Ajuste y generalización

1. Distintos métodos de ajuste

Vamos a probar ahora distintos métodos de ajuste con *netlab*.

1. Carga los datos e inicializa la red como en la práctica anterior
2. Preparar un vector de parámetros de ajuste con `op=[1 0.01 0.01];` y `op(18)=0;` En este caso el ajuste está limitado por el número de iteraciones, que por defecto es 100, salvo que el avance del vector de pesos y del error sean menores que 0.01.
3. Realizar un ajuste con gradiente conjugado usando `red1=netopt(red,op,p,t,'scg');` Los errores que aparecen son un medio de la suma de cuadrados de los residuos
4. El resultado puede depender del punto de partida aleatorio tomado. Volver a inicializar `red` y probar otra vez. Si queremos, podemos repetir esta prueba con los demás ajustes que vamos a probar. También podemos jugar con los parámetros del modelo, los componentes del vector `op` El número de iteraciones, si lo quieres poner, va en la componente 14
5. Realizar un ajuste con cuasi-newton usando `red2=netopt(red,op,p,t,'quasinew');` Hay un parámetro adicional, que es la precisión de la búsqueda lineal, que va en la componente 15 del vector de opciones; su valor por defecto es 0.01
6. Ahora vamos a ver el efecto de regularización, es decir cuando se incluye en el objetivo que los pesos no crezcan descontroladamente. Realizar un ajuste con coste por pesos haciendo una inicialización con `redbay1=mlp(nin,5,nout,'linear',0.01);` y luego ajustarla usando `red3=netopt(redbay1,op,p,t,'scg');` Ese 0.01 juega con él
7. Realizar un ajuste bayesiano, donde la importancia del valor de los pesos no es fija, como en el caso anterior sino que intenta aproximar su máxima verosimilitud dinámicamente, haciendo una inicialización con `redbay2=mlp(nin,5,nout,'logistic',0.01,50);` y luego ajustarla usando `red4=netoptbr(redbay2,op,p,t,'scg');` `alfa` es el inverso de la varianza de los pesos, `beta` es el inverso de la varianza de los datos y `gamma` es el número de pesos bien determinados.
8. Podemos utilizar estas últimas aproximaciones para calcular una evidencia de red dada la muestra e ir variando el número de procesadores en la capa oculta, buscando máxima evidencia. Para ello, en el caso de usar una optimización que no sea la regresión bayesiana, hay que repetir dos o tres veces la secuencia:

```
params(1) = 1;
params(2) = 1.0e-5;% Precision para pesos
params(3) = 1.0e-5;% Precision para el error
params(14) = 100;% Ciclos de optimización
params(18)=0;
rn=netopt(rn,params,p,t,'scg'); %u otro método
[rn, numpardeter, logev] = evidence(rn, p, t, 2);
```

Con esto obtendremos un valor de `logev` que puede depender del punto de partida aleatorio, con lo que podríamos repetir todo y comprobar. Luego, cambiamos la cantidad de procesadores ocultos, creamos otra red y repetimos el proceso. Nos quedamos finalmente con la cantidad de procesadores ocultos donde la evidencia sea máxima.

1.1. Parada anticipada

Este es un método para evitar el sobreajuste, que puede ser que en este caso no sea necesario.

La idea es que, teniendo en cuenta la evolución típica de los errores en los conjuntos de ajuste y validación, el sobreajuste puede evitarse deteniendo anticipadamente el ajuste. El problema que nos queda es determinar ese punto de detención.

Una primera posibilidad es separar unos puntos e ir midiendo el error en ellos; cuando empiece a subir detenemos el ajuste.

Conjunto de validación por agrupamiento Para minimizar la influencia de la suerte, podríamos escoger esos puntos por el método de agrupamiento. Ni `netlab` ni la toolbox propia de Matlab tienen esto implementado, así que recurriremos a la toolbox del curso.

```
red2=optimval(p,5,t);
t2prueba=ejecutapm(red2,pprueba');
plot(tprueba,t2prueba'-tprueba,'xb',tprueba,eje,'r')
```

Pruebas repetidas Lo anterior tiene un problema: la red ha tenido que renunciar a usar la información de todos los puntos, para estimar el sobreajuste. Una manera de evitar esto es: estimar mediante varias pruebas aleatorias el error de ajuste en que deberíamos parar, y luego utilizar todos los puntos en un ajuste final, que se detiene en el error calculado previamente.

```
redbs=optimbs(p,5,t);
tbs=ejecutapm(redbs,pprueba');
plot(tprueba,tbs'-tprueba,'xr',tprueba,eje,'b')
```

2. Generalización

2.1. Error puntual de predicción

Vamos a estimar el error de predicción de la red. Siguiendo las aproximaciones de la regresión no lineal, el error esperable en predicción es:

$$t_{n-p}^{1-\frac{\alpha}{2}} s_\epsilon \sqrt{1 + g_0'(F'F)^{-1}g_0}$$

donde $t_{n-p}^{1-\frac{\alpha}{2}}$ es el cuantil $\frac{\alpha}{2}$ de la distribución t de Student con $n-p$ grados de libertad (n: número de puntos en la muestra, p: cantidad de pesos de la red), s_ϵ es la desviación típica estimada de la muestra ($\sqrt{\frac{\sum_{i=1}^{n}(t_i-\hat{y}_i)^2}{n-1}}$), g_0 es el gradiente (columna) respecto a los pesos en el punto que nos interesa y F es el jacobiano de la salida respecto a los pesos (filas: puntos, columnas : pesos).

Sea p_0 el punto donde queremos evaluar el error de predicción a un nivel $\frac{\alpha}{2}$. Teniendo en cuenta la gran cantidad de grados de libertad, un valor de t común es 1.97 Con `netlab` procederemos:

```
y=mlpfwd(red,p);
jacob=mlpderiv(red,p);
prodjacon=jacob'*jacob;
grad0=mlpderiv(red,p0);
estimvar=(t-y)'*(t-y)/(ndata-1);
estimdesv=sqrt(estimvar)
rang0=1.97*estimdesv*sqrt(1+grad0*(prodjacon\(grad0')))
```

2.2. Muestreo de error de generalización

Es normal que nos planteemos qué tal sale la predicción de nuestro modelo. El error residual de ajuste está bien para chequeos, pero está sesgado para el error previsible del modelo.

Podemos ver un poco mejor qué tal lo hace la red, con los gráficos que nos presentan los valores dados por la red frente a los reales, usando el subconjunto no empleado en el ajuste.

Esa estimación del comportamiento en generalización de la red puede depender de los puntos que hayan caído en el conjunto de prueba. Nuevamente para evitar esto podemos recurrir a varias selecciones aleatorias y tomar una medida central.

```
[red4 msepru]=optimest(p,5,t);
t4prueba=ejecutapm(red4,pprueba');
norm(t4prueba'-tprueba,2)/length(tprueba)
plot(tprueba,t4prueba'-tprueba,'xr',tprueba,eje,'b')
```

En este caso la toolbox hace tres pruebas con distintos puntos iniciales aleatorios y se queda con el mejor.

Compara la estimación que tendríamos con los residuos en el conjunto de ajuste, la que sale de este método y las puntuales del apartado anterior.