ver cómo se distribuyen.

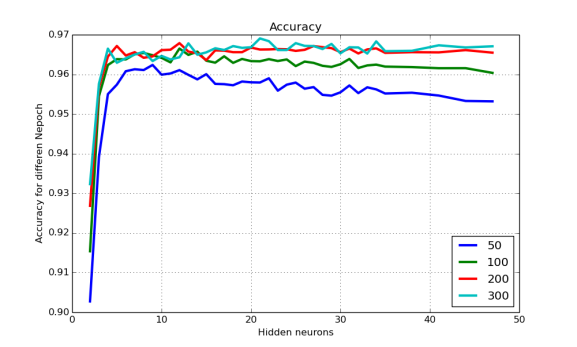
El error omnisciente se satura en Nh = 5.

La distribución de las realizaciones parece correcta. El error de validación y el de test son muy parecidos.

### Resultados para diversos Nepoch

Las siguientes gráficas muestran la evolución de la accuracy de tr, val y tst en función del número de neuronas y el número de épocas:



El error de tr disminuye con Nepoch y aumenta con Nh después de la saturación en Nh = 5

El error de val disminuye con Nh. Falta hacer estudio para mayor Nh.

El error de tst disminuye con Nh depues de satuación.

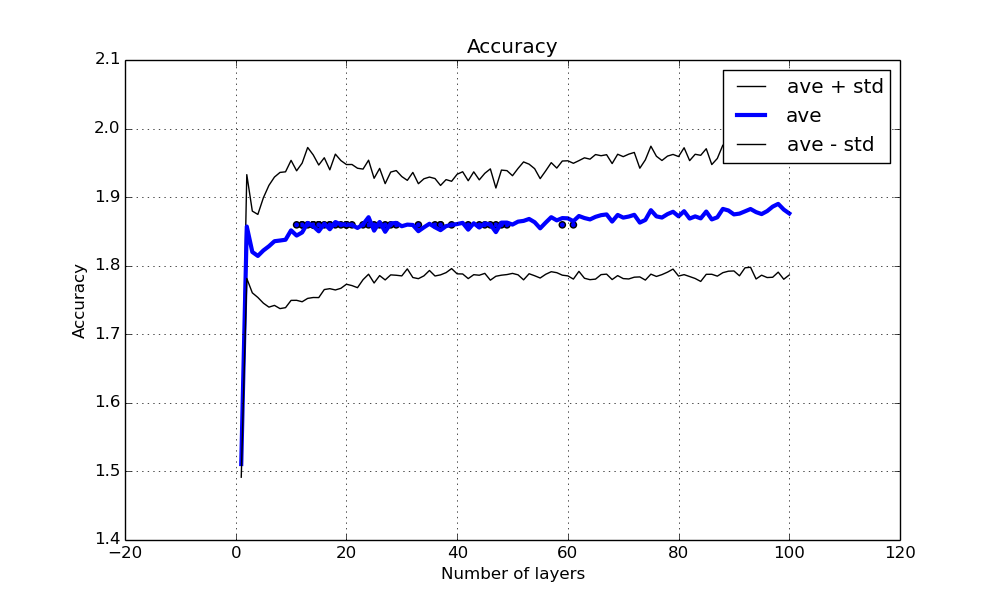
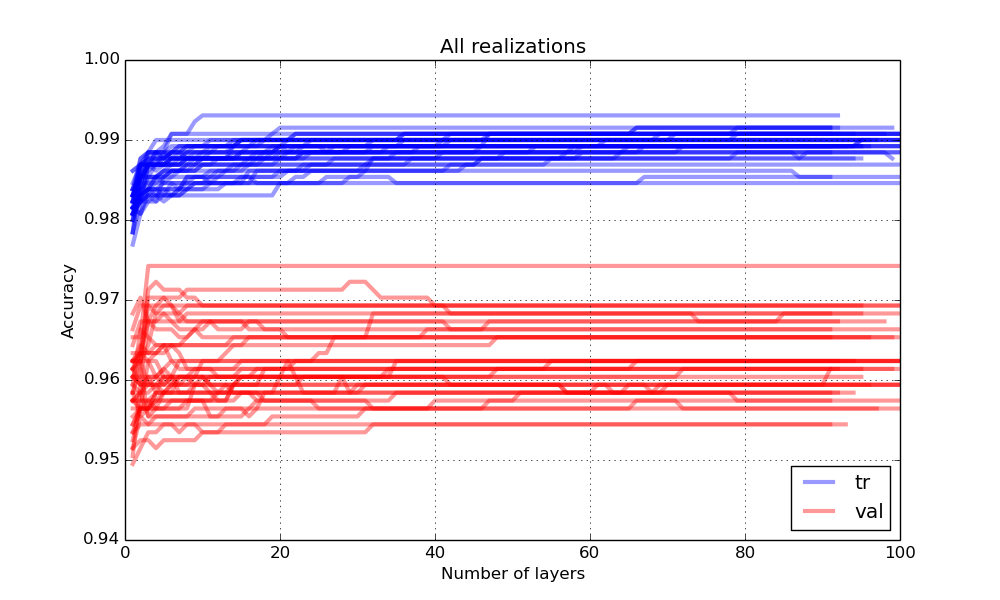
Mejor error CV alcanzado para Nepoch = 200 y mejor error OMN para Nepoch = 200.

## Evolución con el número de capas

Con el objetivo de saber cómo se comporta el algoritmo conforme aumenta el número de capas, se han realizado 40 realizaciones para Nh entre 17 y 47 en pasos de 3 y Nepoch = 50,100,200 del **error omnisciente**. Si bien estos resultados no nos sirven para validar los parámetros, nos dan una idea de las capacidades del algoritmo y de cómo evoluciona para ver cómo deberíamos modificarlo.

### Realizaciones para Nepoch = 100 y Nh = 26

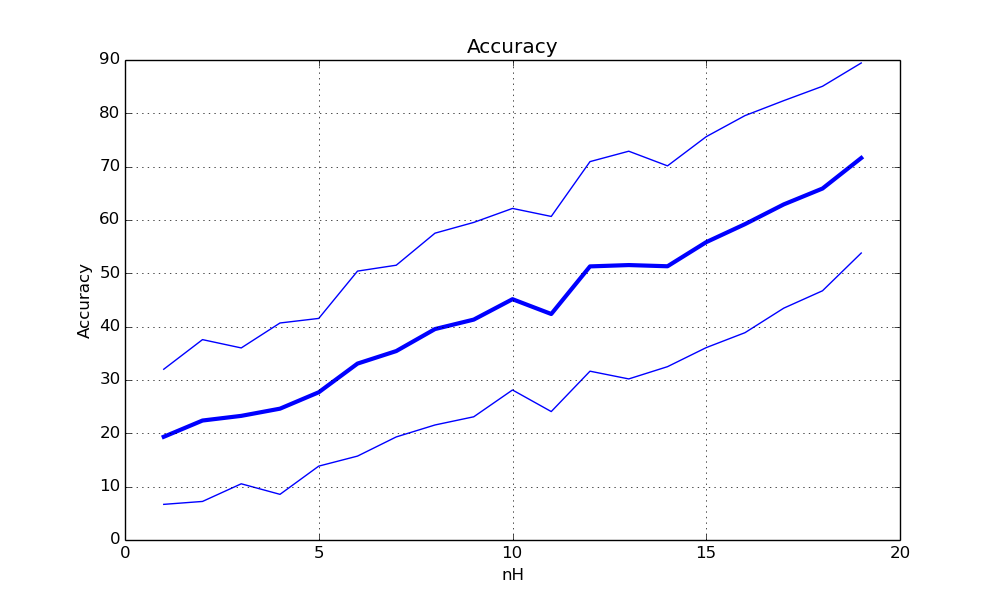
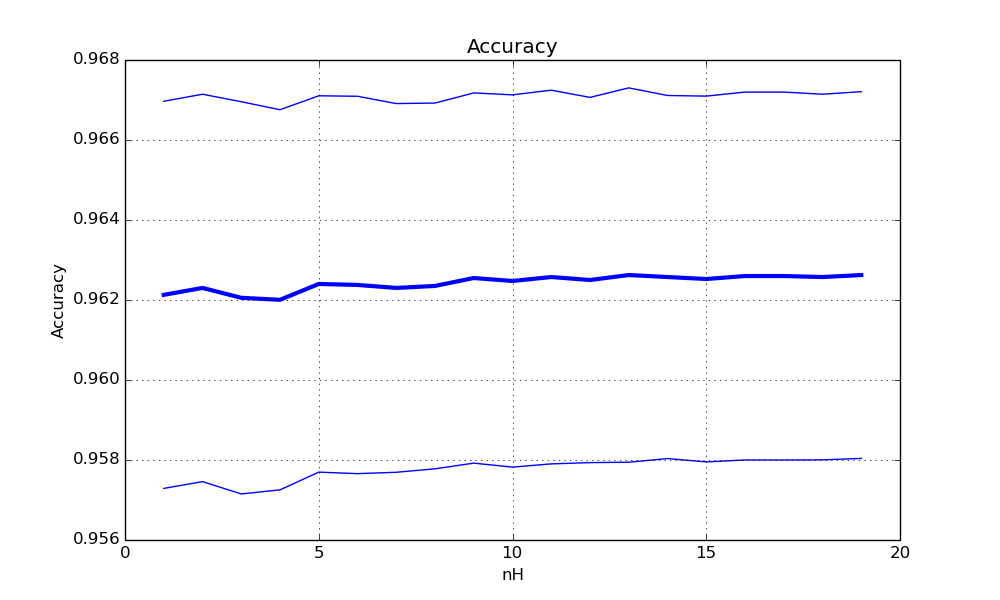
Como no podemos ver como evoluciona el error para todo número de neuronas en una sola gráfica, se ha optamos por esos parámetros que están en un punto intermedio del barrido para que sean representativos a nivel cualitativo. La primera gráfica muestra el accuracy de tr y tst para las 40 realizaciones y la segunda muestra la media y la varianza de gamma, ambas en función del número de capas. La nube de puntos que se ve en la segunda gráfica señala el número de capas donde pararía el criterio de parada utilizado inicialmente para las realizaciones.



Mirando las realizaciones una a una se puede observar que el resultado final de la realización depende mucho de las primeras capas. Las **realizaciones son estables y consistentes**. El número de cambios disminuye conforme aumenta nL. El valor de **gamma** aumenta con el tiempo y se satura. A partir del nL = 18 se estanca el error de tst.

### Accuracy y nL en función del criterio de parada.

Dado que disponemos del resultado de todas las realizaciones capa a capa, podemos aplicar “offline” el criterio de parada que deseemos y ver las propiedades del algoritmo respecto a este. Las siguiente gráficas muestra la accuracy y el número de capas obtenido para diferentes tamaño de la ventana ngamma = 1-19.



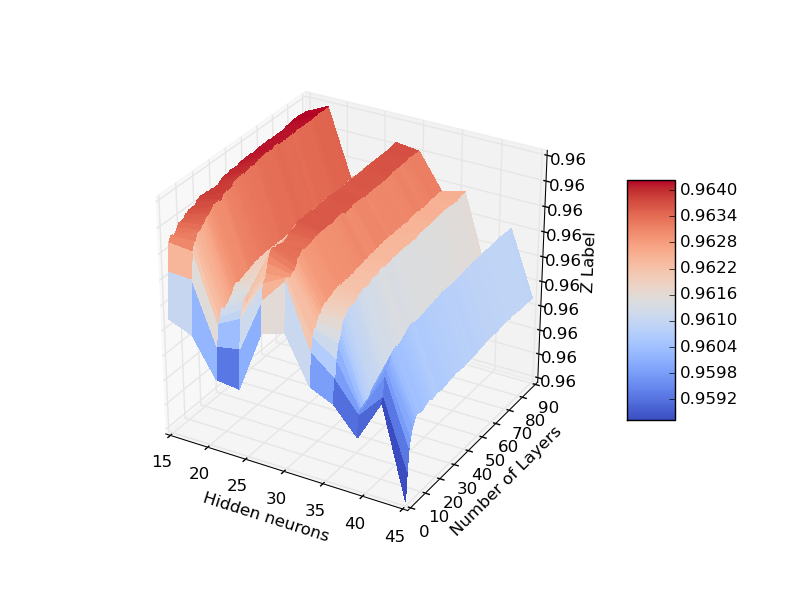
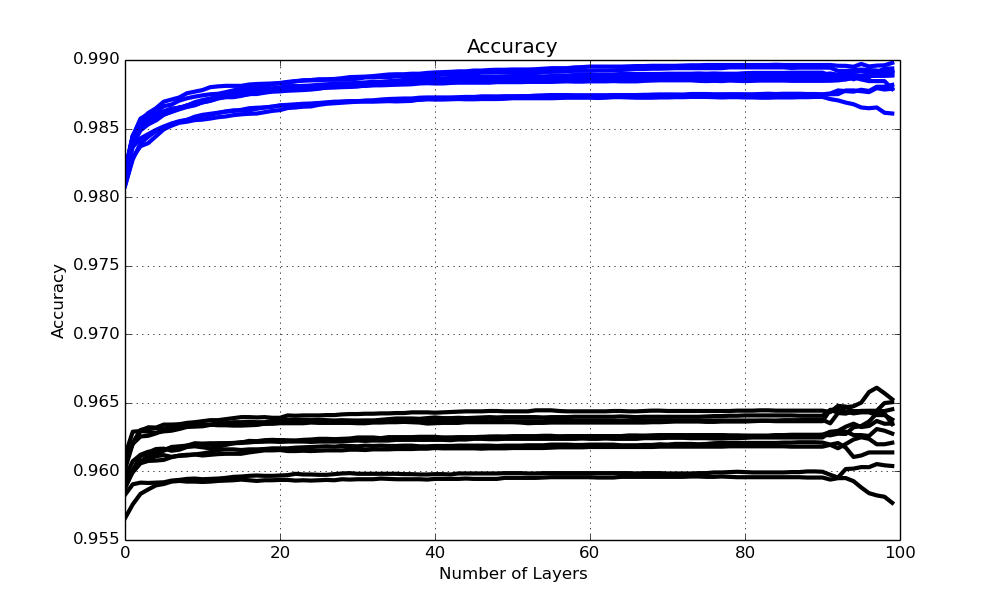
El accuracy no sufre grandes cambios con respecto al tamaño de la ventana.

El número de capas medio aumenta linealmente con el tamaño de ventana hasta saturarse alrededor de 100 que es el número máximo establecido.

### Estadísticos en función del número de neuronas y capas

A falta de una mejor representación de las características del algoritmo en función del número de capas y de neuronas, presentamos 2 gráficas:

La primera muestra el accuracy de train y test para diferentes número de neuronas ocultas en función del número de capas pero no podemos distinguir claramente cual es cual (aún no está señalizado) pero la segunda gráfica nos proporciona esta información.

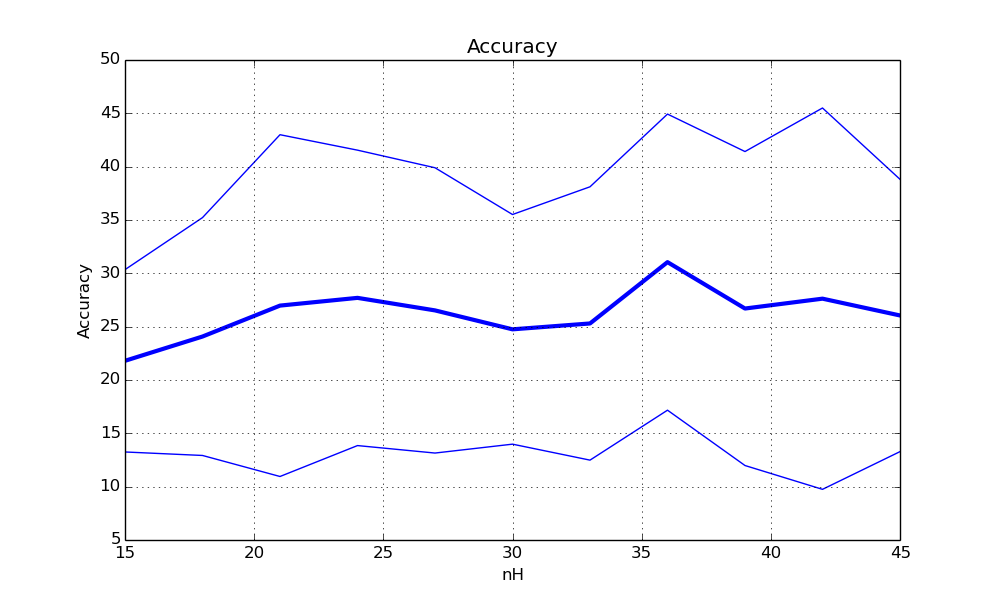
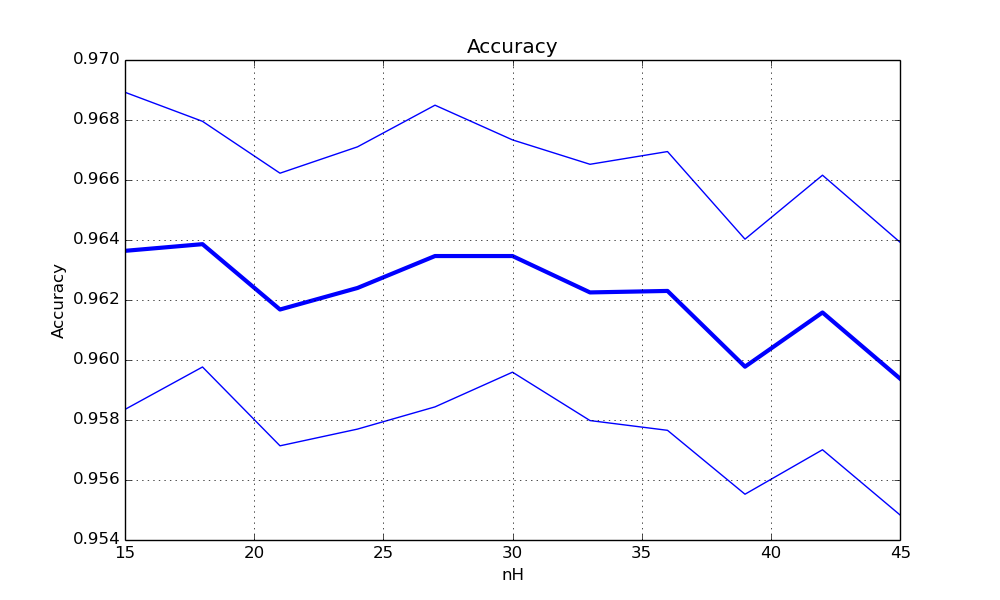


El comportamiento del algoritmo es muy similar independientemente del número de neuronas y los errores de tr y tst parecen estables a lo largo del número de capas, es decir, las curvas no se están entrecruzando todo el rato.

**Nota**: La variación rara que se ve al final es debida a que establecí el número de capas mínimo a 90 y con posibilidad de llegar a 100 si se cumplía el criterio de parada. Debe haber o bien algún error en el código respecto a eso o que las realizaciones inestables tienen mayor varianza.

### Estadísticos en función del número de neuronas

Si aplicamos el criterio de parada inicial (ventana = 5) podemos ver las propiedades del algoritmo en función del número de neuronas, quitándonos la dimensión de las capas. Las siguientes gráficas muestran la accuracy y número medio de capas en función del número de neuronas ocultas utilizando dicho criterio de parada.



Vemos que conforme aumenta el número de neuronas aumenta un poco el error y aumenta la varianza pero no mucho. El número de capas aumenta un poco con el número de neuronas.

## Comentarios generales

# Wav DSNBoost

A continuación se muestran los diversos resultados para la base de datos Waveform utilizando la estructura DSNBOOST. Se mostrarán 2 tipos de resultados, primero, los resultados experimentales del algoritmo descrito anteriormente utilizando el criterio de parada mencionado anteriormente. Posteriormente se analizan resultados del algoritmo capa a capa con el objetivo de visualizar su comportamiento.



## Resultados experimentales totales

En esta sección se mostrarán los errores de train, validación y test para para diferente número de neuronas y épocas. Mostrándose inicialmente los mejores valores de [Nepoch, Nh] tanto omniscientes como crossvalidados.

### Tabla de Resultados absolutos

Empezaremos dando los resultados finales del algoritmo, indicando tanto para el caso OMN, como para el CV, cuales son los mejores resultados en función de la media del error. Se muestra para combinación [Nepoch, Nh], el error de tr, val y tst y la standard deviation del error de test.

**Error CV:**

[Nepoch = 50 Nh: 38] Tr: 0.00731389588766 Val: 0.106457337084 Tst: 0.11260326087 std Tst: 0.00175703252098

[Nepoch = 50 Nh: 24] Tr: 0.00737678773718 Val: 0.106707102672 Tst: 0.113 std Tst: 0.00160575524458

[Nepoch = 50 Nh: 32] Tr: 0.00737659181242 Val: 0.107454172527 Tst: 0.112668478261 std Tst: 0.00188202841494

[Nepoch = 50 Nh: 47] Tr: 0.00737620118361 Val: 0.107479410845 Tst: 0.112434782609 std Tst: 0.00173109556971

[Nepoch = 50 Nh: 30] Tr: 0.00718889588766 Val: 0.107706868261 Tst: 0.112788043478 std Tst: 0.00185668433002

[Nepoch = 50 Nh: 35] Tr: 0.00725159181242 Val: 0.1079697609 Tst: 0.112951086957 std Tst: 0.00151776460775

[Nepoch = 50 Nh: 44] Tr: 0.00756409303313 Val: 0.108229332708 Tst: 0.112510869565 std Tst: 0.00202299516851

[Nepoch = 50 Nh: 28] Tr: 0.00750198366195 Val: 0.108235427411 Tst: 0.112690217391 std Tst: 0.0015476730353

**Error OMN:**

[Nepoch = 50 Nh: 41] Tr: 0.00737659181242 Val: 0.1089697609 Tst: 0.111668478261 std Tst: 0.00170647846802

[Nepoch = 50 Nh: 34] Tr: 0.00750198366195 Val: 0.111454485076 Tst: 0.11225 std Tst: 0.00168078066073

[Nepoch = 50 Nh: 47] Tr: 0.00737620118361 Val: 0.107479410845 Tst: 0.112434782609 std Tst: 0.00173109556971

[Nepoch = 50 Nh: 44] Tr: 0.00756409303313 Val: 0.108229332708 Tst: 0.112510869565 std Tst: 0.00202299516851

[Nepoch = 50 Nh: 14] Tr: 0.00700100403813 Val: 0.110247694952 Tst: 0.112581521739 std Tst: 0.00153733275944

[Nepoch = 50 Nh: 38] Tr: 0.00731389588766 Val: 0.106457337084 Tst: 0.11260326087 std Tst: 0.00175703252098

[Nepoch = 50 Nh: 29] Tr: 0.00718889588766 Val: 0.110482419128 Tst: 0.112630434783 std Tst: 0.00163824217658

[Nepoch = 50 Nh: 32] Tr: 0.00737659181242 Val: 0.107454172527 Tst: 0.112668478261 std Tst: 0.00188202841494

[Nepoch = 50 Nh: 28] Tr: 0.00750198366195 Val: 0.108235427411 Tst: 0.112690217391 std Tst: 0.0015476730353

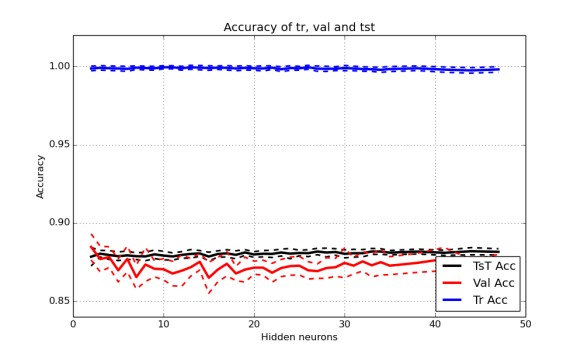
[Nepoch = 50 Nh: 27] Tr: 0.00737698366195 Val: 0.109726168151 Tst: 0.112701086957 std Tst: 0.00155695850989

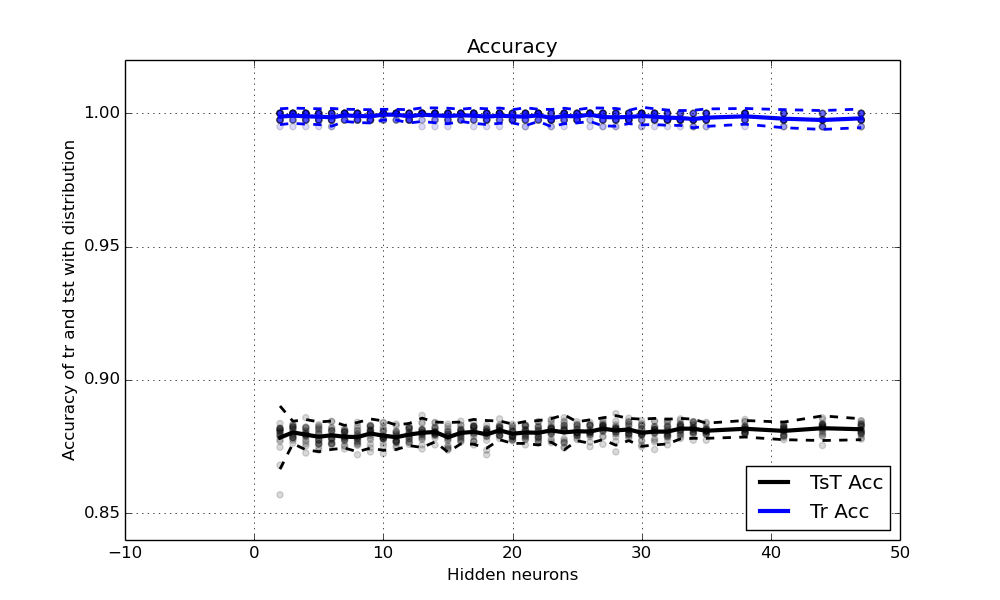
Observaciones:

* Obviamente el error OMN es menor que el error CV.
* La variación entre los mejores OMN y CV es de 0.1% de error.

### Resultados para Nepoch = 100

En esta sección se mostrarán diversas cualidades del algoritmo en función del número de neuronas ocultas para 100 épocas:





**Primera figura**: Muestra la evolución de los errores de train, validación y test para 100 épocas en función del número de neuronas.

**Segunda figura**: Muestra la evolución de los errores de train y test para 100 épocas en función del número de neuronas mostrando además los valores de las 40 realizaciones para ver cómo se distribuyen.

El error omnisciente se satura en Nh = 2.

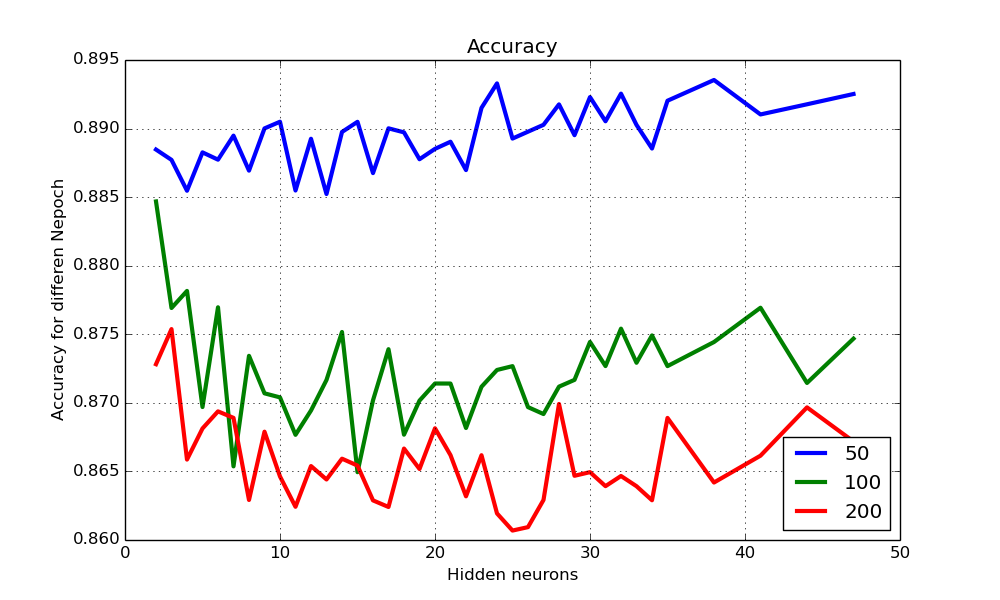
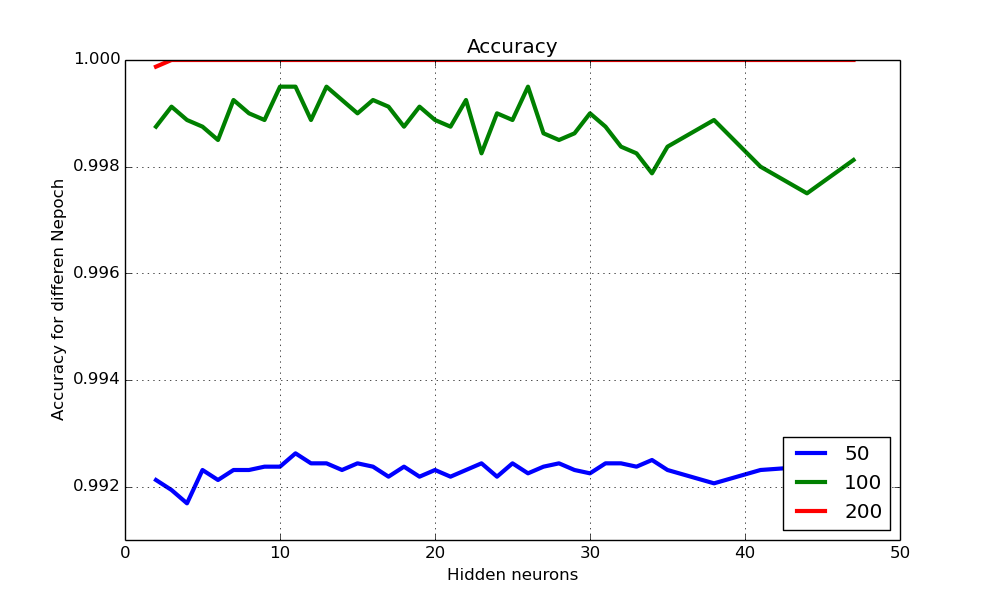
El error de validación crece un poco con Nh

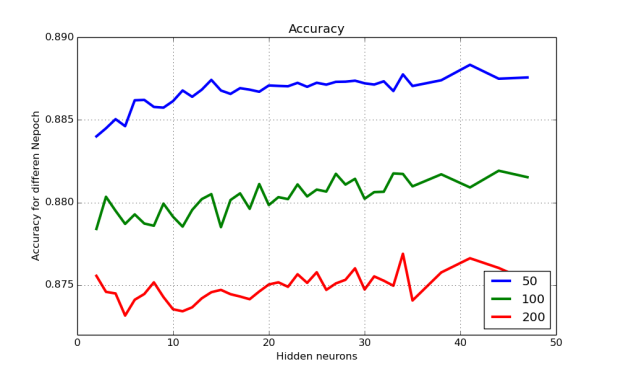
El error de tst es prácticamente 0. Overfitting áun con 2 neuronas ?

La distribución de las realizaciones parece correcta.

### Resultados para diversos Nepoch

Las siguientes gráficas muestran la evolución de la accuracy de tr, val y tst en función del número de neuronas y el número de épocas:



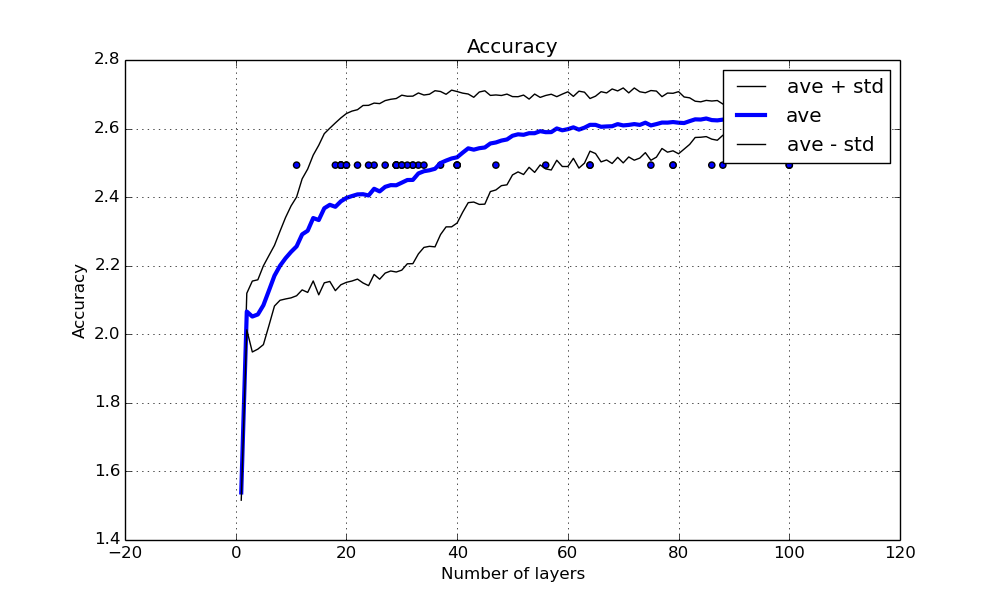
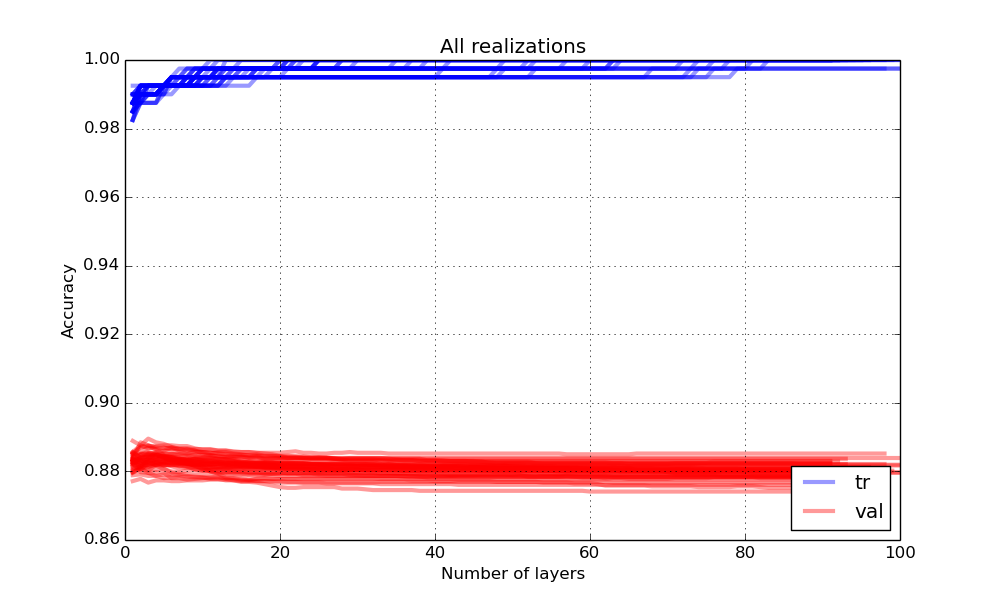
Esta base de datos necesita muy pocas épocas, si no, hace overfitting. Probablemente el mejor Nepoch es menor a 50 y daría mejores resultados. Para Nepoch = 200, el overfitting es brutal. Con pocas épocas parece que el error de validación y test disminuye con el número de neuronas.

## Evolución con el número de capas

Con el objetivo de saber cómo se comporta el algoritmo conforme aumenta el número de capas, se han realizado 40 realizaciones para Nh entre 17 y 47 en pasos de 3 y Nepoch = 50,100,200 del **error omnisciente**. Si bien estos resultados no nos sirven para validar los parámetros, nos dan una idea de las capacidades del algoritmo y de cómo evoluciona para ver cómo deberíamos modificarlo.

### Realizaciones para Nepoch = 100 y Nh = 26

Como no podemos ver como evoluciona el error para todo número de neuronas en una sola gráfica, se ha optamos por esos parámetros que están en un punto intermedio del barrido para que sean representativos a nivel cualitativo. La primera gráfica muestra el accuracy de tr y tst para las 40 realizaciones y la segunda muestra la media y la varianza de gamma, ambas en función del número de capas. La nube de puntos que se ve en la segunda gráfica señala el número de capas donde pararía el criterio de parada utilizado inicialmente para las realizaciones.

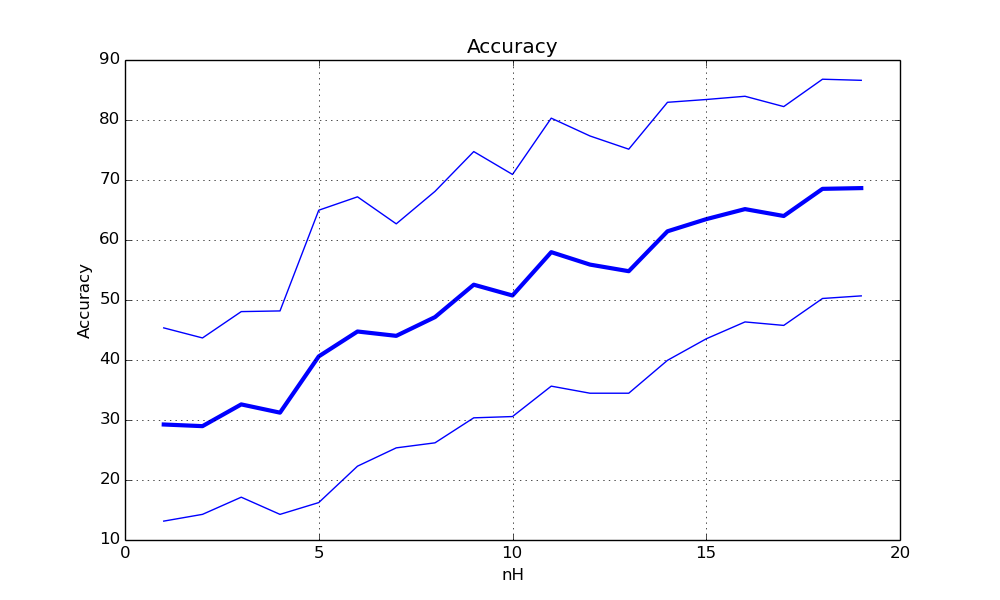


El error de test desciende un poco con nL, a causa probablemente de un ligero overfitting. Las **realizaciones son estables y consistentes**. El número de cambios disminuye conforme aumenta nL. El valor de **gamma** aumenta con el tiempo y se satura. A partir del nL = 18 se estanca el error de tst.

### Accuracy y nL en función del criterio de parada.

Dado que disponemos del resultado de todas las realizaciones capa a capa, podemos aplicar “offline” el criterio de parada que deseemos y ver las propiedades del algoritmo respecto a este. Las siguiente gráficas muestra la accuracy y el número de capas obtenido para diferentes tamaño de la ventana ngamma = 1-19.





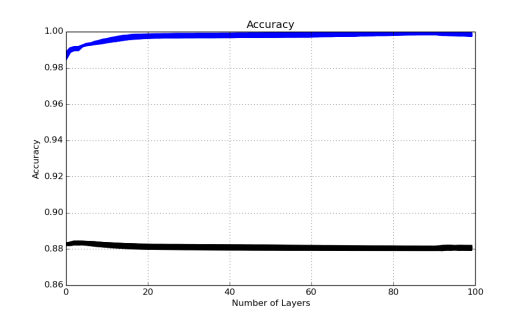
El accuracy no sufre grandes cambios con respecto al tamaño de la ventana. Al principio es mayor ya que después hace overfitting.

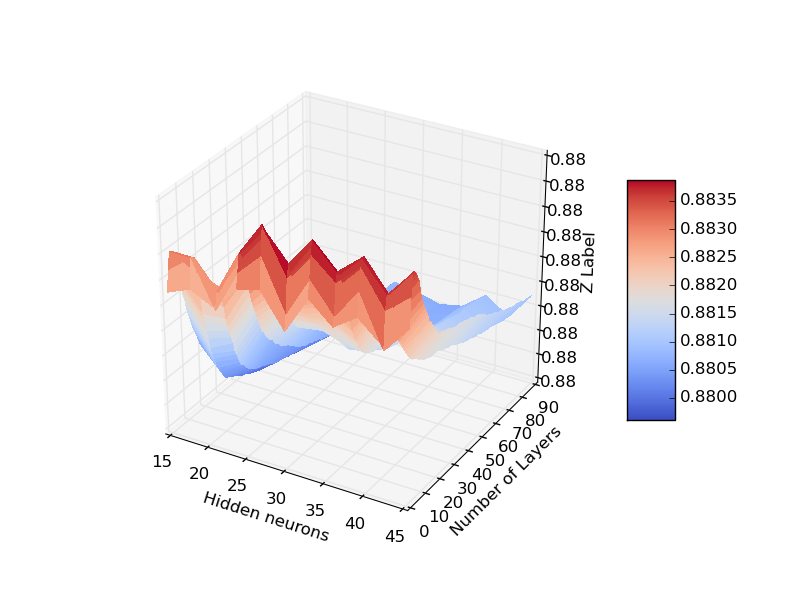
El número de capas medio aumenta linealmente con el tamaño de ventana hasta saturarse alrededor de 100 que es el número máximo establecido.

### Estadísticos en función del número de neuronas y capas

A falta de una mejor representación de las características del algoritmo en función del número de capas y de neuronas, presentamos 2 gráficas:

La primera muestra el accuracy de train y test para diferentes número de neuronas ocultas en función del número de capas pero no podemos distinguir claramente cual es cual (aún no está señalizado) pero la segunda gráfica nos proporciona esta información.

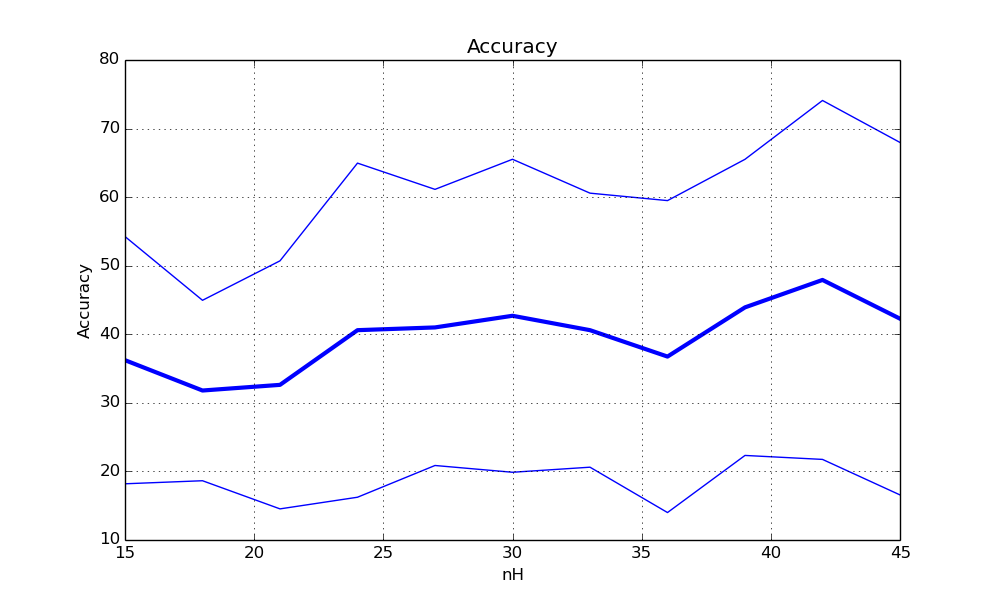
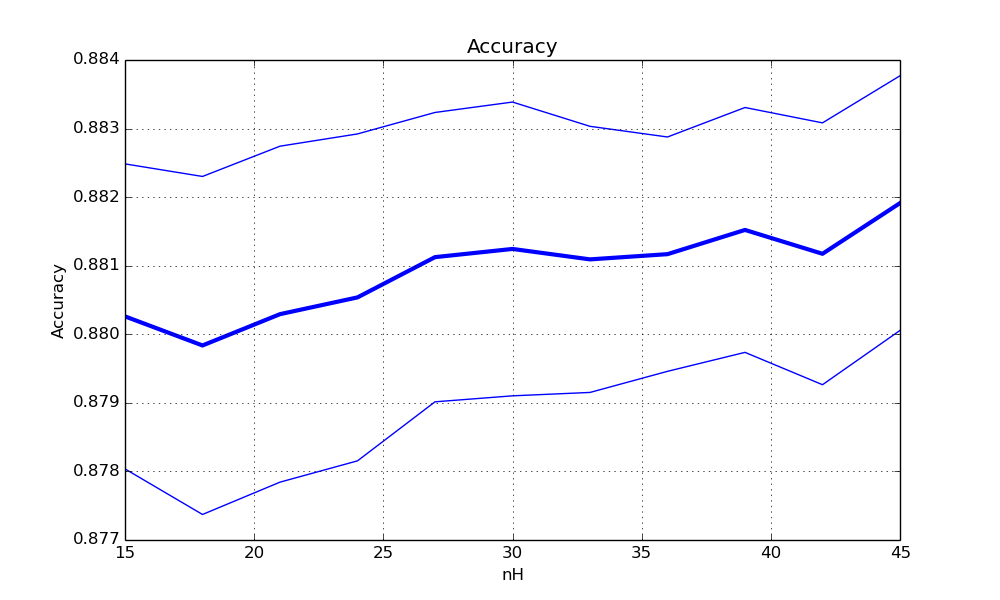




El comportamiento del algoritmo es muy similar independientemente del número de neuronas y los errores de tr y tst parecen estables a lo largo del número de capas, es decir, las curvas no se están entrecruzando todo el rato. Podemos observar que el overfitting conforme avanza el número de capas y disminuye con el número de neuronas.

### Estadísticos en función del número de neuronas

Si aplicamos el criterio de parada inicial (ventana = 5) podemos ver las propiedades del algoritmo en función del número de neuronas, quitándonos la dimensión de las capas. Las siguientes gráficas muestran la accuracy y número medio de capas en función del número de neuronas ocultas utilizando dicho criterio de parada.



Vemos que conforme aumenta el número de neuronas disminuye un poco el error, menos overfitting. El número de capas aumenta un poco con el número de neuronas.

## Comentarios generales

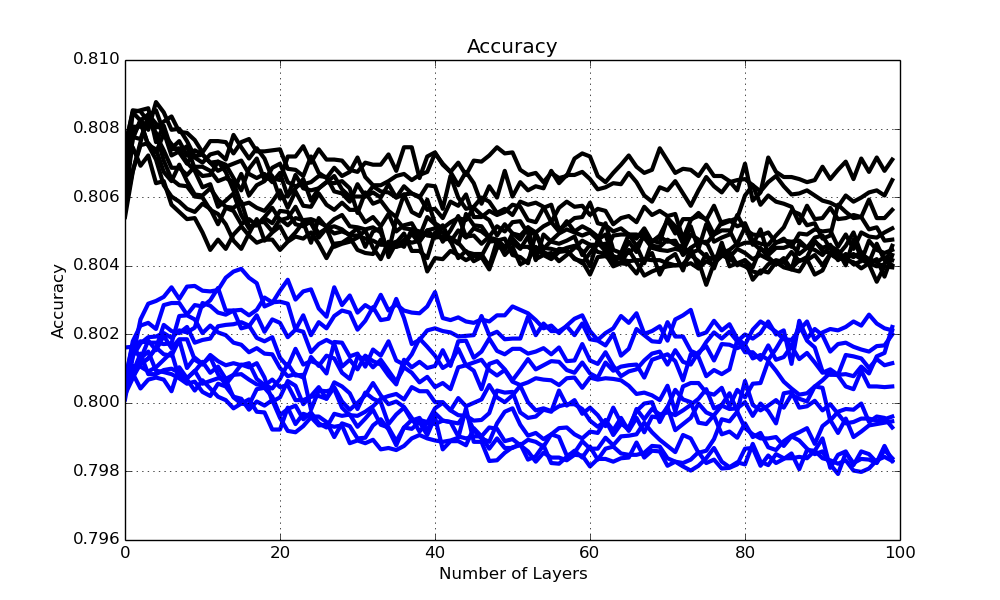
# DSN

Existen ciertas características del algoritmo DSN que deberíamos tener en cuenta antes de evaluar su performance. En el paper original de Microsoft entrenan de la siguiente manera:

* Función de activación de la capa de **salida** **lineal**.
* Inyectan la concatenación de todas las **salidas anteriores**. Como la salida es lineal, las activaciones y las salidas son lo mismo.
* Pre-entrenan cada capa mediante una RBM. Algoritmo generativo no supervisado que en teoría aprende la distribución de las entradas. Con él inicializan los pesos de la capa oculta.
* Después calculan la salida como solución de mínimos cuadrados respecto a la salida de la capa oculta. Después de eso, también prueban hacer entrenamiento Batch-mode (Fine tuning).
* No especifican en ningún momento los parámetros de entrenamiento ni de la RBM ni del Batch-Mode Back-Propagation.

Nosotros simplemente entrenamos cada capa del DSN mediante Back-Propagation con los parámetros descritos al principio de este report. Esto crea el siguiente problema:

* Conforme aumente el número de capas, aumentará el número de inyecciones. Manteniendo el mismo tipo de entrenamiento, el sistema caerá en underfitting. Se puede ver en la siguiente gráfica que muestra las medias del algoritmo para diferentes números de neuronas y 100 épocas en función del número de capas.



Por ello se ha tomado la decisión de establecer una **ventana de inyección** de longitud , es decir, solo se inyectan las últimas activaciones. De esta forma la dimensionalidad de la entrada se satura y no se produce underfitting.

Si bien el tamaño de ventana podría ser crossvalidado se ha elegido un valor de compromiso para todas las ventanas. Un valor lo suficientemente grande como para mostrar las características del algoritmo inicial del DSN pero que sea escalable con grandes números de capas.

No nos entretendremos tanto mostrando todas las gráficas homólogas para esta estructura, simplemente destacaremos aquellas que difieran del DSNBoost.

# Resultados DSN L = 10

Empezaré mostrando los resultados del error CV y OMN de este algoritmo para las mismas bases de datos:

## Abalone

**Error CV:**

[Nepoch = 50 Nh: 26] Tr: 0.197416618482 Val: 0.202306583009 Tst: 0.189895209581 std Tst: 0.0019412075571

[Nepoch = 50 Nh: 30] Tr: 0.197566264263 Val: 0.202624670341 Tst: 0.189805389222 std Tst: 0.00171725575695

[Nepoch = 50 Nh: 47] Tr: 0.197925376458 Val: 0.202704990656 Tst: 0.190134730539 std Tst: 0.00154988927486

[Nepoch = 50 Nh: 23] Tr: 0.197097595456 Val: 0.202706104604 Tst: 0.190419161677 std Tst: 0.00189357943723

[Nepoch = 50 Nh: 44] Tr: 0.197915436159 Val: 0.202745068588 Tst: 0.190134730539 std Tst: 0.00190766968542

[Nepoch = 50 Nh: 21] Tr: 0.19663899111 Val: 0.202823716074 Tst: 0.190164670659 std Tst: 0.00153185485491

[Nepoch = 50 Nh: 31] Tr: 0.197446662962 Val: 0.20294379496 Tst: 0.190149700599 std Tst: 0.0018453900726

[Nepoch = 50 Nh: 34] Tr: 0.197516463585 Val: 0.202944033527 Tst: 0.190239520958 std Tst: 0.00191803800935

**Error OMN:**

[Nepoch = 200 Nh: 14] Tr: 0.191862382862 Val: 0.209168408999 Tst: 0.189805389222 std Tst: 0.00246706405151

[Nepoch = 50 Nh: 30] Tr: 0.197566264263 Val: 0.202624670341 Tst: 0.189805389222 std Tst: 0.00171725575695

[Nepoch = 50 Nh: 33] Tr: 0.197795685739 Val: 0.203862837512 Tst: 0.189835329341 std Tst: 0.00208932916349

[Nepoch = 50 Nh: 26] Tr: 0.197416618482 Val: 0.202306583009 Tst: 0.189895209581 std Tst: 0.0019412075571

[Nepoch = 50 Nh: 18] Tr: 0.19715751041 Val: 0.203622201971 Tst: 0.19001497006 std Tst: 0.00217303194647

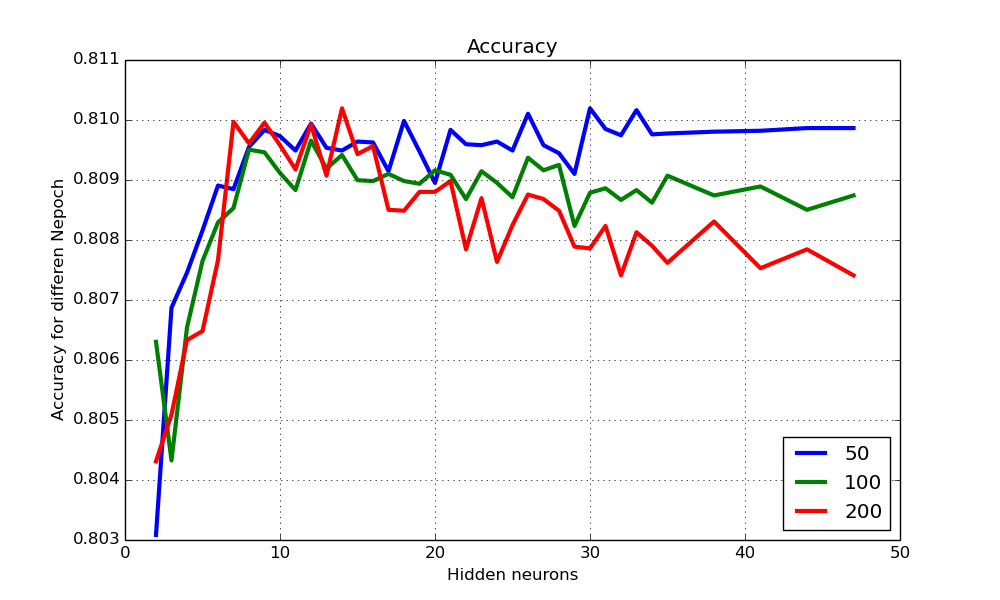
[Nepoch = 200 Nh: 7] Tr: 0.191224441057 Val: 0.211000283099 Tst: 0.19002994012 std Tst: 0.00275562980885

[Nepoch = 200 Nh: 9] Tr: 0.19086497599 Val: 0.208567535845 Tst: 0.19004491018 std Tst: 0.00282989570971

[Nepoch = 50 Nh: 12] Tr: 0.197476419214 Val: 0.203821883882 Tst: 0.19005988024 std Tst: 0.00155458143542

**Gráficas de Validación y Test:**





Podría ser que con **menos épocas**, el error de validación y test fuera menor !

Requiere **menos entrenamiento que el DSNBoost.**

**Misma performance** que el DSNBoost.

## Kwok

**Error CV:**

[Nepoch = 100 Nh: 32] Tr: 0.0936 Val: 0.0966 Tst: 0.117916666667 std Tst: 0.00131455746263

[Nepoch = 100 Nh: 31] Tr: 0.09385 Val: 0.0968 Tst: 0.117485294118 std Tst: 0.00109907820242

[Nepoch = 100 Nh: 23] Tr: 0.093 Val: 0.0976 Tst: 0.117242647059 std Tst: 0.000690528656225

[Nepoch = 100 Nh: 30] Tr: 0.0933 Val: 0.0984 Tst: 0.118112745098 std Tst: 0.00193720503826

[Nepoch = 100 Nh: 24] Tr: 0.09235 Val: 0.0986 Tst: 0.117362745098 std Tst: 0.000593035966072

[Nepoch = 100 Nh: 21] Tr: 0.09215 Val: 0.0988 Tst: 0.117892156863 std Tst: 0.00170655835152

[Nepoch = 100 Nh: 29] Tr: 0.0934 Val: 0.099 Tst: 0.117475490196 std Tst: 0.000971219343242

[Nepoch = 100 Nh: 28] Tr: 0.0921 Val: 0.099 Tst: 0.11762745098 std Tst: 0.0012288581172

[Nepoch = 100 Nh: 25] Tr: 0.09215 Val: 0.099 Tst: 0.117661764706 std Tst: 0.00144046767951

[Nepoch = 200 Nh: 47] Tr: 0.0898 Val: 0.0992 Tst: 0.117522058824 std Tst: 0.000659122357241

[Nepoch = 100 Nh: 38] Tr: 0.09485 Val: 0.0992 Tst: 0.118026960784 std Tst: 0.00160395085728

[Nepoch = 100 Nh: 22] Tr: 0.0932 Val: 0.0992 Tst: 0.117178921569 std Tst: 0.000459053643904

**Error OMN:**

[Nepoch = 100 Nh: 4] Tr: 0.09235 Val: 0.1048 Tst: 0.116590686275 std Tst: 0.000519463307682

[Nepoch = 100 Nh: 6] Tr: 0.09165 Val: 0.1028 Tst: 0.116801470588 std Tst: 0.000654989830609

[Nepoch = 100 Nh: 5] Tr: 0.0919 Val: 0.1016 Tst: 0.116875 std Tst: 0.000600440173987

[Nepoch = 100 Nh: 10] Tr: 0.09145 Val: 0.1012 Tst: 0.116904411765 std Tst: 0.000457690647235

[Nepoch = 100 Nh: 7] Tr: 0.09105 Val: 0.106 Tst: 0.116906862745 std Tst: 0.000629649182514

[Nepoch = 100 Nh: 12] Tr: 0.0903 Val: 0.1024 Tst: 0.116963235294 std Tst: 0.000454318157755

[Nepoch = 100 Nh: 11] Tr: 0.0913 Val: 0.1004 Tst: 0.116965686275 std Tst: 0.000566998172929

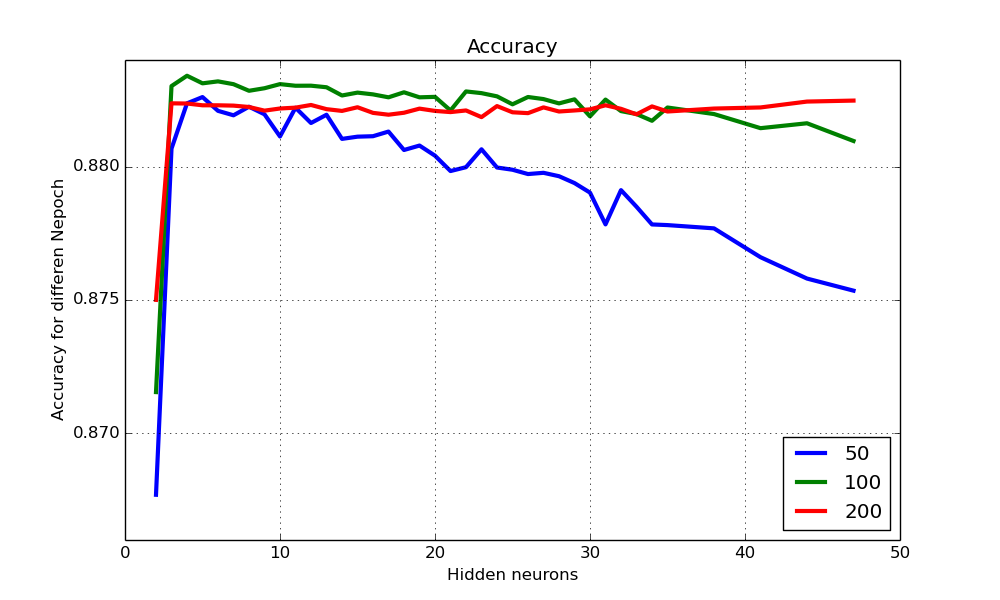
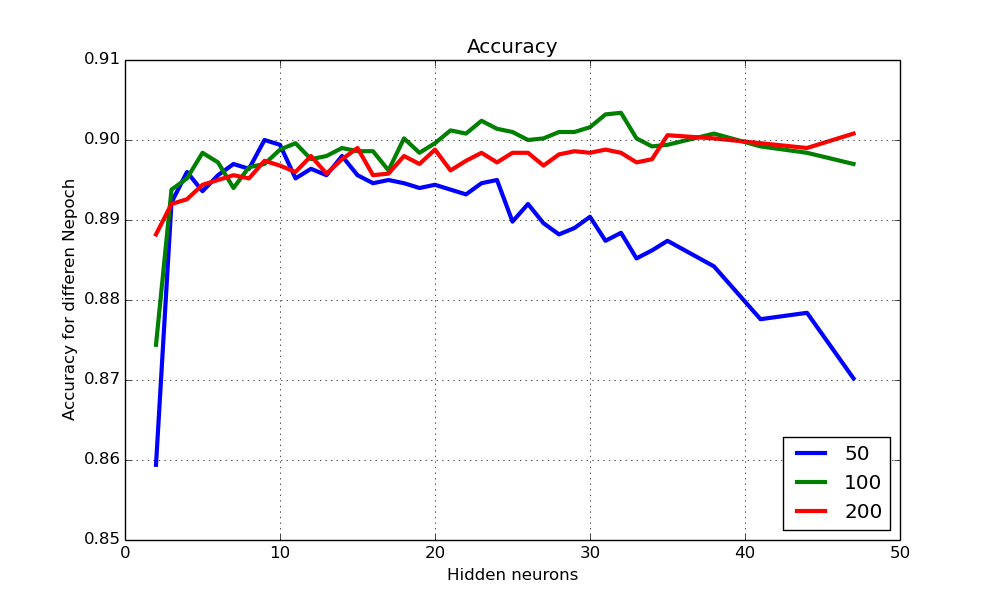
[Nepoch = 100 Nh: 3] Tr: 0.09345 Val: 0.1062 Tst: 0.116977941176 std Tst: 0.00114245925055

[Nepoch = 100 Nh: 13] Tr: 0.09185 Val: 0.102 Tst: 0.117022058824 std Tst: 0.000390737320453

[Nepoch = 100 Nh: 9] Tr: 0.0911 Val: 0.103 Tst: 0.117058823529 std Tst: 0.000674977038355

[Nepoch = 100 Nh: 8] Tr: 0.0916 Val: 0.1034 Tst: 0.117154411765 std Tst: 0.000722931198312

**Gráficas de Validación y Test:**



Podría ser que con **más épocas**, el error de validación y test fuera menor !

Requiere **mismo entrenamiento que el DSNBoost.**

**Peor performance** que el DSNBoost.

## Ima

**Error CV:**

[Nepoch = 200 Nh: 12] Tr: 0.00573034629925 Val: 0.0340838906356 Tst: 0.0339603960396 std Tst: 0.00483630682844

[Nepoch = 200 Nh: 13] Tr: 0.00592276504509 Val: 0.0353912904947 Tst: 0.0334900990099 std Tst: 0.00510111222641

[Nepoch = 200 Nh: 34] Tr: 0.00674959586188 Val: 0.0361593514697 Tst: 0.0328217821782 std Tst: 0.00541643886872

[Nepoch = 200 Nh: 20] Tr: 0.00630710315148 Val: 0.0363131907959 Tst: 0.032004950495 std Tst: 0.00536569494903

[Nepoch = 200 Nh: 24] Tr: 0.00630725129397 Val: 0.0365433751296 Tst: 0.0319801980198 std Tst: 0.00468181798714

[Nepoch = 200 Nh: 28] Tr: 0.00696107901002 Val: 0.0365460322012 Tst: 0.032995049505 std Tst: 0.00430088058045

[Nepoch = 200 Nh: 16] Tr: 0.00613400775504 Val: 0.036618226032 Tst: 0.0331930693069 std Tst: 0.00478427633698

[Nepoch = 200 Nh: 22] Tr: 0.0066148510215 Val: 0.0368549034066 Tst: 0.0321534653465 std Tst: 0.00401562664197

[Nepoch = 200 Nh: 26] Tr: 0.00603787234669 Val: 0.0370031406928 Tst: 0.0324504950495 std Tst: 0.00396959019411

[Nepoch = 200 Nh: 41] Tr: 0.00709550919874 Val: 0.037006106144 Tst: 0.0321782178218 std Tst: 0.00453179900647

**Error OMN:**

[Nepoch = 200 Nh: 30] Tr: 0.0066918296963 Val: 0.0383146780733 Tst: 0.0313613861386 std Tst: 0.00338222444725

[Nepoch = 200 Nh: 35] Tr: 0.00657627842951 Val: 0.0393910299083 Tst: 0.0319306930693 std Tst: 0.00396503441251

[Nepoch = 200 Nh: 24] Tr: 0.00630725129397 Val: 0.0365433751296 Tst: 0.0319801980198 std Tst: 0.00468181798714

[Nepoch = 200 Nh: 20] Tr: 0.00630710315148 Val: 0.0363131907959 Tst: 0.032004950495 std Tst: 0.00536569494903

[Nepoch = 200 Nh: 27] Tr: 0.0064611159923 Val: 0.0381605417467 Tst: 0.0320544554455 std Tst: 0.00513844931918

[Nepoch = 200 Nh: 22] Tr: 0.0066148510215 Val: 0.0368549034066 Tst: 0.0321534653465 std Tst: 0.00401562664197

[Nepoch = 200 Nh: 41] Tr: 0.00709550919874 Val: 0.037006106144 Tst: 0.0321782178218 std Tst: 0.00453179900647

[Nepoch = 200 Nh: 26] Tr: 0.00603787234669 Val: 0.0370031406928 Tst: 0.0324504950495 std Tst: 0.00396959019411

[Nepoch = 200 Nh: 29] Tr: 0.00628796478462 Val: 0.0390063963167 Tst: 0.032599009901 std Tst: 0.00437433037154

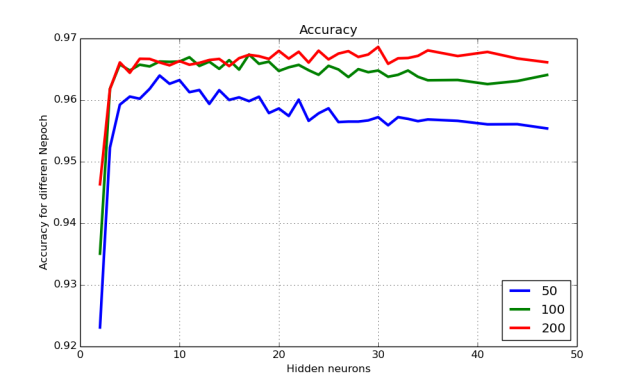
[Nepoch = 100 Nh: 17] Tr: 0.0111344160892 Val: 0.038472083679 Tst: 0.032599009901 std Tst: 0.0049869327335

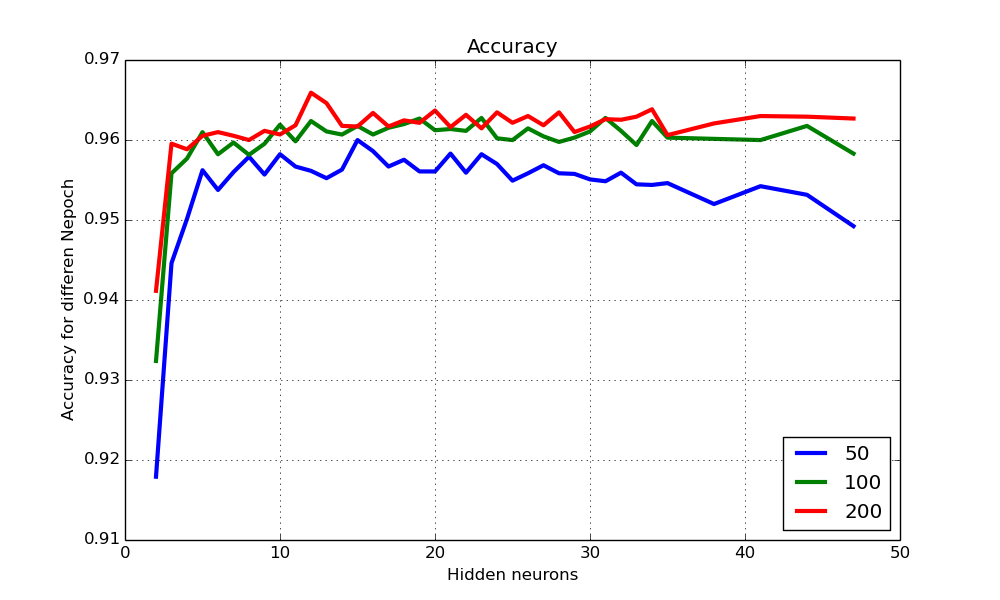
[Nepoch = 200 Nh: 17] Tr: 0.00563415550637 Val: 0.0383131976235 Tst: 0.0326485148515 std Tst: 0.00511837857484

[Nepoch = 200 Nh: 34] Tr: 0.00674959586188 Val: 0.0361593514697 Tst: 0.0328217821782 std Tst: 0.00541643886872

[Nepoch = 200 Nh: 38] Tr: 0.00713395258388 Val: 0.0379285913424 Tst: 0.0328465346535 std Tst: 0.00434340704814

**Gráficas de Validación y Test:**





Podría ser que con **más épocas**, el error de validación y test fuera menor !

Requiere **mismo entrenamiento que el DSNBoost.**

**Misma performance** que el DSNBoost.

## Wav

**Error CV:**

[Nepoch = 50 Nh: 32] Tr: 0.00662522400121 Val: 0.108219292077 Tst: 0.11352173913 std Tst: 0.00248825217086

[Nepoch = 50 Nh: 30] Tr: 0.00668831055479 Val: 0.108700539147 Tst: 0.1145 std Tst: 0.00232679794091

[Nepoch = 50 Nh: 35] Tr: 0.00725081299622 Val: 0.108972691045 Tst: 0.113809782609 std Tst: 0.00295523769271

[Nepoch = 50 Nh: 29] Tr: 0.0065002227805 Val: 0.109225543054 Tst: 0.114190217391 std Tst: 0.00167121995387

[Nepoch = 50 Nh: 26] Tr: 0.00631369996289 Val: 0.110203547429 Tst: 0.113445652174 std Tst: 0.00247938094444

[Nepoch = 50 Nh: 20] Tr: 0.00618830811336 Val: 0.11023488045 Tst: 0.11277173913 std Tst: 0.00178522475061

[Nepoch = 50 Nh: 33] Tr: 0.00706311829217 Val: 0.110688115331 Tst: 0.113690217391 std Tst: 0.00220010713565

[Nepoch = 50 Nh: 44] Tr: 0.00918812927861 Val: 0.110926394749 Tst: 0.115206521739 std Tst: 0.00248385619308

[Nepoch = 50 Nh: 34] Tr: 0.00743792480884 Val: 0.111222769183 Tst: 0.114766304348 std Tst: 0.00244185091059

[Nepoch = 50 Nh: 25] Tr: 0.00687620240432 Val: 0.111469526489 Tst: 0.112880434783 std Tst: 0.00195516257138

**Error OMN:**

[Nepoch = 50 Nh: 11] Tr: 0.00549982848954 Val: 0.118038599781 Tst: 0.112744565217 std Tst: 0.00200877534897

[Nepoch = 50 Nh: 20] Tr: 0.00618830811336 Val: 0.11023488045 Tst: 0.11277173913 std Tst: 0.00178522475061

[Nepoch = 50 Nh: 25] Tr: 0.00687620240432 Val: 0.111469526489 Tst: 0.112880434783 std Tst: 0.00195516257138

[Nepoch = 50 Nh: 18] Tr: 0.00650159181242 Val: 0.114501015784 Tst: 0.112923913043 std Tst: 0.0020531397414

[Nepoch = 50 Nh: 9] Tr: 0.00493635008643 Val: 0.119001328333 Tst: 0.113260869565 std Tst: 0.00188266392127

[Nepoch = 50 Nh: 10] Tr: 0.0059381121886 Val: 0.117244921081 Tst: 0.113266304348 std Tst: 0.00178967812157

[Nepoch = 50 Nh: 15] Tr: 0.00568791626383 Val: 0.116519846851 Tst: 0.113277173913 std Tst: 0.00265421673821

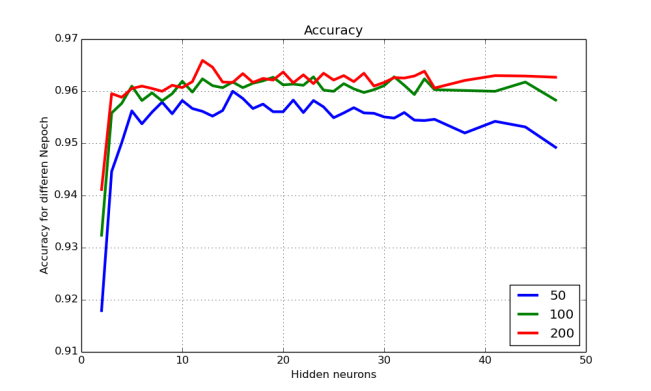
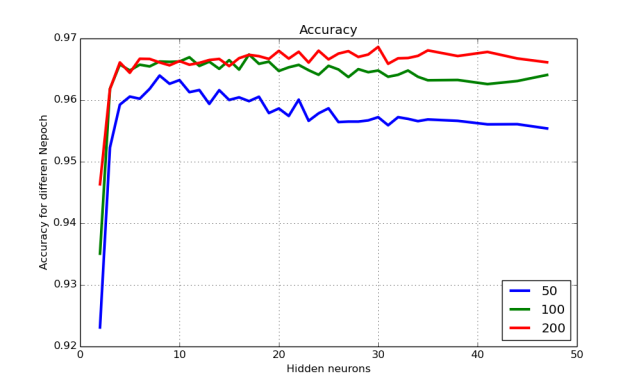
[Nepoch = 50 Nh: 23] Tr: 0.00593791626383 Val: 0.115969604626 Tst: 0.113331521739 std Tst: 0.00200152794895

[Nepoch = 50 Nh: 22] Tr: 0.00650119996289 Val: 0.114485114862 Tst: 0.113336956522 std Tst: 0.00208998308619

[Nepoch = 50 Nh: 19] Tr: 0.00625119996289 Val: 0.114994530395 Tst: 0.113364130435 std Tst: 0.00180119387786

[Nepoch = 50 Nh: 13] Tr: 0.00556272033907 Val: 0.116751015784 Tst: 0.113369565217 std Tst: 0.00229087530015

**Gráficas de Validación y Test:**



Podría ser que con **más épocas**, el error de validación y test fuera menor !

Requiere **mismo entrenamiento que el DSNBoost.**

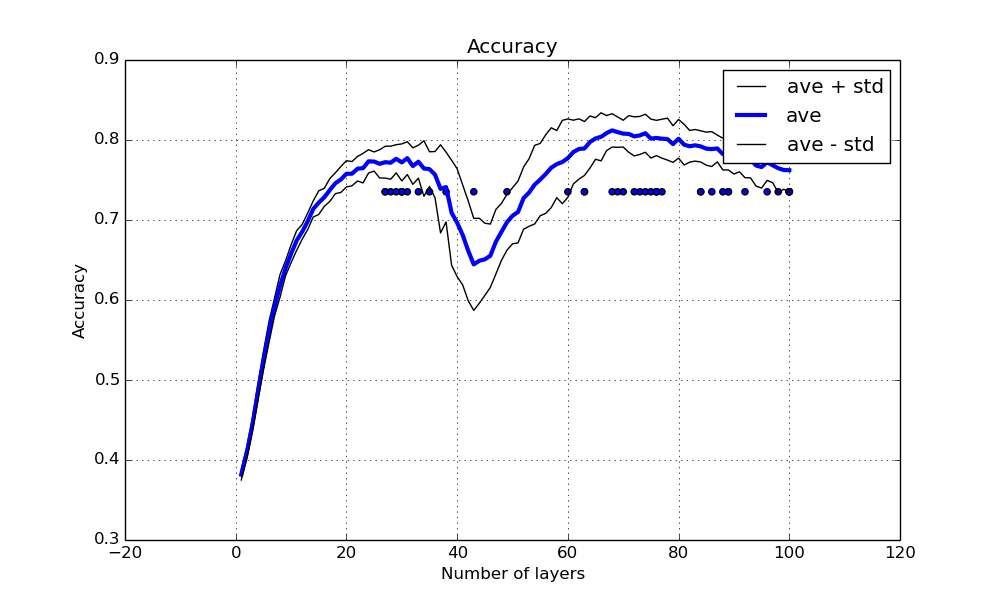
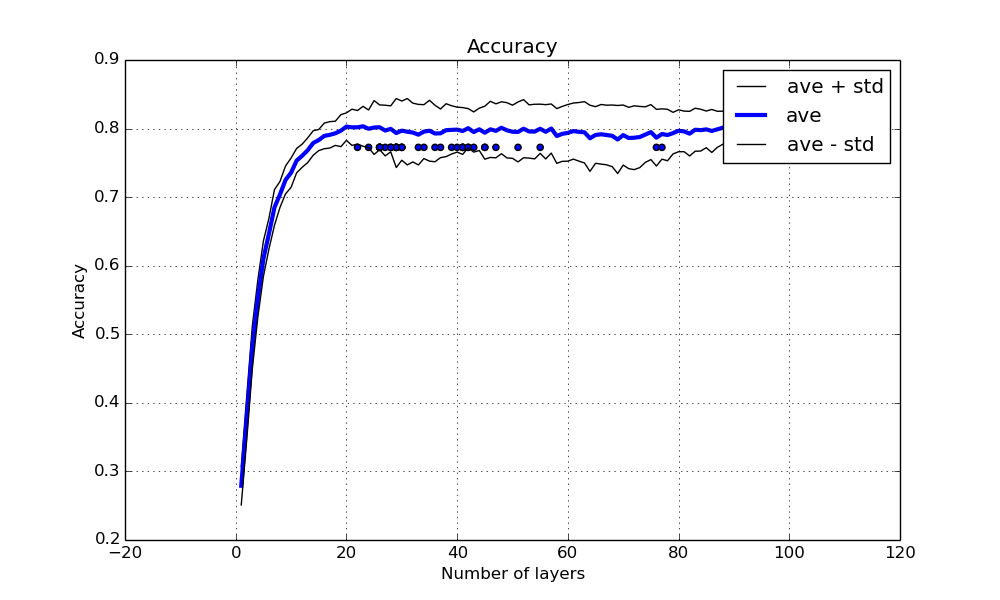
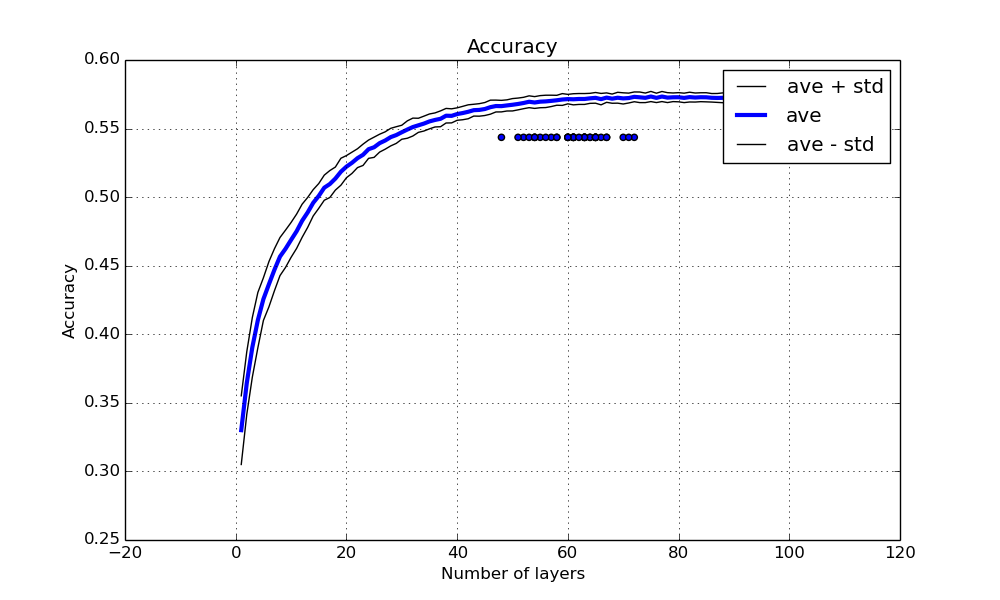
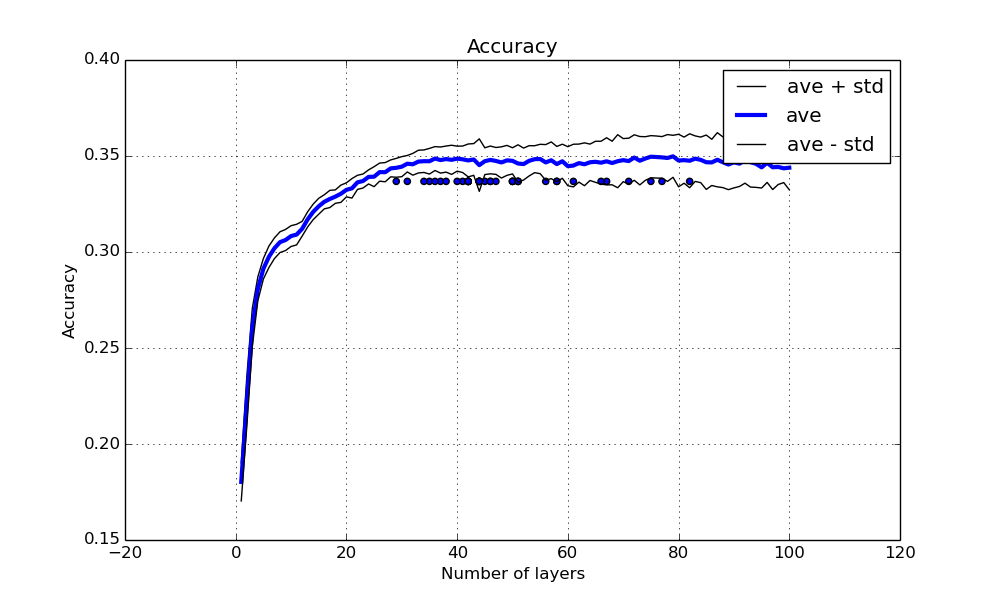
**Peor performance** que el DSNBoost.

# Evolución DSN L = 10

En esta sección evaluaremos a grandes rasgos cómo evoluciona el algoritmo DSN con el número de neuronas, igual que en el DSNBoost.

## Evolución de gamma

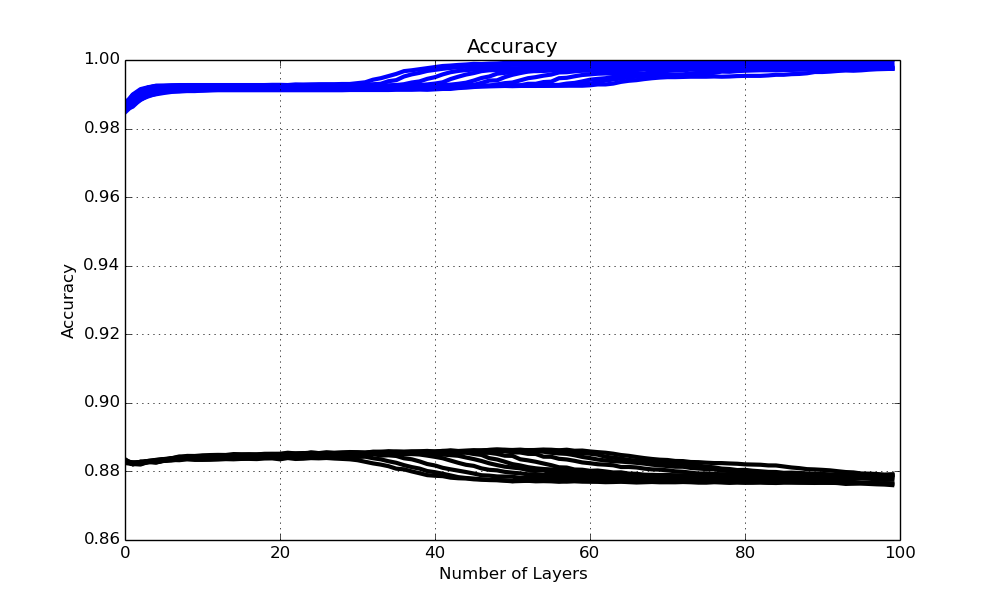
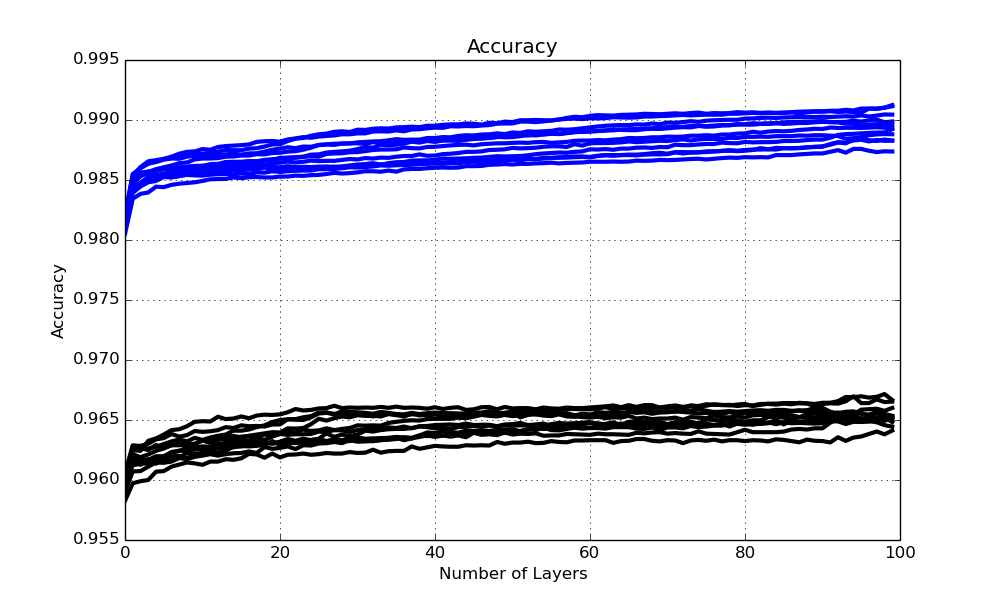
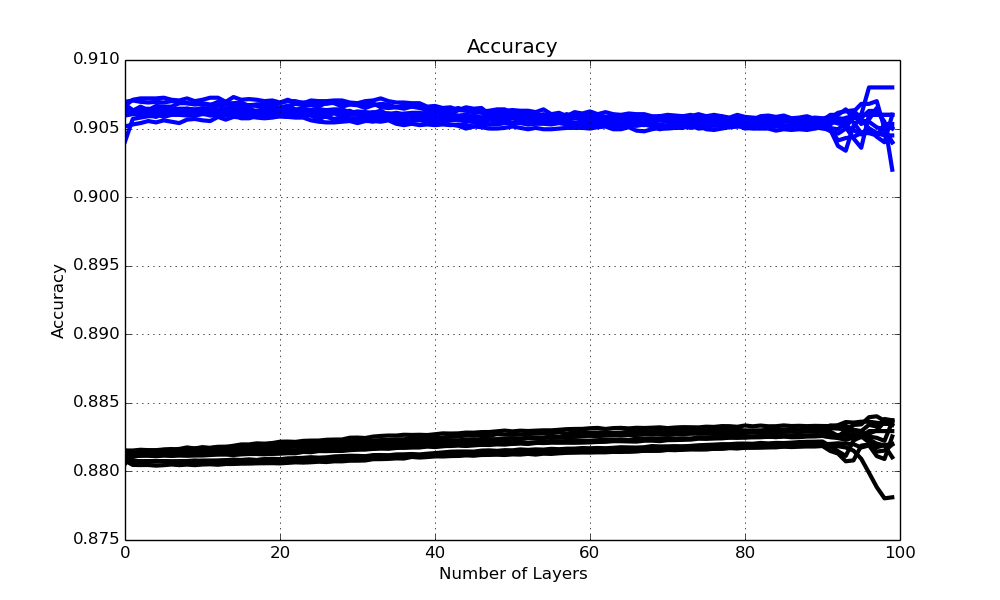
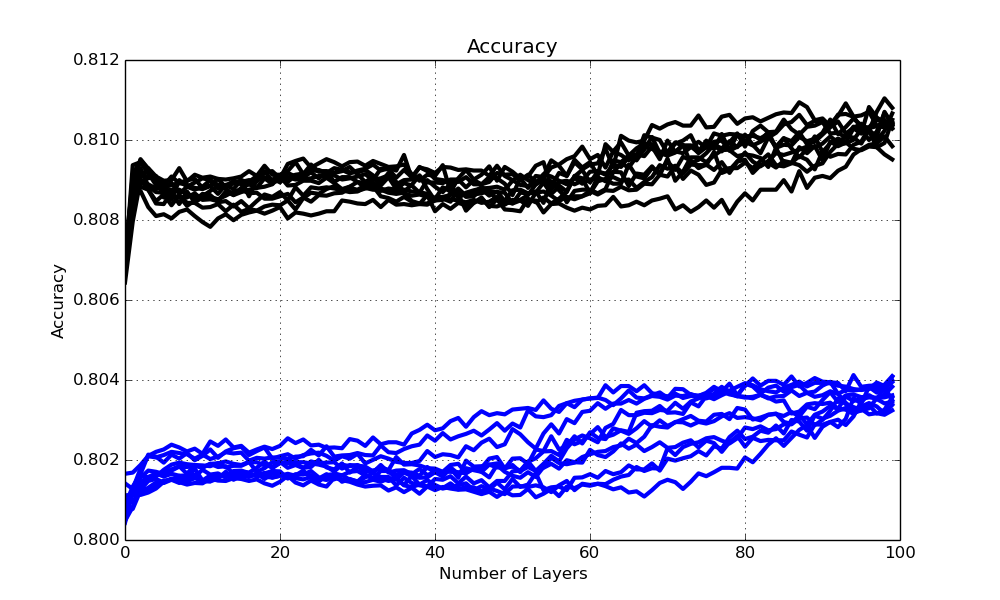
Las siguientes 4 gráficas muestran la evolución de gamma para las 4 bases de datos en función del número de capas para Nh = 27 y Nepoch =100



En este caso, las gammas prácticamente siempre crecen con el número de capas. En DSNBoost, algunas bases de datos tenían gammas que crecen y otras que decrecen. El valor de convergencia de la gamma es mucho menor para DSN que para DSNBoost, alrededor de casí la mitad, por lo que DSNBoost consigue un mejor valor del “margin” que DSN. Como vemos además, DSN requiere un mayor número de capas para converger.

## Evolución de la accuracy

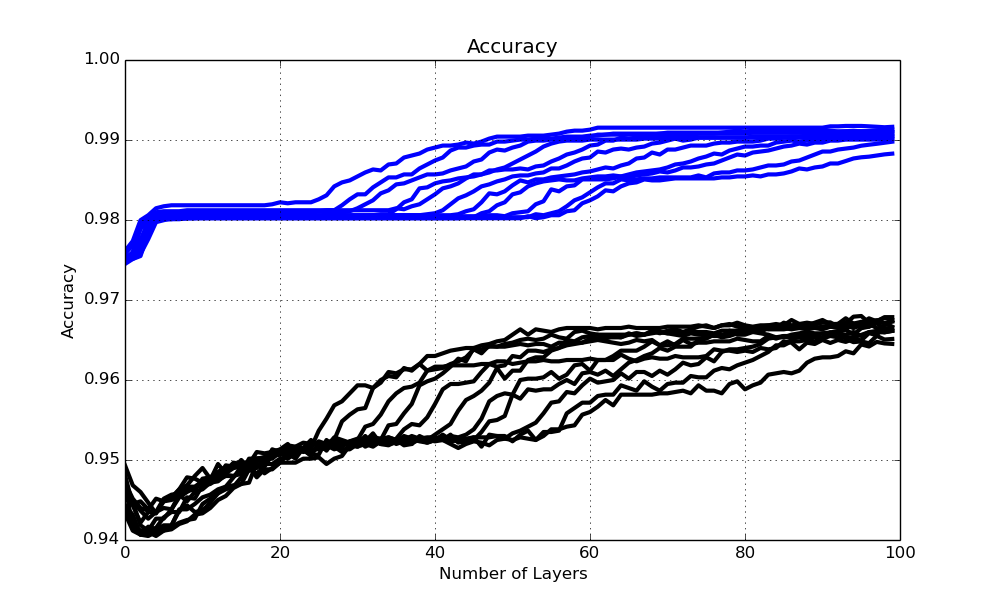
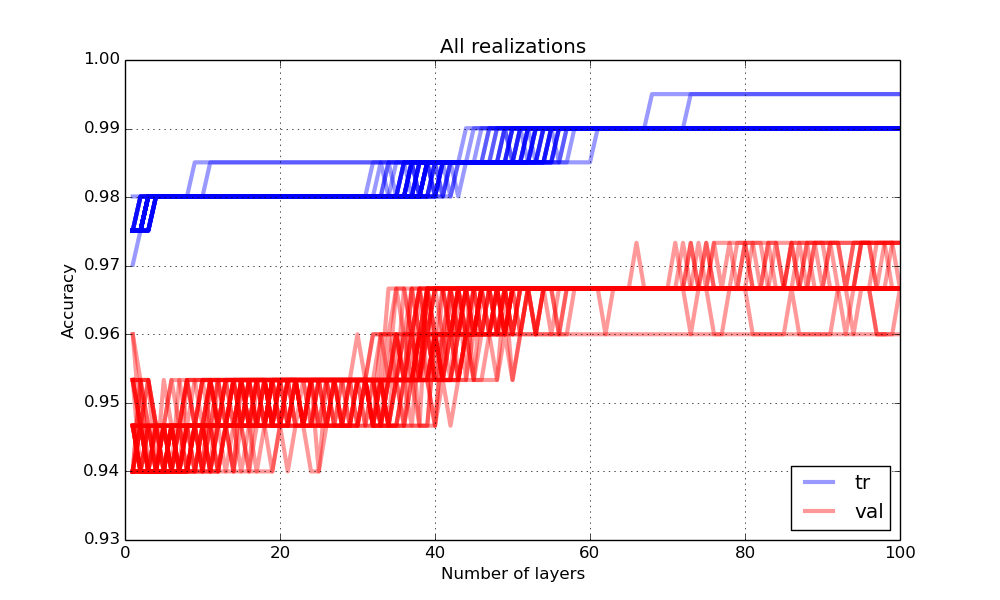
Las siguientes 4 gráficas muestran la accuracy media para las 4 bases de datos en función del número de capas para diferentes números de neuronas y Nepoch = 100, con esto queremos ver el comportamiento del algoritmo en general y en función del número de neuronas



Vemos que, a diferencia del DSNBoost, el algoritmo DSN progresa más allá de las primeras 20-30 capas, en todas ellas se ve una progresión, ya sea overfitting o no.

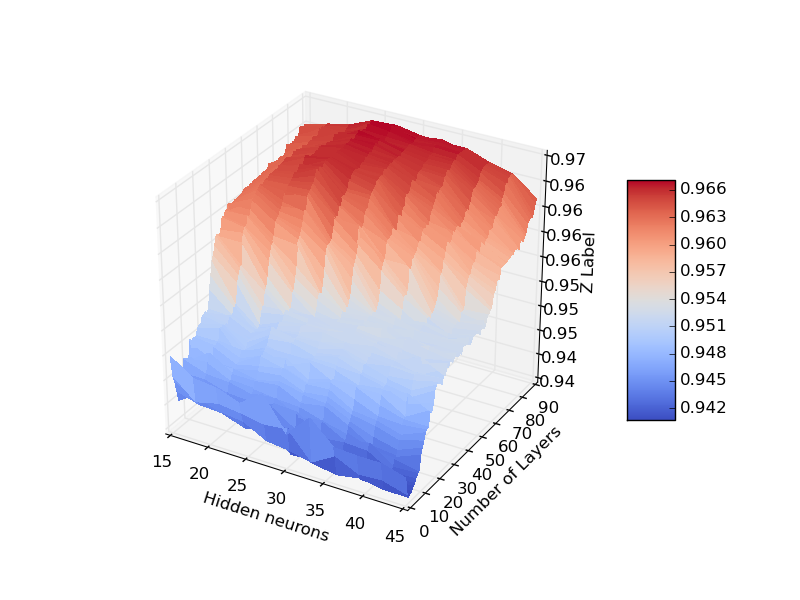
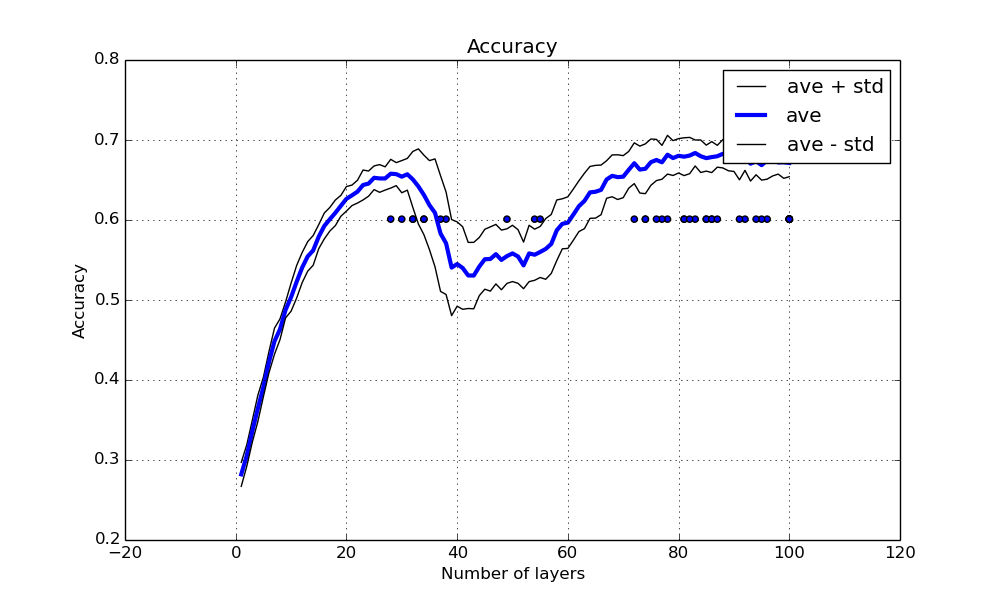
Cabe destacar también que en diversas bases de datos (más de las mostradas aquí), hay un **punto de inflexión** de repente, como se ve en la primera y cuarta gráfica donde el algoritmo sale de un estado estacionario y evoluciona. No tengo muy claro el por qué de este fenómeno.

La base de datos **ionosfera** manifesta este efecto de forma signicativa:



Como podemos apreciar, si se parase el entrenamiento en N = 100, el error de test sería aproximadamente 3% para algunas realizaciones, cuando el mejor error OMN de DSN = L, aplicando los criterios de parada normales es 3.8%.

Tener en cuenta que esta base de datos también hace extraños con las gammas:



**Deberíamos estudiar la causa de este comportamiento deseable, controlarlo e intentar introducirlo en DSNBoost.**

# Conclusiones finales DSNboost

Vamos a intentar sacar conclusiones del algoritmo y posibles variaciones del mismo para mejorar en base a estas 4 bases de datos mostradas y a otras cuantas con las que ha sido probado pero de las que simplemente se dirán de palabra sus resultados ya que son bases de datos pequeñas en las que para obtener el estado del arte probablemente haya que utilizar un algoritmo más fino en vez del que utilizamos ahora un poco a groso modo:

* **Ion**: CV [Nepoch = 200 Nh: 2] Tr: 0.00497670807453 Val: 0.179670731707 Tst: 0.0465 std Tst: 0.0118778505351

OMN [Nepoch = 200 Nh: 29] Tr: 0.00348447204969 Val: 0.19656097561 Tst: 0.0365 std Tst: 0.00446902922593

* **Breast**: Esta base de datos es curiosa, si el número de épocas es pequeño, unas 20 épocas, entonces el error de tst es 1.792% y varianza 0 !!

Si se sube el número épocas entonces el error crece y hay varianza.

En términos generales es un buen algortimo, gana o iguala a muchos algoritmos utilizados sobre dichas bases de datos.

* Requiere un número de neuronas ocultas y capas que rondan las pocas decenas en el peor de los casos.
* Converge a un buen resultado pero quizás demasiado pronto.
* En muchos casos si se aumenta mucho el número de neuronas se acaba haciendo underfitting dado que el entrenamiento es el mismo siempre.
* Funciona bien para las 6 bases de datos utilizadas por lo que probablemente funcione bien para un gran rango de problemas.

Dado que el error al que converge el algoritmo con el número de capas es bueno, es un algoritmo estable cuyo criterio de parada no tiene por qué ser muy fino.

Deberíamos centrar esfuerzos en conseguir que el algoritmo convergiese en un número de capas mayor, manteniendo la buena progresión del mismo.

### Cambiar forma de entrenamiento de las capas

Para algunas bases de datos, 100 épocas son muchas y para otras son pocas. Las prestaciones del algoritmo varían significativamente con el número de épocas. O cambiamos la forma de entrenar o tendremos que validar este parámetro también. Para bases de datos pequeñas es decisivo.

Utilizar entrenamiento en modo **mini-batch mode**.

**Nota**: En el paper de DSN utilizan mini-Batch porque dicen que utilizando Batchpropagation normal no funciona bien. Además reduce considerablemente el tiempo. Ya que tenemos los resultados para BackPropagation normal podríamos probar con mini-Batch mode ahora.

### Cambiar inicialización de las capas

Podríamos aumentar los límites de la inicialización al utilizado por las técnicas de deep learning:

O cualquier otro criterio que creamos conveniente.

Utilizar un **algoritmo generativo** no supervisado para pre-entrenar la capa oculta:

* Autoencoders
* Restricted Boltzman Machines (RBM)

**Nota**: Los autores del paper original de DSN pre-entrenan sus redes con una RBM y después calculan la salida como solución de mínimos cuadrados de la red pre-entrenada.

Además, como el resultado final del entrenamiento depende en gran medida de la buena inicialización de las primeras capas alomejor podría mejorar la media y reducir la varianza.

### Cambiar estructura

Como hemos visto en todos los ejemplos, la NN converge rápido, normalmente menos de 20 capas. Probablemente esto es debido a que las aportaciones de las nuevas capas cada vez es menor (dado que la salida va aumentando), deberíamos dar más valor a las últimas capas ya que probablemente su salida va mejorando (dado que son entrenadas con una inyección mejor cada vez).

Posibles cambios en esta línea son:

* Salida final como combinación convexa de la salida final anterior y la salida de la última capa ponderada por :

Tendríamos que validar

Notar que para los casos específicos:

* + : .

La salida total es la salida de la última capa. Igual que DSN sin la agregación de boosting que hacemos. Igual que solo propragar la salida anterior (por consecuencia).

* + :

Este caso es igual al que ya estamos utilizando (la del RAB) solo que las aportaciones de capas anteriores se reducen geométricamente por la mitad:

* Lo mismo que la anterior pero multiplicado por 2:

De esta manera, para el caso entonces:

Esta es la ecuación de agregación del RAB y la utilizada en este report.

* Hacer que la importancia de las capas anteriores vaya decrementando geométricamente:

Tenemos que:

Hay que validar entre 0 y 1.

* Incrementar la importancia de las últimas capas con:

Validar entre 0 y 1.

* Saturate the output at every layer:

This will give more importance to the last layers but probably wil saturate the values of making them either 1 or -1. Se podría intentar para evitarlo:

* Usar el margen poderado r como constante de la combinación convexa:

De esta forma no hay que validar ningún parámetro más.

* A todas las técnicas utilizadas se podría añadir también normalizar siempre la salida total en cada capa:

De esta forma siempre estará entre 0 y 1 y por ende tendrá más importancia, pero alomejor demasiada, habría que cuestionarse la agregación

Otras estructuras probadas, con la misma función de **agregación** son:

* Entrenar con activación de salida lineal, normalizando la salida después por medio de dividir los pesos y bias de la salida por el valor máximo de
* Entrenar con activación de salida tanh pero después de entrenar cada capa, tratarla como si hubiera sido salida lineal (Normalizamos y agregamos las )
* En vez de inyectar la normalizada, se inyecta dividida por el sumatorio de gammas:

Los resultados son muy muy similares en cuanto al error omnisicente. CV no realizado.

Podríamos también inyectar en vez de sólo la última salida, , **inyectar una ventana de salidas** ó una ventana de salidas de las capas anteriores

Habrá algunos cambios que sean una estupidez y otros que puedan funcionar, pero hay cambios para rato. Lo bueno es que el código está optimizado, se tardaría medio día en calcular el error omnisciente para cada estructura haciendo un barrido de 0 a 90 Nh, 100 capas y 3 valores diferentes de número de épocas. Si hay que validar pues el tiempo requerido se multiplica. Todo esto si el cluster está medio libre claro (700 nodos).

Falta por mejorar las gráficas y hacer un sistema automático de resultados pero en menos de 2 semanas estaría acabado, a partir de entonces sería ir cambiando la estructura, enviarlo al cluster y listo. Si quiere representar lo que sea con los datos que tenemos dígamelo.

### Aplicar función de énfasis sólo con alpha.

He lanzado 20 realizaciones del error omnisciente para 5 bases de datos. Ha tardado lo suyo y la mayoría de los resultados son paupérrimos, los mejores resultados son siempre para valores altos de beta, es decir, teniendo solo en cuenta el error MSE y el bias alpha.

Dada esta información, mi propuesta es primeramente establecer y solo validar

Luego ya, podemos validar .

# Resultados Finales

Como se ha explicado anteriormente, el algoritmo tiene suficientes grados de libertad ajustables como para obtener mejores resultados de los ya obtenidos. Pero lo podemos utilizar como baseline para futuras modificaciones por leves que sean.

La idea hacer el **estudio evolutivo del error omnisciente** para las nuevas estructuras y si este mejora respecto con el que ya tenemos, obtener el conjunto de parámetros óptimo por cross-validation.

Dejo en su mano **decidir qué modificaciones realizar primero**, en el tiempo que pase hasta que nos volvamos a ver iré probando primero aquellos que no requieran nuevos parámetros a validar.

A modo de resumen escribo aquí los resultados del error OMN y CV de la estructura DSNBoost para las bases de datos utilizadas. A comparar con los de otras arquitecturas:

OMN Abalole

CV Abalole

OMN Kwok

CV Wav

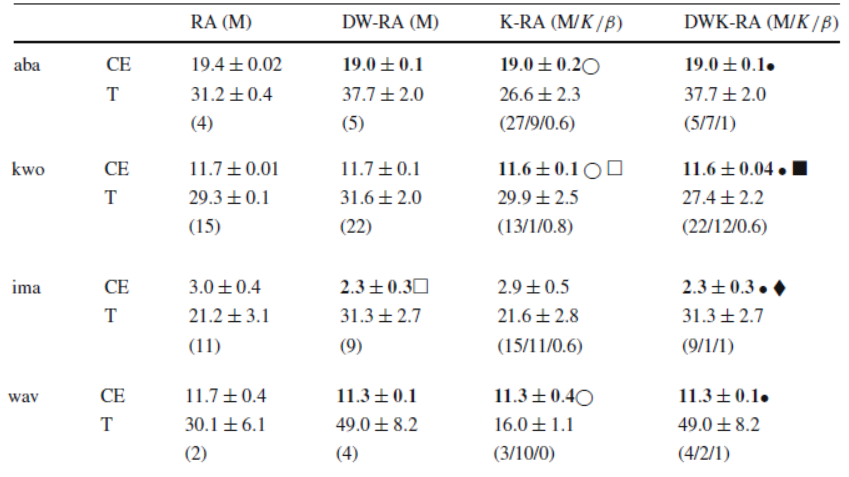
OMN Ima

CV Ima

OMN Wav

CV Wav

Resultados de otras arquitecturas obtenidas del último paper publicado de Anas.



**Nota**: Existen un montón de bases de datos pequeñas anexas como Breast que con poco número de épocas, DSNBoost lo hace muy bien. Los resultados para ion también son muy buenos. Si quiere probarlo en alguna otra base de datos dígamelo.