PRÁCTICA 3 DE NEUROCOMPUTACIÓN
Curso 2015-2016

Otras aplicaciones de las Redes Neuronales Artificiales

4 de Abril de 2016

Entrega: 26 de Abril

1. Autoencoders

Un autoencoder es una red cuya salida es igual a la entrada.

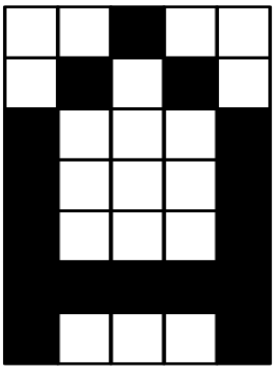
Sus aplicaciones más importantes son la eliminación de ruido y distorsiones en la entrada, y la compresión de datos.

Abre el fichero alfabeto.dat. En él, cada "bloque" son los píxeles que definen una letra del alfabeto, escrita en mayúscula, en una malla de 7x5.

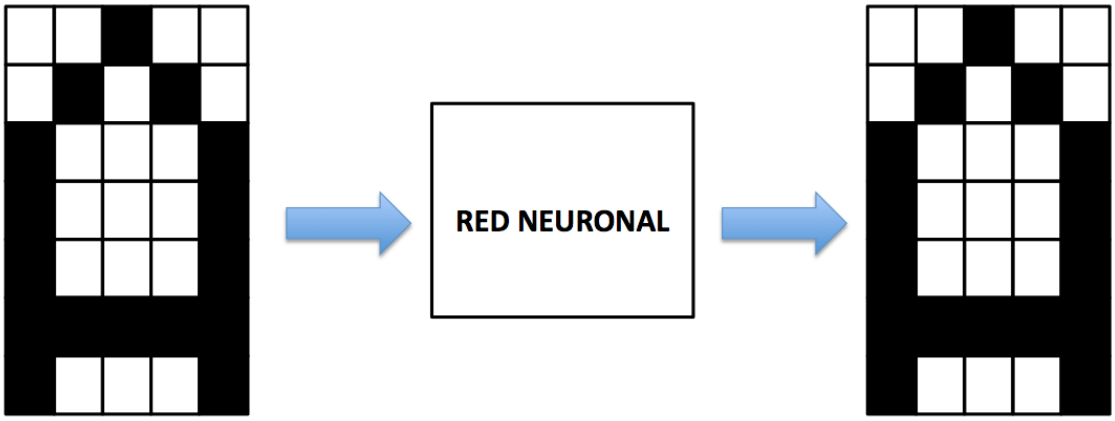
Por ejemplo, el primer bloque:

```
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
```

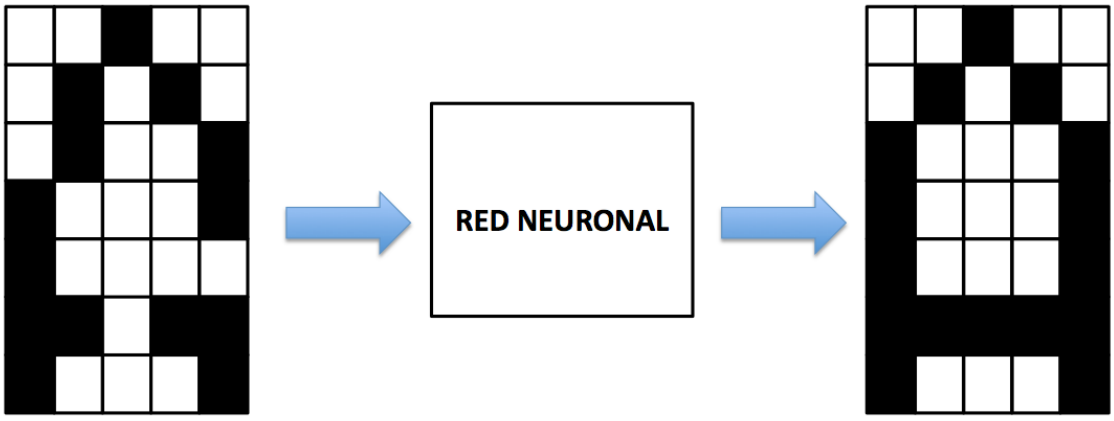
Representa la malla:



Un autoencoder es una red cuya salida esperada es igual a la entrada:



Su utilidad es que, en caso de que en la entrada haya algún tipo de error o diferencia (algo común en aplicaciones donde las letras son escritas a mano y después pixelizadas), la red “trata” de que su salida sea parecida a la salida de “referencia” con la que ha sido entrenada:



1. Toma el fichero alfabeto.dat y creo un fichero alfabeto.txt con el formato utilizado en la práctica anterior. Es decir, cada línea representa un patrón (una letra). En nuestro caso, cada patrón tendrá 35 entradas y 35 salidas. En el fichero de entrenamiento, las salidas son igual a las entradas.

2. Entrena una red neuronal backpropagation tomando como fichero de entrenamiento todo el abecedario, y como fichero de test todo el

abecedario también. Las neuronas ocultas y de salida serán bipolares. Cada neurona de salida representa un píxel, que es negro si la salida de la neurona es mayor que cero, y es blanco en caso contrario. Cambia el cálculo del error final, de tal forma que se compute en explotación el error promedio en términos de número de píxeles errados por letra, "**PE**". ¿Es capaz tu red de aprender todo el abecedario sin cometer un solo error? En caso de que no, ¿qué subconjunto más grande de letras has encontrado que puede aprenderse sin error?

3. Repite el apartado anterior, cambiando la forma de testear: ahora, en el conjunto de test, a cada letra vas a introducirle una pequeña distorsión cambiando el valor de F píxeles al azar. Explora F=1, 3, y 5. Genera para cada letra de test 10 versiones ruidosas. ¿Cuál es el error en test de tu red? ¿"limpia" algo el ruido de las letras? (es decir, ¿se puede hacer **PE < F** ?)

4. Repite el apartado anterior introduciendo en la etapa de entrenamiento ruido aleatorio en las letras, de tal forma que sea parecido al de test. ¿Mejora el resultado de la red?

Sugerencia de resultados a presentar:

- **Caso training = test = abecedario sin ruido**.
  Tabla que muestre PE para varios subconjuntos de letras, incluido el mayor que se pueda aprender perfectamente.

- **Caso training = abecedario sin ruido, test = versión ruidosa del training (x 10).**
  Tabla donde se muestre PE para varios subconjuntos y varios F (1,3,5).

- **Tanto test como training son versiones ruidosas de las letras originales (x 10).** Tabla donde se muestre PE para varios subconjuntos y varios F (1,3,5).

2. Series temporales

En esta parte usaremos las redes neuronales desarrolladas hasta el momento para predecir series temporales. Tenemos dos ficheros de series temporales (p3serie1.txt, p3serie2.800.txt), donde cada fila es un tiempo (los tiempos están ordenados), y en cada fila hay un solo valor.

Usaremos la red neuronal para realizar regresión, y no clasificación como hasta ahora. Para ello:

1. Haz una función *adapta-fichero-serie* que tome como argumentos:

   - Nombre del fichero con la serie original
   - Nombre del fichero que se quiere crear
   - Número de puntos anteriores (Na) que se usan para predecir.
   - Número de puntos siguientes (Ns) que se quieren predecir.

   Dicha función deberá leer el fichero original y crear el fichero nuevo en un formato similar al que hemos usado hasta ahora para definir problemas de clasificación.
   Como ejemplo, supón que el fichero de la serie original contiene sólo 7 puntos (7 filas):

   0.5

   0.3

   0.6

   0.7

   0.1

   0.1

   0.5

   Si Na=1 y Ns=1, la función *adapta-fichero-serie* generará el siguiente fichero:

   1 1

   0.5 0.3

   0.3 0.6

   0.6 0.7

   0.7 0.1

   0.1 0.1

   0.1 0.5

En la primera línea, en este caso, 1 es el número de atributos, y 1 el número de datos a predecir. En este caso la salida que se espera que dé la red neuronal es el valor de la serie en el siguiente instante temporal.

Si ahora Na=2, Ns = 1, generará:

```
2  1
0.5  0.3  0.6
0.3  0.6  0.7
0.6  0.7  0.1
0.7  0.1  0.1
0.1  0.1  0.5
```

Y si ahora Na = 2, Ns = 2 generará:

```
2  2
0.5  0.3  0.6 0.7
0.3  0.6  0.7 0.1
0.6  0.7  0.1 0.1
0.7  0.1  0.1 0.5
```

Por tanto, el número de patrones en el fichero final es igual a:

(número de puntos de la serie temporal original) – Na – Ns + 1.

2. Cambia la configuración de tu red neuronal para que haya tantas neuronas de salida como Ns, que tendrán funciones de transferencia lineal, al ser ahora un problema de regresión.

3. Cambia el código de tu red neuronal para que el conjunto de entrenamiento esté formado por los primeros datos del fichero de entrada a la red, y el conjunto de test por el restante. Esto lo haremos porque ahora sí que hay un orden temporal en el fichero: los primeros datos representan el pasado, lo que usamos para construir el modelo. El test representa datos futuros, lo que queremos predecir.

4. Adapta el código para que compute el error cuadrático medio del modelo tanto en entrenamiento como en test, y para que compute el error cuadrático medio del **modelo más básico para predicción de series temporales**, en el que se predice para todas las componentes del vector de salida el valor más a la derecha del vector de entrada.

Es decir, para el último ejemplo dado arriba (caso Na=2, Ns=2), el modelo más básico predecirá lo siguiente:

Patrón 0.5 0.3 -> predice 0.3 0.3

Patrón 0.3 0.6 -> predice 0.6 0.6

Patrón 0.6 0.7 -> predice 0.7 0.7

Patrón 0.7 0.1 -> predice 0.1 0.1

5. Haz una gráfica de la serie temporal dada en p3serie1.txt. ¿Te parece una serie temporal sencilla?

6. Entrena tu red con un tamaño de entrenamiento de 200 patrones, y un tamaño de test de 200 patrones. ¿Es capaz de predecir adecuadamente tu red? ¿con cuántas neuronas? Prueba Na=1, Na=2 y Na=5, usando Ns = 1.

Compara el error cuadrático medio en test con el error del modelo básico. Haz una gráfica también de las predicciones de tu red comparando con la serie original.

7. Haz lo mismo que en el punto anterior pero usando un tamaño de entrenamiento de 100 patrones y uno de test de 300. ¿Es capaz de predecir ahora adecuadamente tu red? ¿por qué?

8. Implementa una función “predice_recursivamente” que tome como argumentos de entrada:

- el punto de partida con Na componentes
- el número de instantes futuros Nf a predecir

y prediga toda la sucesión de Nf puntos futuros predichos recursivamente por el modelo.

9. Prueba tu función aplicándola a p3serie1.txt con 200 patrones de entrenamiento, 200 de test y Na = 5, Ns = 1. Haz gráficas de la predicción recursiva de tu red a partir de diferentes puntos, prediciendo 50 puntos futuros.

10. Repetir los mismos apartados anteriores con los datos p3serie2.txt y p3serie3.txt, seleccionando Na, Ns y el tamaño de entrenamiento que consideres mejor en cada caso.

Entrega: martes 26 de Abril, 12:00.

Se entregará un zip que contendrá el código fuente y un pdf con la memoria de la práctica (10 caras máximo).