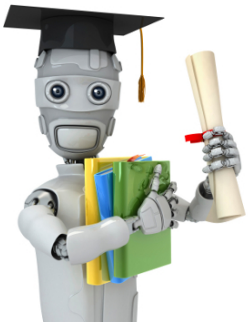
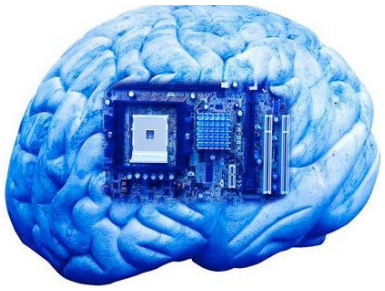
Práctica 5: Aprendizaje automático

(Machine Learning)





Grado en Ing. Informática. Inteligencia Artificial 2020/2021

D.L: TE-126-2013

Contenido

Esta práctica se corresponde con el Tema 5 de la asignatura, que a su vez está basado en el Capítulo 18 de la bibliografía básica ("*Artificial Intelligence: A Modern Approach, Chapter 18: Learning from Examples*").

Objetivos de la práctica

La quinta práctica de la asignatura se centrará en la utilización de técnicas de aprendizaje automático en una tarea concreta de clasificación. Se empleará el aprendizaje supervisado en el que se disponen de una serie de ejemplos etiquetados con un valor de pertenencia a clase. Estos ejemplos se utilizarán para entrenar (*train*) un algoritmo de aprendizaje que halle patrones en los datos, de forma que sea capaz de utilizarse para etiquetar nuevos ejemplos de los que se desconozca su valor de clase.

Redes neuronales artificiales

Las redes neuronales artificiales (*Artificial Neural Networks, ANNs*) son un algoritmo de aprendizaje inspirado en la estructura neuronal del cerebro. Las redes neuronales están formadas por neuronas interconectadas organizadas en capas, donde la capa de entrada está formada por los atributos de cada uno de los ejemplos disponibles, y la salida correspondería con el valor de clase del mismo. Las capas ocultas permiten aumentar el poder de expresión de la red neuronal.

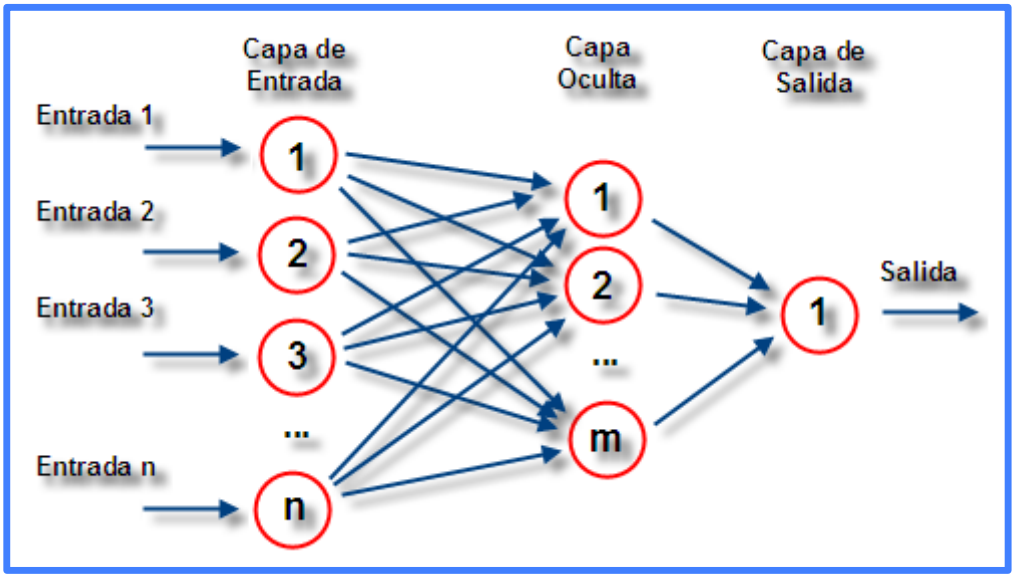


Figura 1: Estructura general de una red neuronal

Las neuronas son las unidades básicas de procesamiento en redes neuronales. Cada neurona posee una serie de entradas (numéricas), las cuales se ponderan con un peso para cada entrada (w_i) y se suman sus valores. Este valor sumado posteriormente pasa por una función de transferencia que normalmente limita el valor de la salida a un intervalo concreto, normalmente [-1, 1] o [0, 1].

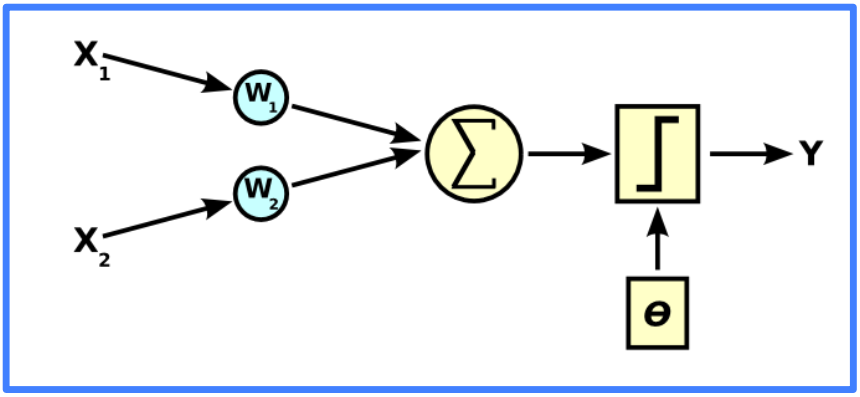


Figura 2: Estructura de una neurona con dos entradas

La capacidad de expresión de una red neuronal depende del número de capas de la red, así como del número de neuronas de cada capa.

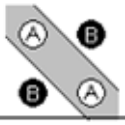
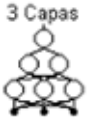
Estructura	Regiones de Desición	Problema de la XOR	Clases con Regiones Mezcladas	Formas de Regiones más Generales
1 Capa 	Medio Plano Limitado por un Hiperplano			
2 Capas 	Regiones Cerradas o Convexas			
3 Capas 	Complejidad Arbitraria Limitada por el Número de Neuronas			

Figura 3: Capacidad de discriminación de una red neuronal

Aprendizaje supervisado usando redes neuronales

En la variante supervisada de aprendizaje, se dispone de una serie de ejemplos con un valor de clase asignado que se emplean para hallar un patrón que permita clasificar nuevos ejemplos. Las redes neuronales precisan de una fase de entrenamiento en la que se ajustan los pesos de cada neurona así como los umbrales para las funciones de activación mediante una función de retropropagación (*backpropagation*). Básicamente, se modifican aquéllos pesos que consiguen reducir en mayor proporción el error entre la salida de la red y el valor esperado, usando un tamaño de paso (a veces llamado η) que es necesario ajustar. En general, el algoritmo de retropropagación es el siguiente:

1. Calcular la salida de la red a partir de uno de los conjuntos de valores de prueba
2. Comparar con la salida correcta y calcular el error cuadrático medio
3. Calcular las derivadas parciales del error con respecto a los pesos que unen la capa oculta con la de salida
4. Calcular las derivadas parciales del error con respecto a los pesos que unen la capa de entrada con la oculta
5. Ajustar los pesos de cada neurona en función del tamaño de paso η para reducir el error
6. Repetir el proceso varias veces por cada par de entradas-salidas de prueba

Problema a resolver

Se va a desarrollar una red neuronal para clasificar caracteres codificados mediante matrices de 7x5 píxeles, donde un valor de 1 representa el color negro y un valor 0 representa el color blanco. Se intentará obtener una red neuronal que sea capaz de reconocer las 26 letras mayúsculas del alfabeto inglés, de forma que la red deberá tener 35 entradas (una para cada píxel) y 26 salidas (una por cada posible valor de clase).

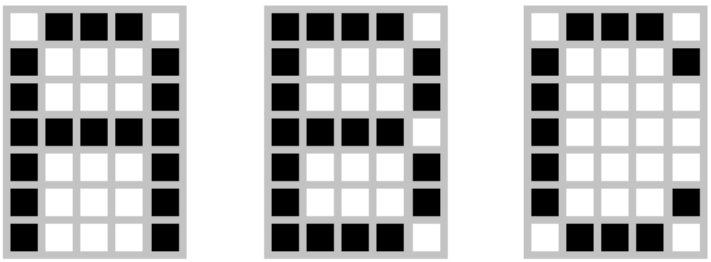


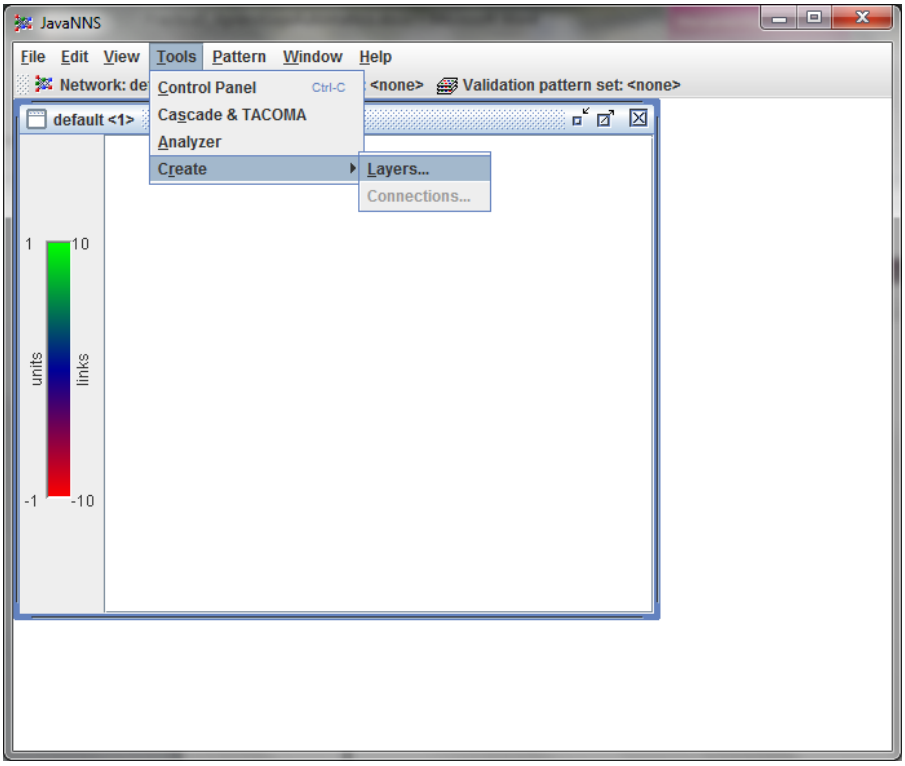
Figura 4: Ejemplos de letras codificadas como píxeles

Java Neural Network Simulator (JavaNNS)

JavaNNS[1] es una implementación en Java del paradigma de redes neuronales realizada en la Universidad de Tübingen (Alemania). Proporciona una interfaz gráfica para la creación, entrenamiento y validación de redes neuronales. Las diferentes acciones que pueden realizarse con este software (y que desarrollaremos durante la práctica) son las siguientes:

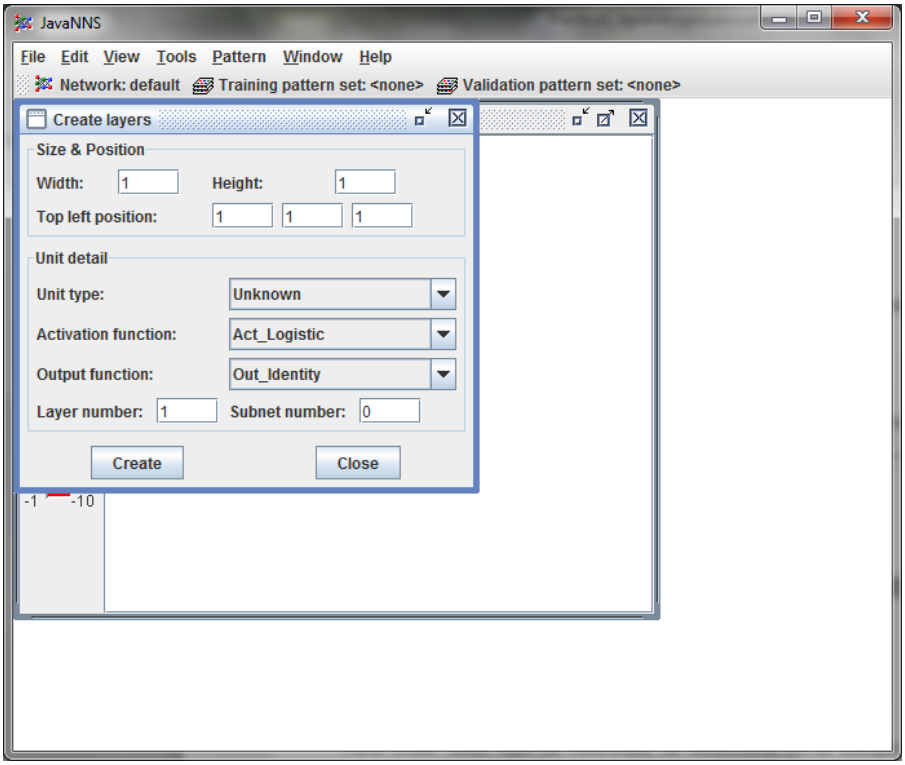
Creación de una red

Las neuronas se crean por capas, normalmente una capa de entrada, varias (o ninguna) capas ocultas, y una de salida. Para crearlas, se selecciona *Tools* → *Create* → *Layers...*

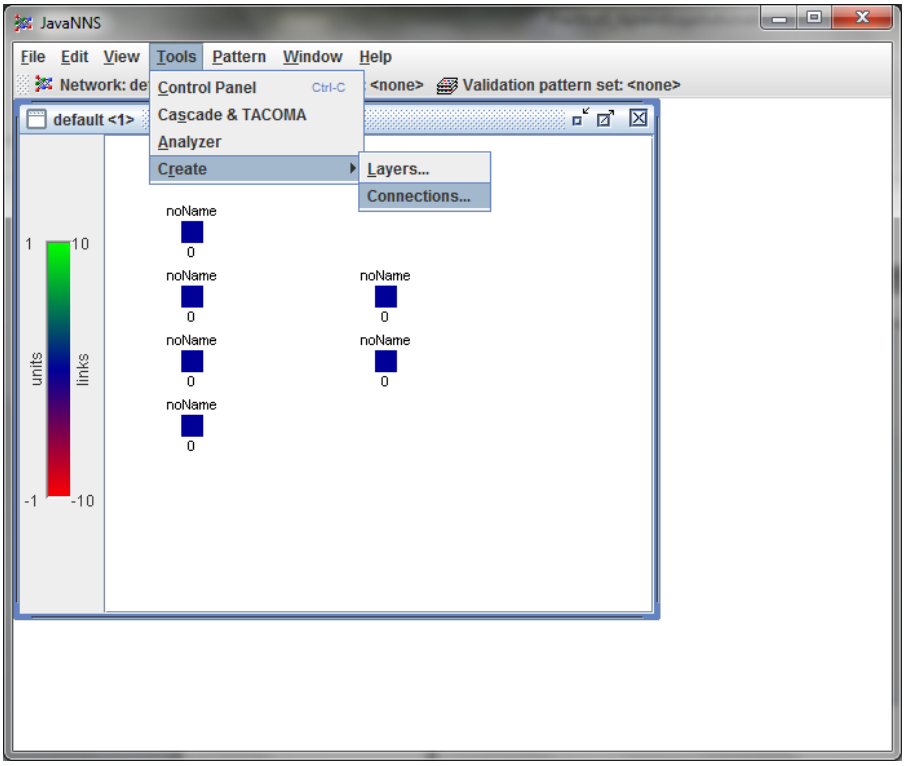


Para crear cada tipo de neuronas, se selecciona en el campo *Unit type*. El número de las mismas se define mediante los parámetros *Width* y *Height*, donde la capa tendrá *Width x Height* neuronas en forma de matriz. A la hora de crear las conexiones, es importante que la capa número 1 (parámetro *Layer Number*) esté formada por las neuronas de entrada, las capas ocultas sean las intermedias, y la última capa se corresponda por las neuronas de salida.

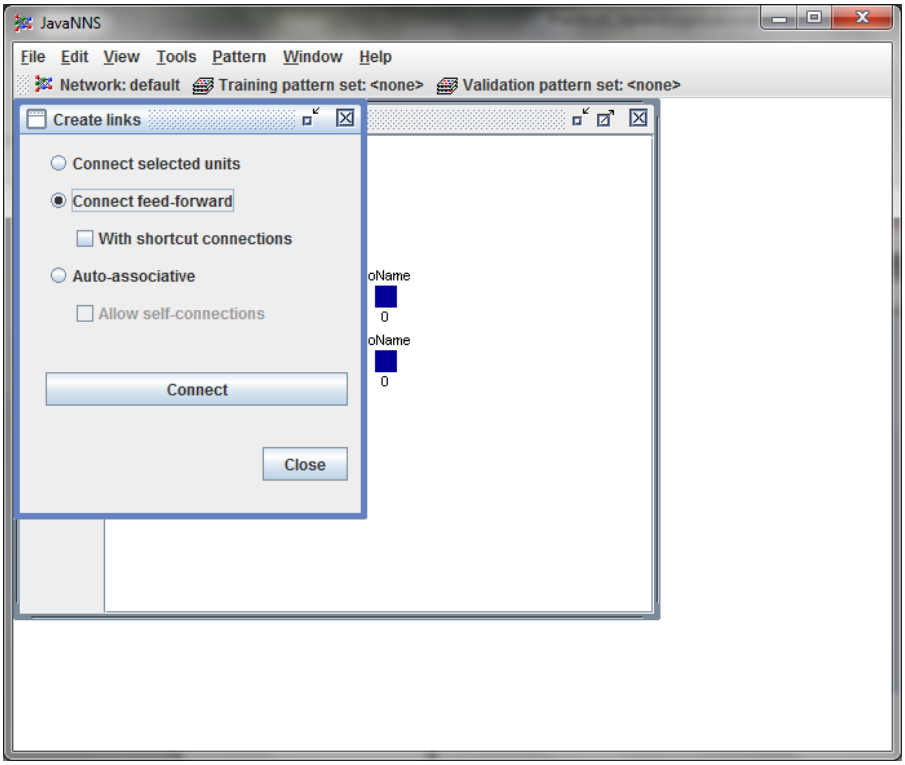
[1] http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/welcome_e.html



Una vez creadas las capas, se unen usando *Tools* → *Create* → *Connections…*



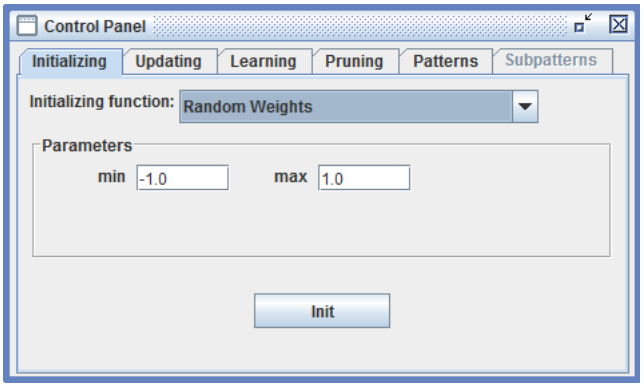
De las opciones disponibles, se selecciona *Connect feed-forward*.



Entrenamiento de una red

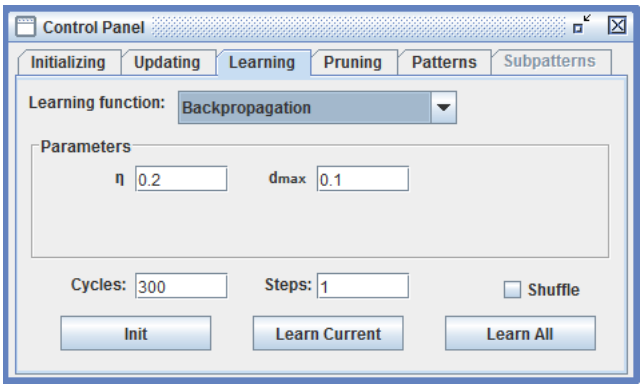
Antes de entrenar la red, debe cargarse un fichero con los patrones de entrenamiento, que contengan la entrada para cada una de las neuronas de entrada y la correspondiente salida esperada, de forma que se puedan ajustar los pesos de la red. Los patrones se cargan mediante *File* → *Open*.

Una vez cargado el fichero de entrenamiento, se abre el panel de control mediante *Tools* → *Control Panel*. El panel de control posee varias pestañas, pero en la práctica nos centraremos en *Initializing*, *Learning* y *Patterns*.

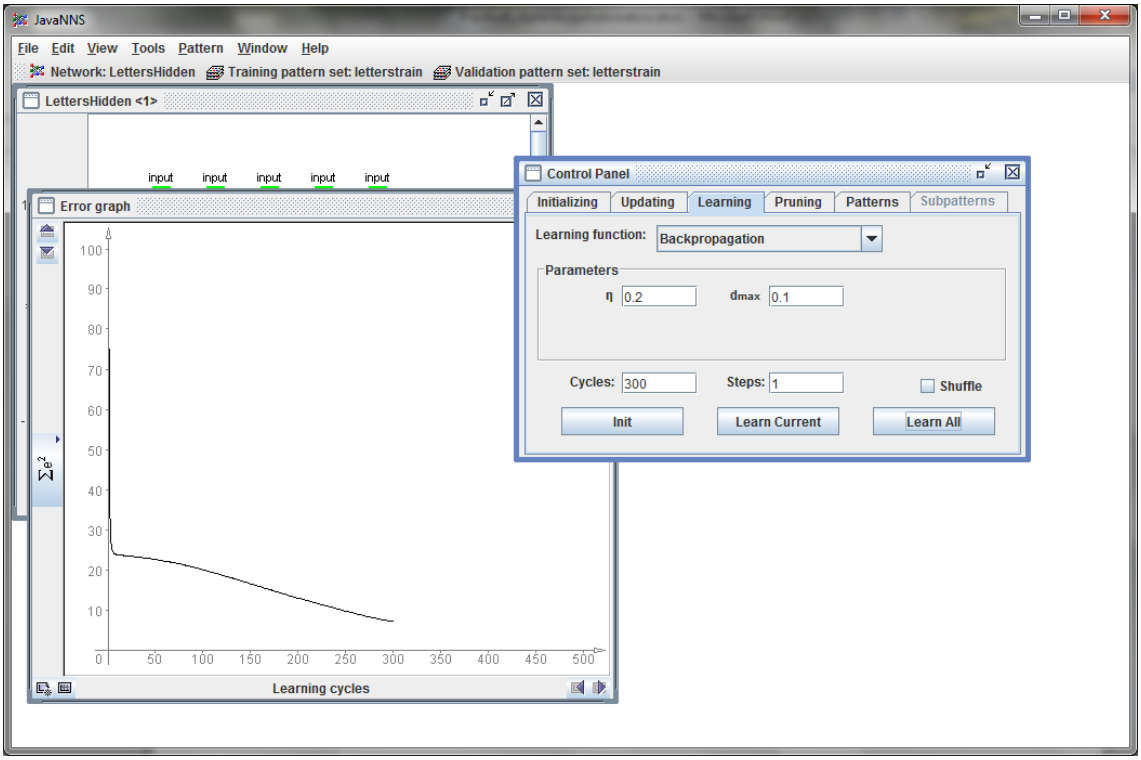


En la pestaña *Initializing*, se puede modificar el intervalo de inicialización de los pesos de la red. En la pestaña *Learning* se pueden seleccionar los diferentes algoritmos de entrenamiento de la red con sus parámetros correspondientes, así como el número de ciclos de

entrenamiento a ejecutar. En la pestaña *Patterns* se indica qué patrones de todos los abiertos se emplean como patrones de entrenamiento y de validación.

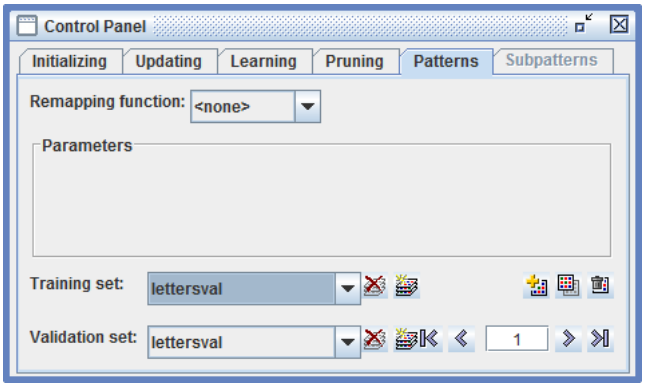


Para comprobar la eficiencia del entrenamiento, se selecciona *View* → *Error Graph*. El gráfico de error muestra la evolución del error cometido por la red conforme pasan los ciclos de entrenamiento.

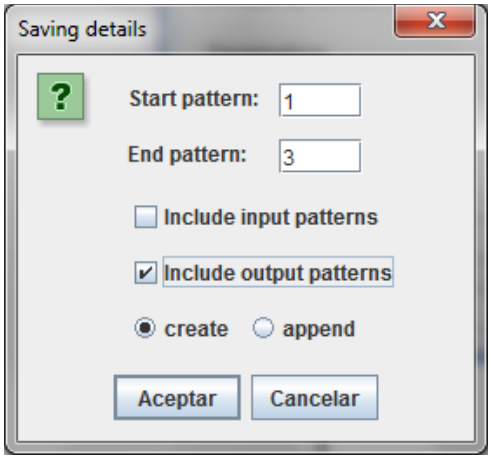


Uso de patrones de validación

Los patrones de validación se cargan como los de entrenamiento (*File* → *Open*). Con la red previamente entrenada, se selecciona el patrón de validación tanto en los desplegables *Training set* y *Validation set*.

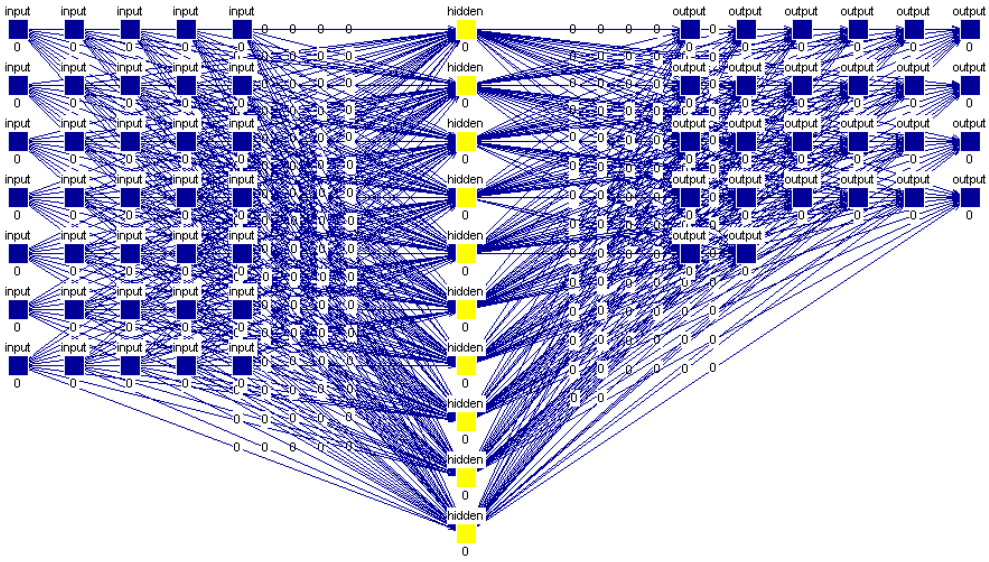


A continuación, se usa la opción *File* → *Save data…* para guardar el resultado de la clasificación. Se selecciona el tipo de archivo *Result files *.res*, y se guarda el fichero con un nombre, indicando en el diálogo *Include output patterns*. El fichero resultante es de texto plano y se puede consultar con un editor de texto cualquiera.



Ejercicios a realizar

1. Construir dos redes neuronales para resolver el problema de la clasificación de letras a partir de sus píxeles. Una de la redes tendrá una capa oculta y otra no tendrá capa oculta. Las características de las redes serán las siguientes:
    - Capa de entrada: 35 neuronas (7 x 5). *Unit type:* Input
    - Capa oculta (si necesaria): 10 neuronas (10 x 1). *Unit type:* Hidden
    - Capa de salida: 26 neuronas (6 x 4 + 2). *Unit type:* Output



2. Entrenar ambas redes usando el fichero de patrones *letterstrain.pat* y empleando el método de retropropagación estándar (*Backpropagation*) hasta que se alcance la convergencia de la red (es decir, el error tiende a cero) o hasta alcanzar 5.000 ciclos. Repetir el proceso 5 veces por cada combinación de parámetros a usar. Utilizar los siguientes parámetros y comentar las diferencias en los resultados (número de ciclos hasta completar el entrenamiento, uniformidad en la evolución del error, si se consigue entrenar la red correctamente o no, etc.):
    - η: usar los valores 0.2, 2.0, 5.0 y 10.0
    - d_{max}: 0.1

3. Usando el valor de η más eficiente del apartado anterior para ambas redes (es decir, que consiga entrenar la red con éxito todas las veces en el menor número de ciclos), cambiar los intervalos de inicialización de los pesos a los intervalos [-0.01, 0.01], [-0.1, 0.1] y [-5.0, 5.0]. Comentar los resultados obtenidos.

4. (Opcional) Comentar la diferencia básica entre el método de retropropagación estándar (*Backpropagation*) y retropropagación con *momentum* o término de inercia (*Backprop-Momentum*). Comprobar las diferencias al entrenar la red con capa oculta con ambos métodos, usando el parámetro η con los valores 0.2 y 2.0, y el resto de parámetros a su valor por defecto. Repetir 5 veces el entrenamiento para cada combinación de parámetros y algoritmo y comentar los resultados obtenidos.

5. Entrenar la red con capa oculta empleando los parámetros más eficientes. Con la red ya entrenada, cargar el fichero *lettersval.pat* y seleccionarlo como patrón de entrenamiento y validación en *Control Panel* → *Patterns* → *Training set*. Este fichero contiene tres letras con diferente nivel de ruido (2, 4 y 6 píxeles incorrectos). A continuación, guardar el resultado de la clasificación mediante *File* → *Save data…* → *Archivos de tipo: Result file *.res*. Abrir el fichero generado y comprobar los resultados de la clasificación para cada patrón con diferentes niveles de error en los caracteres obtenidos. ¿Qué caracteres son los más complejos de clasificar? ¿Hay alguno clasificado incorrectamente? ¿Qué nivel de ruido es aceptable? Comenta los resultados.

Ficheros de la práctica

La práctica está compuesta por los siguientes ficheros:

- `JavaNNS-<SistemaOperativo>`: ficheros comprimidos con la aplicación JavaNNS para diferentes sistemas operativos
- `letterstrain.py`: contiene los patrones de entrenamiento de las letras del alfabeto inglés representadas mediante píxeles (26 en total)
- `lettersval.py`: contiene los patrones de validación de la red neuronal entrenada con los patrones de entrenamiento

Entrega de la práctica

Deberán completar las siguientes tareas para considerar la práctica entregada:

1. Realizar los ejercicios propuestos y entregarlos en una memoria con cada apartado debidamente comentado
2. Crear las redes neuronales adecuadas para la resolución de los ejercicios (ficheros *.net), y un fichero de resultados *.res con el proceso de validación

La entrega de esta práctica se realizará en una única modalidad a través de correo electrónico Se deberá entregar un fichero comprimido (ZIP/RAR) con el/los archivos de JavaNNS y la memoria realizada en formato PDF. La fecha límite de entrega será el viernes **08 de enero de 2021**.