# Red neuronal monocapa de reconociemiento de dígitos

Alex Ferrando de las Morenas | Elías Abad Rocamora

tr_seed = 230580 | te_seed = 158096

En esta práctica, se pretende estudiar diferentes métodos de optimización sin constricciones a través de una red neuronal monocapa de reconocimiento de números.

Para entrenar la red se introducen como datos de entrenamiento un conjunto de imágenes de 7x5 píxeles de algún número distorsionado entre el 0 y el 9. A cada valor de píxel de entrada le aplicamos una función de activación sigmoide que mapea cada valor de píxel entre 0 i 1. Cada resultado se multiplica por una variable $w_i$ que representa un peso sobre cada píxel. Estos resultados se suman finalmente en una última neurona y se vuelve a aplicar una sigmoide sobre el resultado. Finalmente se redondea el valor obtenido; si es un 1, la red dirá que la imágen de entrada sí que corresponde al número target indicado. Si por el contrario el resultado es 0, la red devolverá que la imágen de entrada no corresponde con el target indicado.

El objetivo final es conseguir encontrar la asignación óptima para los pesos $w_i$. Para ello, se pretende encontrar los pesos que minimicen el número de predicciones incorrectas de la red sobre el conjunto de datos de entrenamiento. Definimos la función objetivo (*loss function*) de la siguiente forma:

$$L(w; X^{TR}, y^{TR}, \lambda) = \sum_{j=1}^{p} (y(x_j^{TR}, w) - y_j^{TR})^2 + \lambda \frac{||w||^2}{2}$$

Se le añade un parámetro de regularización $\lambda \frac{||w||^2}{2}$ para generar un sesgo en el modelo y conseguir una mayor generalización del mismo evitando un posible *overfitting*.

Para la búsqueda del conjunto de pesos que minimizan la función pérdida utilizaremos un conjunto de algoritmos vistos en la asignatura y evaluaremos sus propiedades a partir de los resultados que obtengamos. Los algoritmos serán:

**1.** Método del gradiente. **2.** Método del gradiente conjugado. **3.** BFGS **4.** Método del gradiente estocástico.

## 1. Análisis de la convergencia

Mostramos a continuación el *batch* obtenido tras ejecutar el algoritmo. En el dataset de salida se muestran en order cada *num_target* entre el 0 i el 9 i, para cada número las combinaciones de *lambdas* y métodos de cálculos de puntos óptimos posibles. Además, se añade otra información como ahora el número de iteraciones de cada ejecución, la precisión de predicción sobre el conjunto de entrenamiento y test o el valor de la función objetivo en el punto óptimo.

```
##    num_target la isd niter    tex tr_acc te_acc       L.       iter_t
## 1           1  0   1     2 0.0479  100.0  100.0 8.81e-08 2.395000e-02
## 2           1  0   3     2 0.0413  100.0  100.0 8.81e-08 2.065000e-02
## 3           1  0   7     8 0.0039  100.0  100.0 1.46e-01 4.875000e-04
## 4           1  1   1    76 0.3133  100.0  100.0 3.24e+00 4.122368e-03
```

```
## 5          1  1   3    18 0.1090  100.0  100.0 3.24e+00 6.055556e-03
## 6          1  1   7  1000 0.0855  100.0  100.0 1.04e+01 8.550000e-05
## 7          1 10   1    41 0.2372  100.0  100.0 1.37e+01 5.785366e-03
## 8          1 10   3    38 0.2163  100.0  100.0 1.37e+01 5.692105e-03
## 9          1 10   7  1000 0.0452  100.0  100.0      Inf 4.520000e-05
## 10         2  0   1   336 0.9612  100.0   99.6 1.32e-06 2.860714e-03
## 11         2  0   3   132 0.4717  100.0   98.0 3.26e-07 3.573485e-03
## 12         2  0   7    58 0.0025   99.6   99.2 9.37e-01 4.310345e-05
## 13         2  1   1   252 0.7688   99.6   99.2 7.10e+00 3.050794e-03
## 14         2  1   3    25 0.1235   99.6   99.2 7.10e+00 4.940000e-03
## 15         2  1   7  1000 0.0373   95.6   92.4 1.87e+01 3.730000e-05
## 16         2 10   1  1000 1.7993   97.2   93.2 2.15e+01 1.799300e-03
## 17         2 10   3    41 0.1573   97.2   93.2 2.15e+01 3.836585e-03
## 18         2 10   7  1000 0.0364   89.6   81.6      Inf 3.640000e-05
## 19         3  0   1  1000 2.7526  100.0   98.4 5.82e-03 2.752600e-03
## 20         3  0   3    22 0.0972  100.0   95.6 8.78e-14 4.418182e-03
## 21         3  0   7    20 0.0013   58.4   22.8 1.04e+02 6.500000e-05
## 22         3  1   1   492 1.5021   99.6   99.6 1.12e+01 3.053049e-03
## 23         3  1   3    30 0.1481   99.6   99.6 1.12e+01 4.936667e-03
## 24         3  1   7  1000 0.0555   90.8   87.6 2.78e+01 5.550000e-05
## 25         3 10   1    89 0.3076   98.0   99.6 2.98e+01 3.456180e-03
## 26         3 10   3    38 0.1705   98.0   99.6 2.98e+01 4.486842e-03
## 27         3 10   7  1000 0.0504   54.4    9.6      Inf 5.040000e-05
## 28         4  0   1     4 0.0387  100.0   99.6 8.55e-13 9.675000e-03
## 29         4  0   3     4 0.0459  100.0   99.6 1.41e-14 1.147500e-02
## 30         4  0   7     5 0.0003   98.8   99.2 1.50e+00 6.000000e-05
## 31         4  1   1    95 0.2817  100.0  100.0 3.26e+00 2.965263e-03
## 32         4  1   3    19 0.0964  100.0  100.0 3.26e+00 5.073684e-03
## 33         4  1   7  1000 0.0458  100.0  100.0 1.03e+01 4.580000e-05
## 34         4 10   1    58 0.1913  100.0  100.0 1.35e+01 3.298276e-03
## 35         4 10   3    36 0.1590  100.0  100.0 1.35e+01 4.416667e-03
## 36         4 10   7  1000 0.0438   90.8   87.6      Inf 4.380000e-05
## 37         5  0   1    31 0.1045  100.0  100.0 4.97e-07 3.370968e-03
## 38         5  0   3    80 0.2551  100.0   99.6 2.66e-07 3.188750e-03
## 39         5  0   7    80 0.0038  100.0  100.0 9.42e-03 4.750000e-05
## 40         5  1   1   155 0.4614  100.0  100.0 4.32e+00 2.976774e-03
## 41         5  1   3    27 0.1278  100.0  100.0 4.32e+00 4.733333e-03
## 42         5  1   7  1000 0.0379  100.0  100.0 1.38e+01 3.790000e-05
## 43         5 10   1    73 0.2538  100.0  100.0 1.72e+01 3.476712e-03
## 44         5 10   3    38 0.2360  100.0  100.0 1.72e+01 6.210526e-03
## 45         5 10   7  1000 0.0454   79.6   57.2      Inf 4.540000e-05
## 46         6  0   1   206 0.6092  100.0   99.2 7.58e-07 2.957282e-03
## 47         6  0   3    20 0.1897  100.0   94.8 4.22e-39 9.485000e-03
## 48         6  0   7    31 0.0020   99.6   99.2 3.56e-01 6.451613e-05
## 49         6  1   1   234 0.7486  100.0   99.2 5.82e+00 3.199145e-03
## 50         6  1   3    26 0.1303  100.0   99.2 5.82e+00 5.011538e-03
## 51         6  1   7  1000 0.0449  100.0   98.4 1.74e+01 4.490000e-05
## 52         6 10   1  1000 1.8041  100.0   99.2 2.06e+01 1.804100e-03
## 53         6 10   3    40 0.1749  100.0   99.2 2.06e+01 4.372500e-03
## 54         6 10   7  1000 0.0386   80.0   57.6      Inf 3.860000e-05
## 55         7  0   1     3 0.0313  100.0   99.6 1.32e-07 1.043333e-02
## 56         7  0   3     3 0.0355  100.0   99.6 8.08e-08 1.183333e-02
## 57         7  0   7     9 0.0005   99.6   99.6 8.02e-01 5.555556e-05
## 58         7  1   1   105 0.3412  100.0   99.6 3.89e+00 3.249524e-03
```

```
## 59          7  1   3    18 0.1137  100.0   99.6 3.89e+00 6.316667e-03
## 60          7  1   7  1000 0.0394  100.0   99.6 1.19e+01 3.940000e-05
## 61          7 10   1    44 0.1730  100.0   99.6 1.50e+01 3.931818e-03
## 62          7 10   3    34 0.1829  100.0   99.6 1.50e+01 5.379412e-03
## 63          7 10   7  1000 0.0355   94.0   84.8      Inf 3.550000e-05
## 64          8  0   1  1000 2.7911  100.0   96.4 1.04e-01 2.791100e-03
## 65          8  0   3    19 0.0764  100.0   91.6 1.52e-07 4.021053e-03
## 66          8  0   7   113 0.0056   90.0   84.8 1.66e+01 4.955752e-05
## 67          8  1   1   711 1.9076   98.8   97.6 1.39e+01 2.682982e-03
## 68          8  1   3    29 0.1311   98.8   97.6 1.39e+01 4.520690e-03
## 69          8  1   7  1000 0.0358   87.6   82.0 3.21e+01 3.580000e-05
## 70          8 10   1   142 0.3906   98.0   96.4 3.41e+01 2.750704e-03
## 71          8 10   3    42 0.1753   98.0   96.4 3.41e+01 4.173810e-03
## 72          8 10   7  1000 0.0374   54.0   12.8      Inf 3.740000e-05
## 73          9  0   1  1000 2.4126  100.0   98.4 1.54e-06 2.412600e-03
## 74          9  0   3    13 0.0791  100.0   94.8 4.60e-15 6.084615e-03
## 75          9  0   7   256 0.0117  100.0   98.0 6.74e-02 4.570313e-05
## 76          9  1   1   470 1.3677   99.6   98.4 9.57e+00 2.910000e-03
## 77          9  1   3    29 0.1652   99.6   98.4 9.57e+00 5.696552e-03
## 78          9  1   7  1000 0.0464   98.0   96.8 2.55e+01 4.640000e-05
## 79          9 10   1   119 0.4635   99.2   97.6 2.82e+01 3.894958e-03
## 80          9 10   3    40 0.1825   99.2   97.6 2.82e+01 4.562500e-03
## 81          9 10   7  1000 0.0539   54.0    9.6      Inf 5.390000e-05
## 82          0  0   1   282 0.7776  100.0   99.6 7.89e-07 2.757447e-03
## 83          0  0   3    16 0.1120  100.0   97.6 7.98e-28 7.000000e-03
## 84          0  0   7     7 0.0005   52.4   10.0 1.19e+02 7.142857e-05
## 85          0  1   1   314 0.8909  100.0   99.6 6.98e+00 2.837261e-03
## 86          0  1   3    41 0.1790  100.0   99.6 6.98e+00 4.365854e-03
## 87          0  1   7  1000 0.0378  100.0   98.8 2.11e+01 3.780000e-05
## 88          0 10   1   110 0.3448  100.0   99.6 2.42e+01 3.134545e-03
## 89          0 10   3    37 0.2142  100.0   99.6 2.42e+01 5.789189e-03
## 90          0 10   7  1000 0.0346   52.4   10.0      Inf 3.460000e-05
```

## 1.1 Convergencia global

La función objetivo a minimizar es el error cuadrático, que sabemos que es convexa por ser una norma al cuadrado entre el vector de resultados observados y predecidos por nuestro modelo. A la función se le suma una lambda (parámetro de regularización) por la norma 2 del vector de pesos al cuadrado. Entonces, en ser la Hessiana de $\lambda\frac{||w||^2}{2}$ igual a $\lambda Id$ (semidefinida positiva) cuando $\lambda \geq 0$, la Hessiana de $L$ será, obligatoriamente, semidefinida positiva y, por lo tanto, podemos afirmar que $L$ será convexa.

Como sabemos que $L$ es convexa, podemos asegurar que si un algoritmo converge a un punto estacionario éste será mínimo local y global. Cuando $\lambda = 0$ el valor mínimo de $L$ es 0 y cuando es 1 o 10, podemos coger como mínimo global el valor conseguido con el método del gradiente o BFGS cuando éstos converjan, es decir, cuando el número de iteraciones sea menor que *kmax* (1000).

Table 1: Target = 0

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 8.00e-07 | 0.00 | 119.0 |
| 1 | 6.98e+00 | 6.98 | 21.1 |
| 10 | 2.42e+01 | 24.20 | Inf |

Table 2: Target = 1

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 1.00e-07 | 1.00e-07 | 0.146 |
| 1 | 3.24e+00 | 3.24e+00 | 10.400 |
| 10 | 1.37e+01 | 1.37e+01 | Inf |

Table 3: Target = 2

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 1.30e-06 | 3.00e-07 | 0.937 |
| 1 | 7.10e+00 | 7.10e+00 | 18.700 |
| 10 | 2.15e+01 | 2.15e+01 | Inf |

Table 4: Target = 3

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 0.00582 | 0.0 | 104.0 |
| 1 | 11.20000 | 11.2 | 27.8 |
| 10 | 29.80000 | 29.8 | Inf |

Table 5: Target = 4

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 0.00 | 0.00 | 1.5 |
| 1 | 3.26 | 3.26 | 10.3 |
| 10 | 13.50 | 13.50 | Inf |

Table 6: Target = 5

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 5.00e-07 | 3.00e-07 | 0.00942 |
| 1 | 4.32e+00 | 4.32e+00 | 13.80000 |
| 10 | 1.72e+01 | 1.72e+01 | Inf |

Table 7: Target = 6

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 8.00e-07 | 0.00 | 0.356 |
| 1 | 5.82e+00 | 5.82 | 17.400 |
| 10 | 2.06e+01 | 20.60 | Inf |

Table 8: Target = 7

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 1.00e-07 | 1.00e-07 | 0.802 |
| 1 | 3.89e+00 | 3.89e+00 | 11.900 |
| 10 | 1.50e+01 | 1.50e+01 | Inf |

Table 9: Target = 8

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 0.104 | 2.00e-07 | 16.6 |
| 1 | 13.900 | 1.39e+01 | 32.1 |
| 10 | 34.100 | 3.41e+01 | Inf |

Table 10: Target = 9

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 1.50e-06 | 0.00 | 0.0674 |
| 1 | 9.57e+00 | 9.57 | 25.5000 |
| 10 | 2.82e+01 | 28.20 | Inf |

Para el estudio de la convergencia global en los tres métodos, nos basaremos en un hecho fundamental. Como hemos explicado anteriormente, la función objetivo es convexa. Podemos afirmar entonces que si el método del gradiente converge antes de las 1000 iteraciones este llegará a un mínimo local y global de la función pérdida. Además, si usamos el método BFGS o el método del gradiente, como el algoritmo que utilizamos para encontrar la longitud de paso busca que se satisfagan las $SWC$, la convergencia global está asegurada y por lo tanto sabemos que si estos métodos convergen a algún punto antes de las 1000 iteraciones, por ser la función convexa, este punto deberá representar obligatoriamente un mínimo global de la función. Así pues, esto nos proporciona un punto de partida comparativo para ver si los diferentes métodos convergen globalmente.

Si observamos los resultados, podemos ver que el método BFGS (isd = 3) siempre converge antes de las 1000 iteraciones, por lo tanto, los demás también lo harán si el valor de L* és el mismo que el mostrado por el algoritmo cuasi Newton. Vemos, consecuentemente, que el método del gradiente, al presentar valores superiores de L* y llegar a k = 1000, no siempre consigue coverger al óptimo global. Sin embargo, si dejáramos $k \to \infty$ el método del gradiente tenderá al mismo punto de convergencia que muestra BFGS.

En el caso que usemos el método del gradiente estocástico (isd = 7) la ejecución nunca converge hasta el mínimo global. Vemos que para $\lambda = \{0, 1\}$, casi nunca se acerca al mínimo obtenido con BFGS, esto tiene sentido porque este método utiliza cómo dirección de descenso el gradiente de la muestra aleatoria, puede ser que esta dirección sea de descenso para la función con $m$ muestras, pero no para la función que estamos minimizando (con todas las muestras).

Dicho esto, en los casos $\lambda = 0$ y $\lambda = 1$ las soluciones obtenidas no son del todo malas. El problema viene con $\lambda = 10$. En este caso, para cualquier target, obtenemos siempre un valor de la Loss function que tiende a infinito. Esto se debe a que la learning rate no es la adecuado para la función objetivo con este valor de $\lambda$.

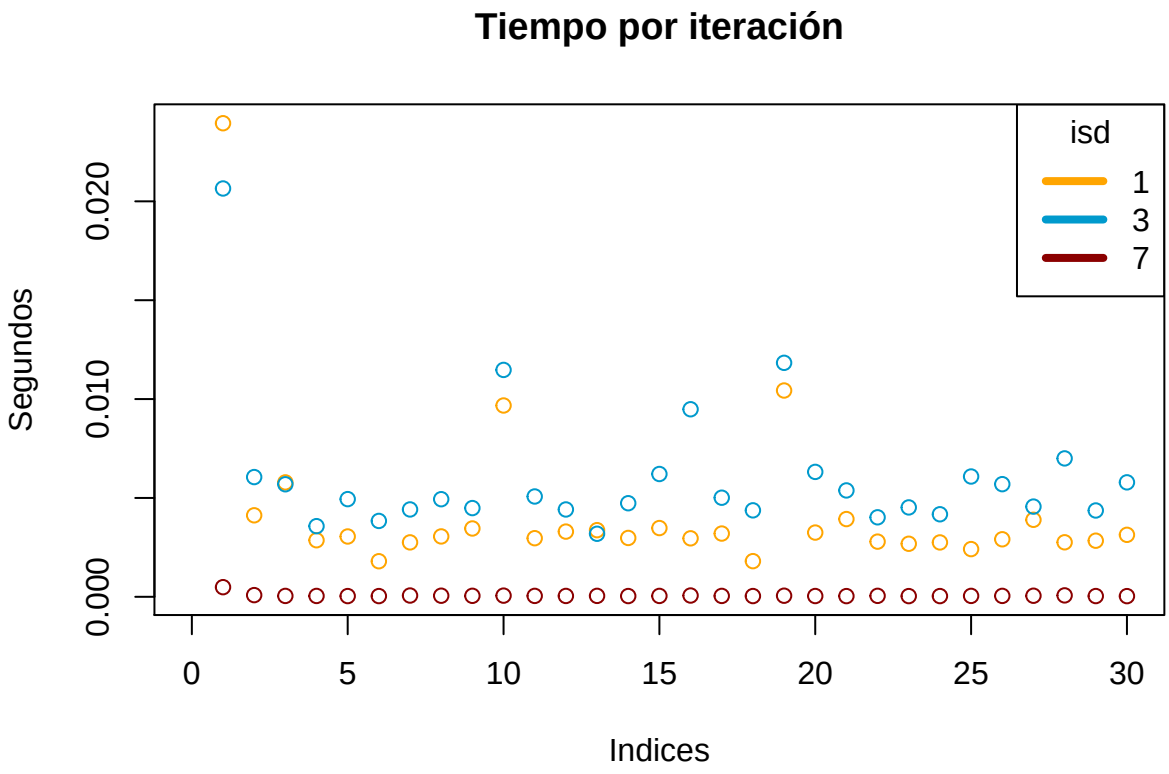
Cuánto mayor es $\lambda$, la función objetivo se vuelve más "puntiaguda", se acerca más al valor $w = (0, 0, ..., 0)^T$ y su valor mínimo es mayor. El hecho de que sea más puntiaguda, hace que la tasa de aprendizaje no sea adecuada ya que esta longitud de paso, que para otro valor más pequeño de $\lambda$ nos llevaba a un punto dónde se satisfacián las *WC*, en este caso puede pasar justo lo contrario y acabar en un punto con valor de la función objetivo mayor y mayor módulo del gradiente. Esto ocurre en prácticamente cada iteración, haciendo que el valor final de la loss function sea infinito.

### 1.2 Convergencia local

Planteamos una tabla del número de iteraciones en función de la $\lambda$ y el *isd*.

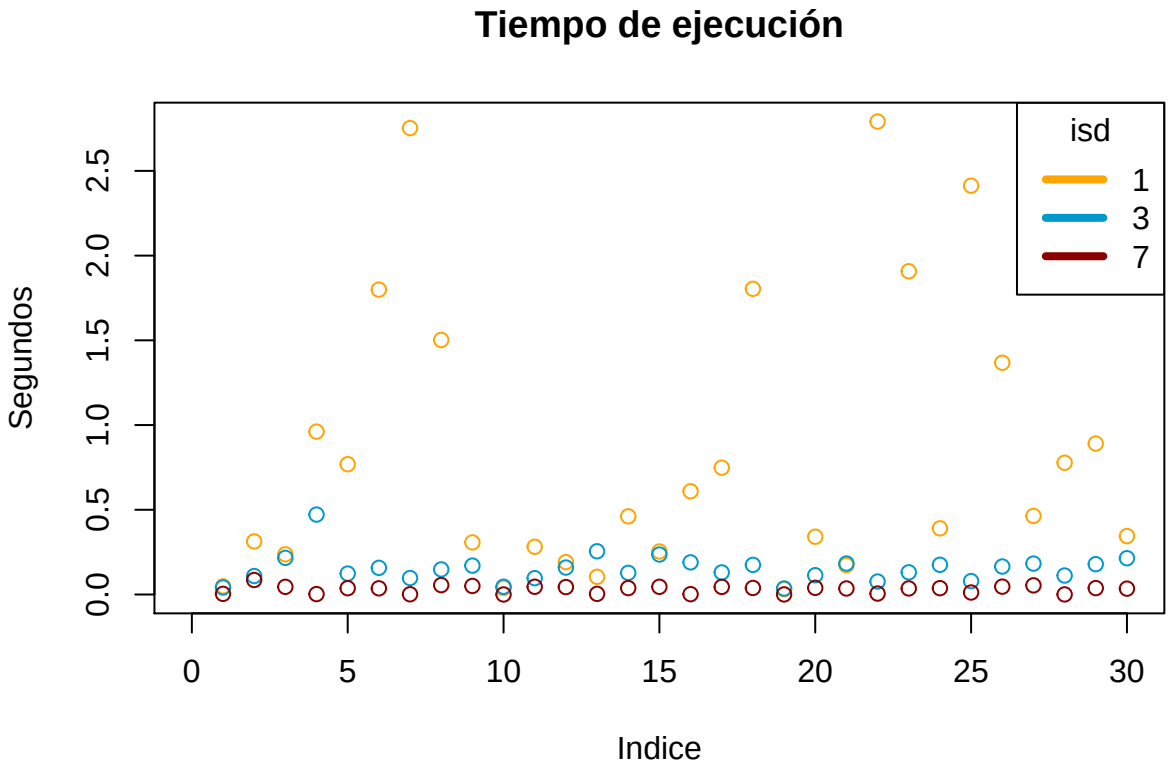
| isd = | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 386.4 | 31.1 | 58.7 |
| 1 | 290.4 | 26.6 | 1000 |
| 10 | 267.6 | 38.4 | 1000 |

Vemos que al tener convergencia superlineal (por cumplirse en las iteraciones las *WC* podemos asegurar la convergencia global) el método BFGS hace significativamente menos iteraciones que el método del gradiente. Por otra parte, el método del gradiente estocástico, salvo para $\lambda = 0$ hace un número de iteraciones mucho mayor que los otros métodos. De hecho, hace el máximo número de iteraciones permitido por el algoritmo (1000).

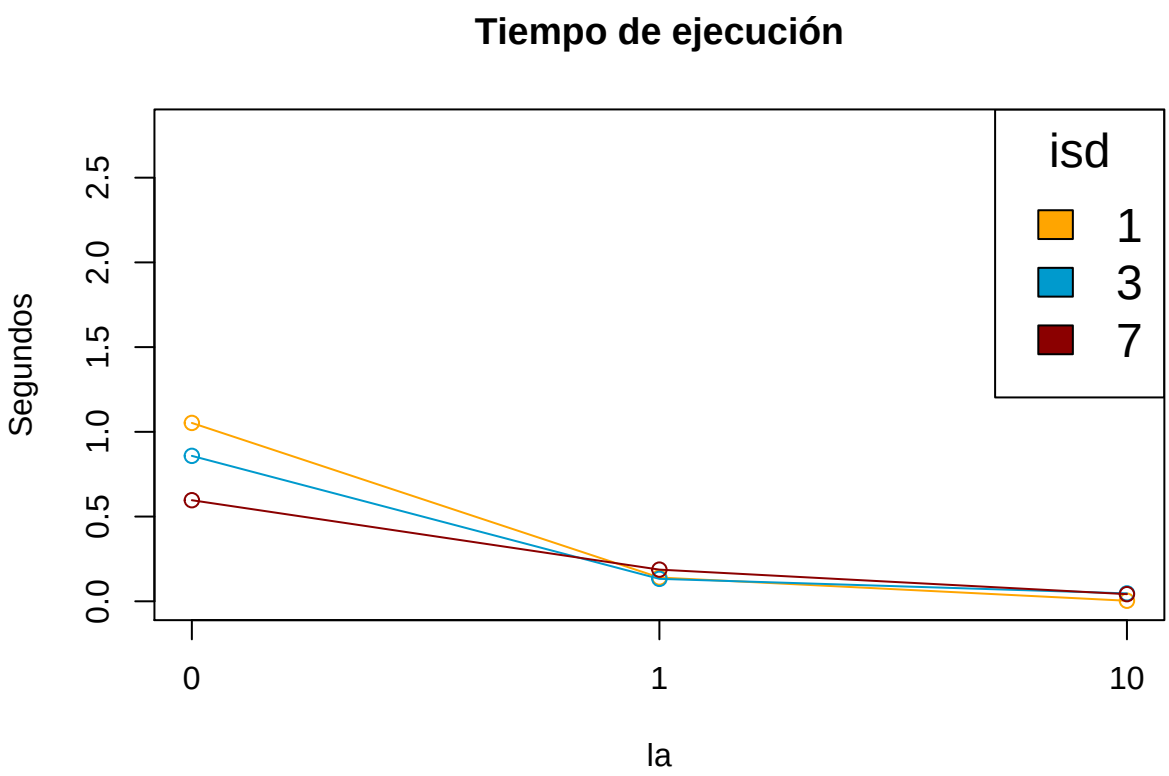



En cuánto a la convergencia local, podemos decir que el método 7 es con diferencia el más rápido por iteración, ya que no evalua el gradiente con todas las muestras sino con un número mucho menor y utiliza

una longitud de paso fija. Por otro lado, el más lento es BFGS ya que a cada iteración tiene que calcular la aproximación de la matriz Hessiana y calcular mediante BLS la longitud de paso.




En la gráfica de tiempo de ejeución podemos observar que el método generalmente més lento es el método del gradiente pues aunque sea más rápido que BFGS en términos de tiempo por iteración, la convergencia local con el método del gradiente es más lenta y, por lo tanto, hace muchas más iteraciones en promedio. Por otra parte, el método del gradiente estocástico, es el más rápido en absolutamente todas las pruebas realizadas aunque no con gran diferencia respecto a los otros métodos. Esto sucede porque aunque se llegue casi siempre a las 1000 iteraciones, como hemos observado en la gráfica anterior, el tiempo por iteración es tan pequeño en comparación con los otros dos métodos que el tiempo total se mantiene más bajo.




Si consideramos, a parte del *isd*, la $\lambda$ utilizada, vemos que a medida que augmentamos la lambda, el tiempo de ejecución del algoritmo se reduce. Este punto será importante más tarde para evaluar el *trade-off* entre *accuracy* y tiempo de ejecución.

**Conclusiones**

Concluimos, entonces que el método BFGS es el que mejor comportamiento muestra ya que es el que converge a la solución óptima siempre y en un tiempo de ejecución notablemente bajo. Aunque sea el algoritmo que tarda más por iteración, converge suficientemente rápido al óptimo global (convergencia superlineal) por ser $f \in C^2$, el algoritmo converger a un mínimo y ser la Hessiana Lipschitz continua.

Por el contrario, el método del gradiente, muestra un tiempo por iteración menor que el método casi-Newton, pero en tener convergencia lineal, hace significativamente más iteraciones. Este método, convergerá al óptimo global por ser la dirección de avance siempre de descenso y cumplir la *CAC*. Sin embargo, al establecer un máximo de iteraciones para el algoritmo, el método no consigue siempre converger, a diferencia que el método BFGS, suficientemente rápido al mínimo de la función.

Por último, el método estocástico, pese a ser el método más rápido de todos, no garantiza la convergencia global. De hecho, en ninguna ejecución del *batch* llega al óptimo global. Aun así para *lambdas* pequeñas da resultados suficientemente correctos. Sin embargo, para este experimento realizado, el método del gradiente estocástico no aporta una mejora de tiempo tan importante como para no usar los otros dos métodos que sí que nos aseguran la convergencia al óptimo. De todas formas, si se nos planteará un problema con un conjunto observacional muy grande, donde el coste computacional para el cálculo del gradiente fuera muy grande, entonces este método seria de gran utilidad para obtener un resultado preliminar de forma rápida.

## 2. Análisis de rendimiento

**Accuracy para cada número en función de la e isd**

Table 12: Target = 0

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100 | 100 | 52.4 |
| 1 | 100 | 100 | 100.0 |
| 10 | 100 | 100 | 52.4 |

Table 13: Target = 1

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100 | 100 | 100 |
| 1 | 100 | 100 | 100 |
| 10 | 100 | 100 | 100 |

Table 14: Target = 2

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100.0 | 100.0 | 99.6 |
| 1 | 99.6 | 99.6 | 95.6 |
| 10 | 97.2 | 97.2 | 89.6 |

Table 15: Target = 3

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100.0 | 100.0 | 58.4 |
| 1 | 99.6 | 99.6 | 90.8 |
| 10 | 98.0 | 98.0 | 54.4 |

Table 16: Target = 4

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100 | 100 | 98.8 |
| 1 | 100 | 100 | 100.0 |
| 10 | 100 | 100 | 90.8 |

Table 17: Target = 5

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100 | 100 | 100.0 |
| 1 | 100 | 100 | 100.0 |
| 10 | 100 | 100 | 79.6 |

Table 18: Target = 6

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100 | 100 | 99.6 |
| 1 | 100 | 100 | 100.0 |
| 10 | 100 | 100 | 80.0 |

Table 19: Target = 7

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100 | 100 | 99.6 |
| 1 | 100 | 100 | 100.0 |
| 10 | 100 | 100 | 94.0 |

Table 20: Target = 8

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100.0 | 100.0 | 90.0 |
| 1 | 98.8 | 98.8 | 87.6 |
| 10 | 98.0 | 98.0 | 54.0 |

Table 21: Target = 9

| | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 100.0 | 100.0 | 100 |
| 1 | 99.6 | 99.6 | 98 |
| 10 | 99.2 | 99.2 | 54 |

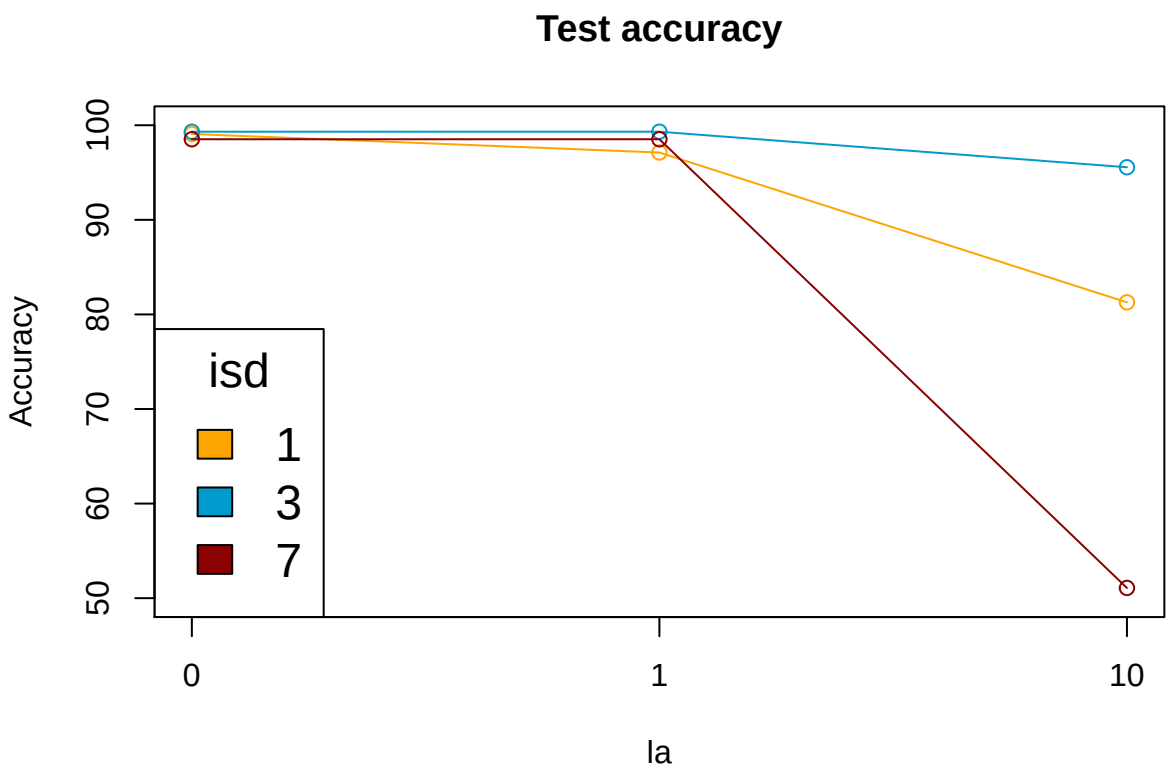
**Accuracy media para cada target**

```
##         0         1         2         3         4         5         6         7
##  79.37778 100.00000  95.06667  79.15556  98.44444  95.20000  94.00000  97.95556
##         8         9
##  83.95556  87.73333
```

Vemos que en promedio el dígito 3 es el que peor porcentaje de acierto en la predicción tiene. Sin embargo, si miramos detalladamente la tabla de accuracies para cada número vemos que es el 8, pues aunque en media

no sea el que peor se comporta, es el que muestra un peor comportamiento general en la predicción. Esto es debido a que para el método del gradiente estocástico los dígitos 0 y 3, para algunos valores de $\lambda$ obtienen porcentajes de acierto sumamente bajos (entorno al 10%).

Veamos ahora el test accuracy medio para cada combinación de $\lambda$ y *isd*:




| isd = | 1 | 3 | 7 |
|---|---|---|---|
| 0 | 99.08 | 97.12 | 81.28 |
| 1 | 99.32 | 99.32 | 95.56 |
| 10 | 98.52 | 98.52 | 51.08 |

Como podemos observar en los valores de precisión obtenidos en las predicciones sobre el conjunto de test, volvemos a comprobar que el método BFGS es el mejor con diferencia. Para $\lambda = 0$ i $\lambda = 1$ observamos que todos los métodos muestran accuracies cercanas al 100% por lo que podemos afirmar que todos funcionan correctamente. Sin embargo, la diferencia entre los tres métodos utilizados se hace notoria en augmentar la $\lambda$. Para $\lambda = 10$, vemos grandes diferencias. Al augmentar esta variable, creamos un sesgo sobre la predicción, que puede explicar un augmento en el error. Además, como hemos explicado anteriormente, la función se vuelve más *puntiaguda* y el método del gradiente estocástico tiende a no converger, de ahí que muestre un error tan grande.

Si además, nos fijamos en el tiempo de ejecución para cada $\lambda$, como hemos visto anteriormente, a medida que aumentamos la variable penalizadora, el tiempo de ejecución disminuye. Si queremos considerar un *trade-off* bueno entre tiempo de ejecución y precisión de la predicción la mejor elección de $\lambda$ es 1. Observamos que tanto para BFGS como para el método del gradiente estocástico, la precisión de predicción sobre el conjunto de test es igual para *lambda* 0 y 1. Por lo tanto, al escoger $\lambda = 1$ mantenemos la precisión pero disminuímos tiempo de ejecución.

**Estudio caso dígito 8**

Observando los resultados anteriores vemos que el 8 es más difícil de identificar que los demás números en general, independientemente de la combinación algoritmo-$\lambda$. Evaluaremos el porqué de este comportamiento utilizando los resultados del algoritmo al utilizar el método BFGS i $\lambda = 1$.

Figure 1: Resultados num_target = 8, isd = 3, la = 1

Como podemos ver en la imagen, nuestro algorítmo comete varios positivos falsos (dígitos distintos de 8 clasificados como 8). Esto se puede deber a que algunos dígitos como el 3, el 6 o el 9, al ser difuminados pueden ser confundidos fácilmente con un 8.

Si nos fijamos en los pesos de la red neuronal (*Figure 2*) podemos ver que los valores de las posiciones (4,1) y (4,5) (los más oscuros), son los que más influyen negativamente en que un dígito sea clasificado cómo un 8 o no, ya que el dígito 8 no utiliza esos píxels. Por otro lado, el píxel (5,1) el que más contribuye positivamente (más claro).

En definitiva, la combinación de valores bajos en las posiciones (4,1) y (4,5) y un valor alto en la posición (5,1), es la más característica del dígito 8.

Los dígitos 3, 6 y 9 correlan fuertemente con las características mencionadas y presentan importantes similitudes en otra cantidad grande de píxeles, así que si al hacer el difuminado modificamos el píxel que no corresponde con un 8 y pasa a hacerlo, puede ser que se clasifique como un 8 erróneamente. Es por esta razón por la que se observa que estos son los números que más se confunden con el target 8.

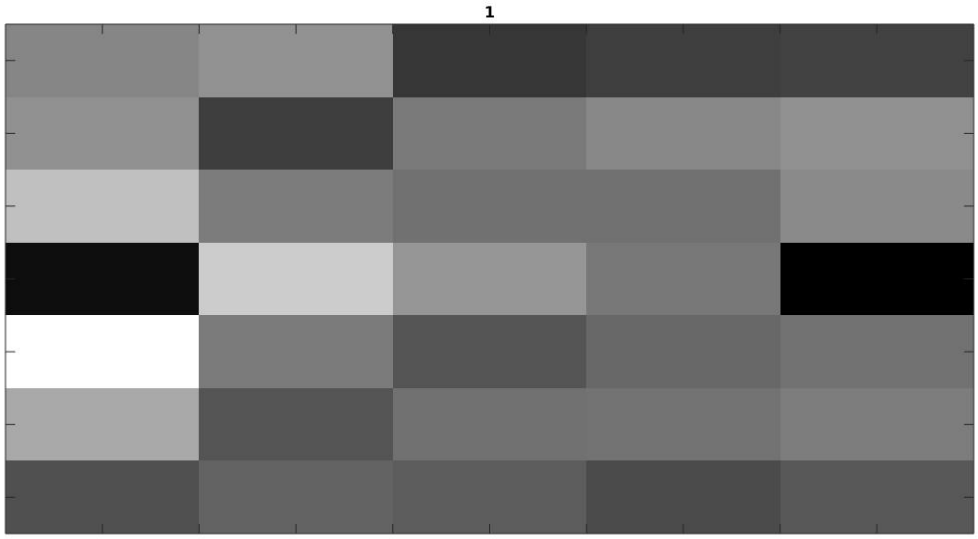



Figure 2: Pesos para num_target = 8

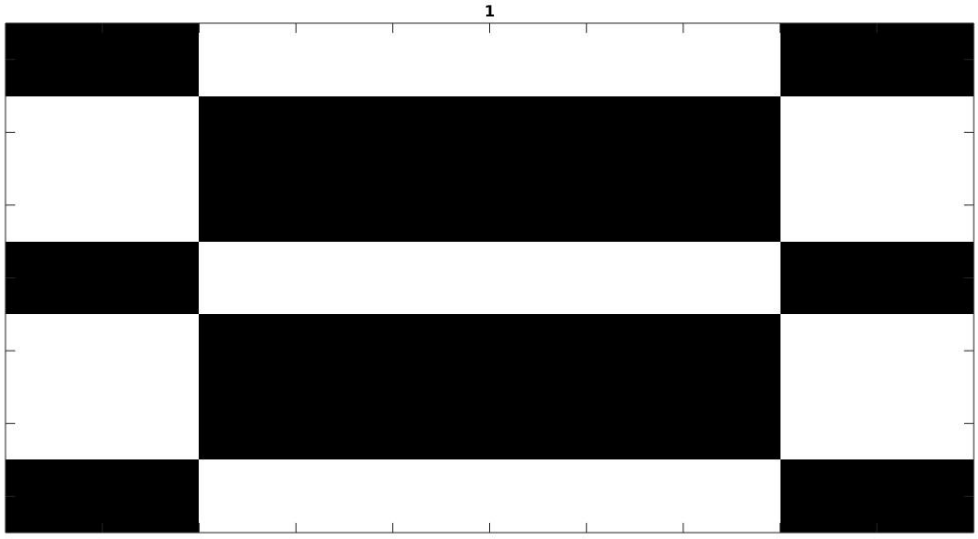



Figure 3: Dígito a identificar (8)

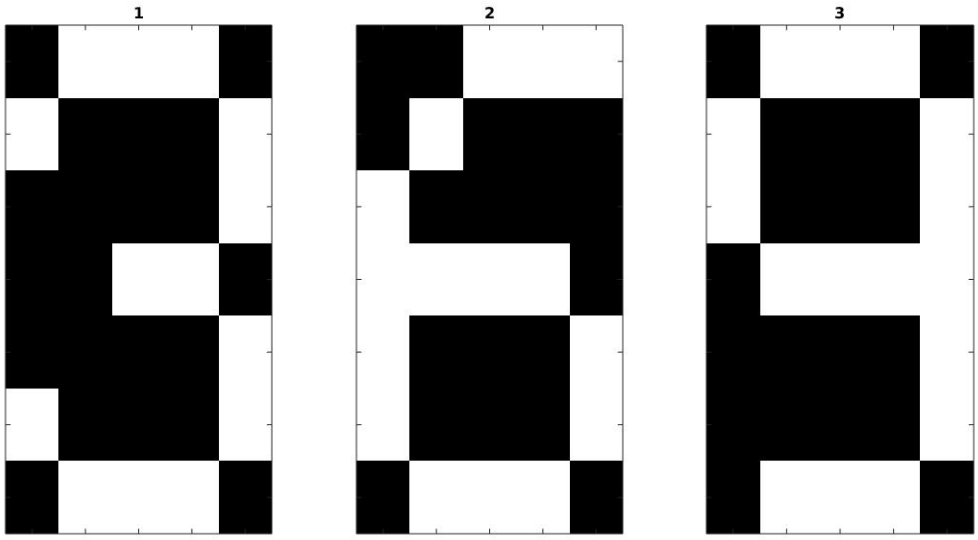



Figure 4: Dígitos que más se confunden con el 8