UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERIA Y AGRIMENSURA

(F.C.E.I.A.)

TÉCNICAS DE INTELIGENCIA ARTIFICIAL

APRENDIZAJE AUTOMATIZADO

ALUMNOS:

CARRAZZONI, Renzo C-6092/5
RAMIREZ, Fernando R-3888/1

AÑO: 2021

1. Índice

2. Introducción

El Aprendizaje Automatizado es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan . En otras palabras, se trata de desarrollar algoritmos para convertir observaciones o datos en programas de computadora, sin tener que escribirlos explícitamente. Existen diferentes tipos de AA como aprendizaje supervisado, donde se cuenta con muchísimas observaciones (con sus atributos) etiquetadas del proceso a automatizar, o no supervisado, donde se aprende directamente de la iteración del proceso. En este Trabajo Práctico nos propusimos desarrollar una IA para resolver el problema propuesto. Expondremos a continuación los pasos y las elecciones tomadas para alcanzar la máquina más eficiente. Se trata de un problema de clasificación, partimos de un conjunto de datos para entrenar distintos modelos, es decir, aprendizaje supervisado, para finalizar clasificando un conjunto de observaciones sin etiquetas, es decir, desconociendo la respuesta real o perfecta. Los algoritmos implementados son los Árboles de Decisión y Redes Neuronales Artificiales.

El primero es un método sencillo, pero con gran potencial, de entrenamiento rápido y con diferentes adaptaciones posibles. Tiene la limitación de que las decisiones se toman definiendo un umbral constante, con lo cual se dividen los atributos con líneas rectas “paralelas” a los ejes de atributos si pensamos en gráficas en ejes cartesianos. Dentro de los seteos que se pueden realizar tenemos la posibilidad de poner limitaciones al crecimiento de los árboles, ya sea limitando la cantidad de hojas (nodos finales con la clasificación del dato) o limitando la cantidad de nodos padres. Otra forma de acortar la extensión de los árboles es podarlos, se trata de sacarle niveles, ya sea una cantidad definida por ensayos o una cantidad sugerida por un algoritmo. Además, se puede elegir el criterio de optimización entre una serie de posibilidades que, como veremos, generan resultados diferenciables.

Por otro lado, las Redes Neuronales Artificiales son, como su nombre lo indica, algoritmos que buscan emular el cerebro, o mejor dicho la forma de memorizar, asociar hechos y adquirir conocimiento. Se trata de establecer un peso a cada entrada (dato) por el cual se la multiplica, luego sumarla a un umbral y finalmente ser evaluada por una función transferencia elegida. Esto en cada neurona. La función elegida es la que establece la clasificación final y es el mayor fuerte de este método. Entre estas se encuentra la función sigmoidea, que combinada con la función lineal se puede aproximar casi cualquier función. La combinación entre funciones es posible porque se pueden agregar diferentes capas, llamadas capas ocultas , que sumadas a la variabilidad de la cantidad de neuronas por capa, dan lugar a una mayor versatilidad para clasificar comparando con los ADs.

Utilizaremos como herramienta de software Matlab dado que cuenta con las librerías necesarias además de ser muy versátil para realizar diferentes pruebas con diferentes seteos de las IAs.

3. Problema Planteado

Se cuenta con el dataset “letras.mat” que contiene dos matrices: “datosLetras” y “targetsLetras” con 20.000 observaciones de 16 atributos. Estos son números ordenados en filas, siendo las columnas los atributos, y las etiquetas son letras ordenadas en filas. La cantidad de clases es 26. Aplicando las herramientas de Árboles de Decisión y Redes Neuronales vistas al dataset, navegar entre distintas configuraciones y variables hasta obtener modelos convenientes y errores mínimos. Documentar el proceso y evaluar un conjunto de datos suministrado por la cátedra con las redes y los árboles obtenidos.
Finalmente obtener conclusiones en base a los criterios vistos en clase, comparando distintos aspectos de los modelos obtenidos y realizar un análisis de los resultados para cada caso.

4. Desarrollo

Empezaremos por exponer el desarrollo del trabajo con Árboles de Decisión, para luego explayarnos en las Redes Neuronales Artificiales.

4.1 Arboles de Decisión

4.1.1 Preparación de datos

Los datos de entrenamiento ya se encuentran ordenados y listos para utilizar. Mientras que las etiquetas debemos pasarlas a números para poder emplear las herramientas de Matlab. Para esto utilizamos la función “ char()” para generar un arreglo de caracteres y luego lo transformamos a números con la función “double()” haciendo uso del código ASCII. Finalmente le restamos 65 a cada elemento de este último arreglo para saltear los primeros 64 caracteres no utilizados, ya que sólo tenemos letras en mayúsculas.

4.1.2 Entrenamiento y evaluación

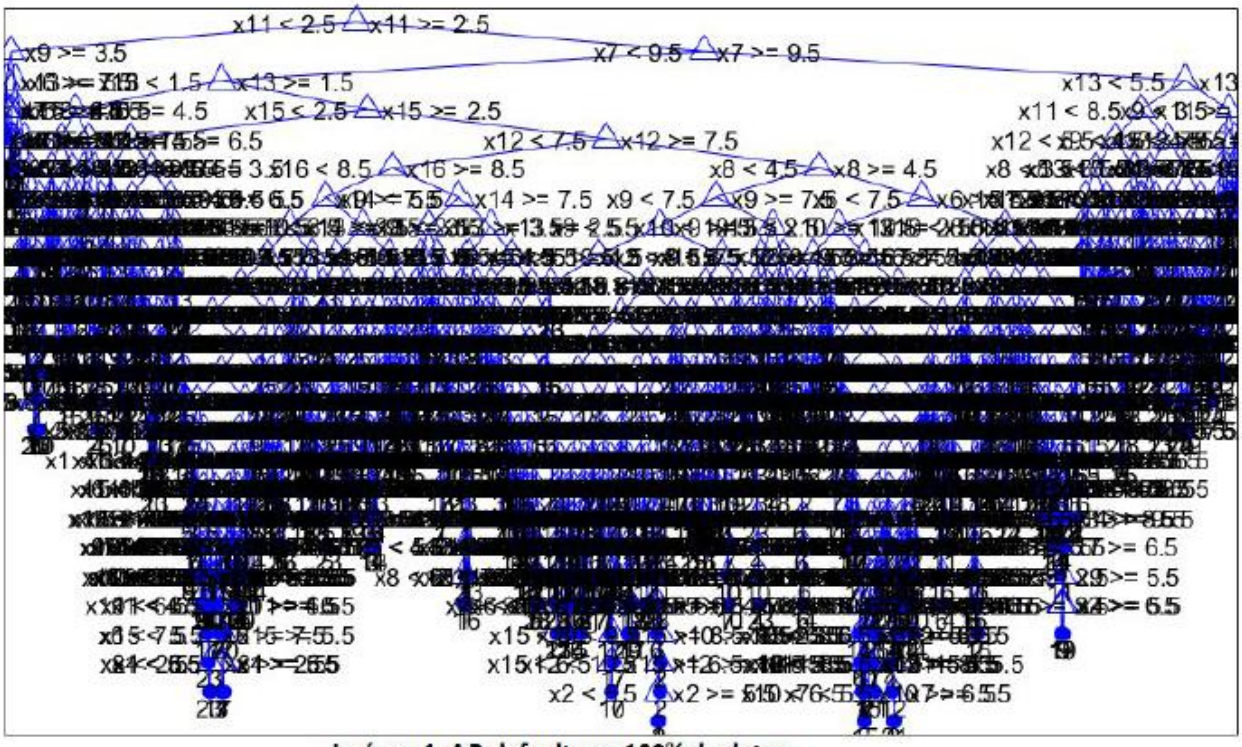
Empezaremos por mostrar un árbol y sus características, entrenado con todos los datos y con los parámetros por defecto. Buscamos con esto dimensionar la situación planteada.

- Criterio de optimización diversidad de Gini.
- MinParentSize = 10
- MinLeafSize = 1
- MaxNumSplits = size(datos,1) - 1

En la imagen 1 vemos el árbol que tiene las siguientes características.

- Nodos totales = 2415
- Nodos padres = 1207
- Nodos hojas = 1208
- Error de resubstitución = 0,0559

Es evidente observar que obtenemos un árbol muy grande y sobre entrenado. El error de resubstitución que se trata de evaluar los mismos datos de entrenamiento no da cuenta de ello por lo bajo que es.



Imágen 1. AD default con 100% de datos

Tomamos como primera medida que se hizo evidente, entrenar árboles con la herramienta "validación cruzada" . Se trata de particionar el conjunto de datos para generar diferentes subconjuntos de entrenamiento y de testeo. Teniendo 20.000 datos probamos con particionar en kfold = 4, 8 y 10, de manera que quedaran conjuntos de: 15.000/5.000, 17.500/2.500 y 18.000/2.000.

Utilizando como criterio el default, obtuvimos los siguientes errores por validación cruzada y en el testeo de los mejores árboles de cada partición:

Diversidad de Gini		
Partición	Error de Resubtitución	Error de Testeo
4	0,0625	0,1498
8	0,061	0,1472
10	0,0597	0,1425

Se puede observar que particionar en 10 tiene mejor error de resubstitución, lo cual es lógico por tener más datos de entrenamiento, pero también tiene mejor error de testeo, lo cual nos hizo inclinar por esta opción.

Como mencionamos, el último ensayo fue realizado con el criterio diversidad de Gini, por lo cual nos proponemos ahora entrenar árboles con las mismas particiones, a pesar de obtener mejor resultado con kfold=10, para los tres criterios distintos: diversidad de Gini (ya realizado), entropía y Twoing. De esta manera buscamos descartar al menos un criterio al ser menos eficiente.

Twoing		
Partición	Error de Resubstitución	Error de Testeo
4	0,0667	0,138
8	0,0639	0,1356
10	0,0637	0,1345

Entropía		
Partición	Error de Resubstitución	Error de Testeo
4	0,0622	0,1388
8	0,0543	0,1344
10	0,0548	0,1275

La primera conclusión que sacamos es que particionar en 10 es la mejor opción para todos los casos. En segundo lugar, se ve que la optimización por entropía parece ser el que mejor se adapta. De esta manera, a partir de ahora utilizamos este último criterio para profundizar en la eficiencia de la IA.

Como mencionamos, los árboles obtenidos están sobre entrenados, esto quiere decir que clasifica los datos con un nivel de detalle que solo aplica a los datos de entrenamiento. Para contrarrestar esto cambiamos los parámetros de entrenamiento, por un lado, y por otro podamos el árbol. Así poder contrastar con dos métodos diferentes.

Las variaciones de los parámetros son los mostrados en las tablas en el orden:‘MinParentSize’ ; ‘MinLeafSize’ ; ‘‘MaxNumSplits’.

12 - 5 - 950		
Partición	Error de Resubstitución	Error de Testeo
4	0,0955	0,1484
8	0,0907	0,142
10	0,0897	0,139

12 - 5 - 1000		
Partición	Error de Resubtitución	Error de Testeo
4	0,0961	0,1536
8	0,0894	0,1388
10	0,0883	0,1335

8 - 3 - 1000		
Partición	Error de Resubstitución	Error de Testeo
4	0,0719	0,1514
8	0,0722	0,1388
10	0,0713	0,121

5 - 1 - 1000		
Partición	Error de Resubstitución	Error de Testeo
4	0,0605	0,1448
8	0,0681	0,1396
10	0,0679	0,1365

En las últimas dos tablas se puede ver una mejora muy considerable de los errores de testeo comparado con el árbol que tenía como límite los parámetros por defecto.

Ahora analizaremos la poda, es decir, dejar el crecimiento con parámetros por defecto y aplicar una poda en el nivel sugerido por el algoritmo de la función " cvLoss()" y valores arbitrarios que decidimos probar.

La función mencionada arrojó un valor de poda óptimo de 3 niveles. Y el e_resubstitución con dicha poda fue de 0,0585 para un árbol con un Kfold=10.

No obstante, hicimos unas tablas para evaluar los errores de substitución distintos niveles de poda con distintas particiones de Kfold.

Poda de 3 niveles	
Partición	Error de Resubstitución
4	0,0641
8	0,0594
10	0,0585

Poda de 4 niveles	
Partición	Error de Resubstitución
4	0,0669
8	0,0593
10	0,0603

Poda de 5 niveles	
Partición	Error de Resubstitución
4	0,0677
8	0,0601
10	0,0611

Siendo el menor error de 0,0585, la mayoría son mucho mejores que los errores que habíamos obtenido con la limitación de crecimiento. Queda muy claro que, al menos para este problema, esta es la mejor opción.

A raíz de todo lo desarrollado es que optamos por quedarnos con el árbol de decisión entrenado con los parámetros y las características mostradas a continuación:

- Entrenado con validación cruzada de 10 particiones.
- Parámetros por defecto: ‘MinParentSize’ = 10 ‘MinLeafSize’ = 1 ‘MaxNumSplits’= size(datos,1)-1.
- Nivel de poda igual a 3

4.1.3 Predicción

Para finalizar con la sección de Árboles de Decisión, utilizamos el algoritmo seleccionado para predecir la frase dada como enunciado. La predicción arrojó como resultado:

'KLMKJORKXPKRPOPAMHPKNFUKUNDPAAPRKNDPZ'

No hace falta ser el mejor aprendiz, para darse cuenta de que la predicción es errónea. Se puede vislumbrar como el árbol confunde la E con la K, y se puede ver que la P la confunde tanto con la T como con la I sin embargo la reconoce bien cuando es una P, además, la B la confunde con la H.

Probamos con niveles de poda mayor, sin embargo, notamos que los resultados empeoraban haciéndose mas ilegibles paso a paso. Podemos atribuir esto a que tenemos una cantidad de datos muy grandes para ser sólo 16 atributos, quedando evidenciado un sobre entrenamiento sostenido. Con lo cual concluimos que el método no es el apropiado y que presenta serias limitaciones para reconocer estos tipos de datos dado que clasificar con umbrales constantes no es lo mas apropiado para este caso.

4.2 Redes Neuronales Artificiales

4.2.1 Preparación de datos

Se acondicionaron los datos convirtiendo los targets de arreglo de cells, a arreglo de doubles, donde cada campo contiene la representación ASCII del carácter correspondiente. A diferencia de los datos necesarios para entrenar los AD, necesitamos que estén traspuestos, es decir, las observaciones en columnas y los atributos en filas. Para esto utilizamos el siguiente algoritmo:

```
y = zeros(size(targetsLetras));
for i=1:1:length(targetsLetras)
y(i) = double(targetsLetras{i})-64;
end
```

4.2.2 Entrenamiento y evaluación

Hicimos uso de la función “train()” de Matlab la cual usa los datos para entrenar, testear y validar a la red. Con la validación, el propio algoritmo de entrenamiento mide los errores y decide cuándo finalizar el entrenamiento, es decir, corta con las iteraciones en las que va ajustando la matriz de pesos de las entradas de cada neurona.

Entre las opciones vistas podemos aplicar diferentes funciones transferencias: un escalón o la función lineal, que se aplican en redes de una sola capa, o lo que se llama “Backpropagation”. Por defecto, la función para crear estas redes, “feedforwardnet()” , genera una primera capa de entrada, una capa oculta con la función sigmoidea y una capa de salida con la función lineal.

En base a la sección anterior, decidimos empezar directamente por la función lineal, dado que el problema planteado no es sencillo y la clasificación por umbrales constantes no dio para nada buen resultado.

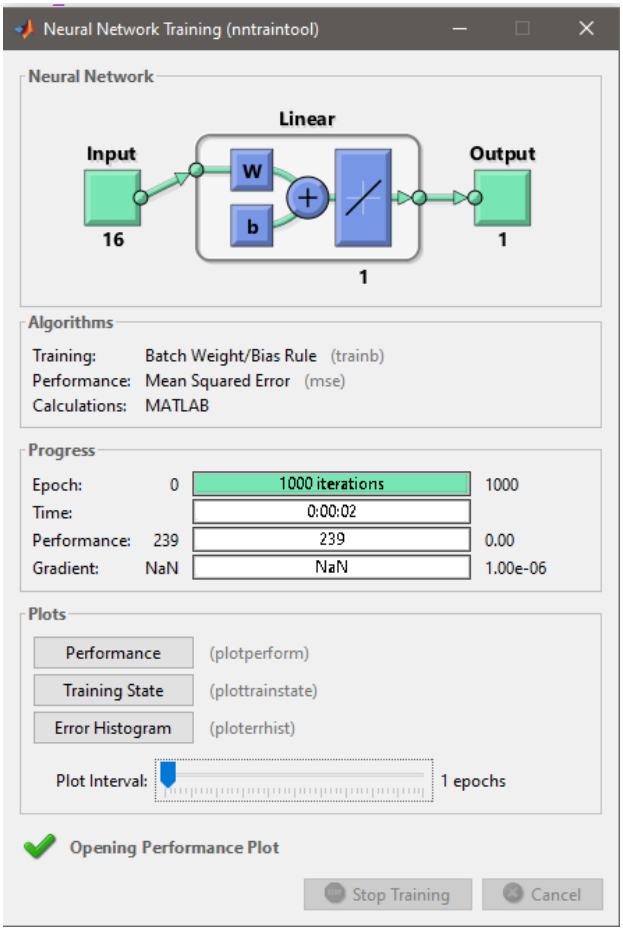


Fig.2 - Ventana de training de Matlab

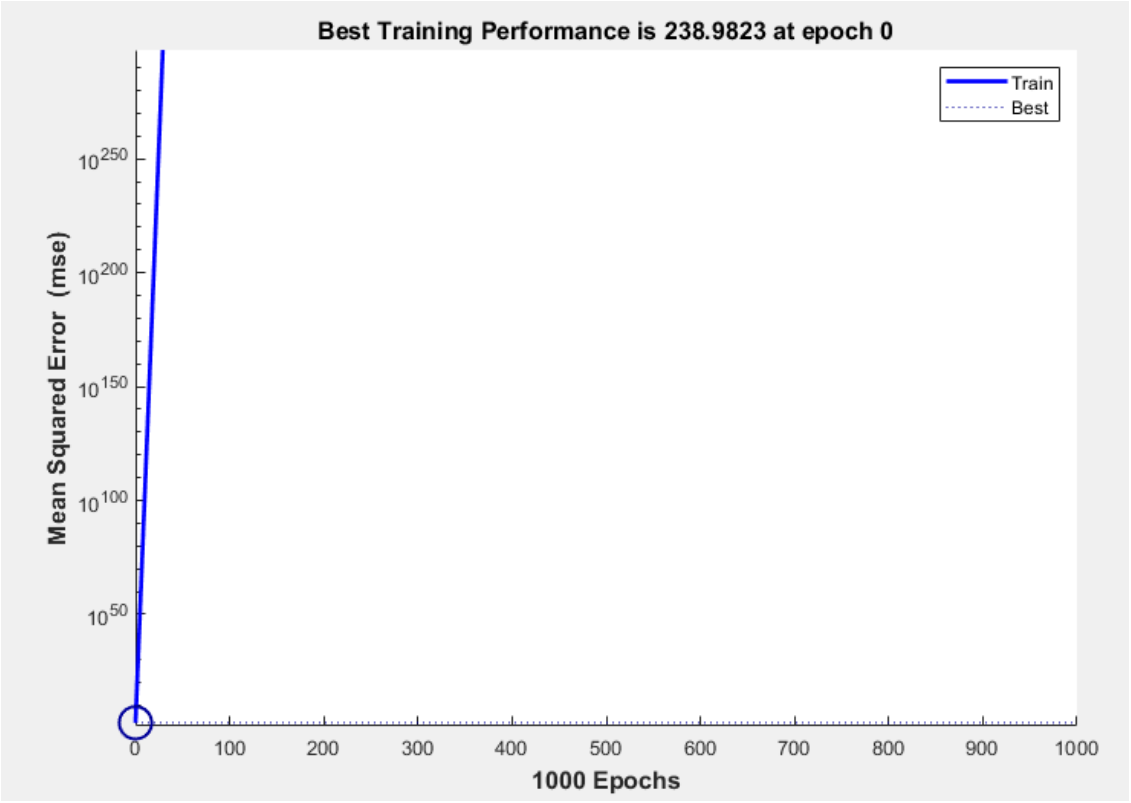


Fig.3 - Performance Plot de la red neuronal

El resultado entregado fue:

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'

Nos encontramos inmediatamente con una particularidad que nos anticipó la mala performance de esta opción: la red se entrenó con el máximo de iteraciones posibles (mil iteraciones). Dado la simplicidad de la red, el proceso de entrenamiento fue rápido, pero claramente no satisfactorio. Ambos resultados muestran la no posibilidad de utilizar esta herramienta, incluso con resultados muy diferentes a los de los Árboles de Decisión.

Pasaremos entonces a utilizar Backpropagation . Iniciaremos con una cantidad de diez neuronas en una capa oculta. En la imagen posterior vemos la performance de dicha red y ya visualizamos su mejora al menos en el entrenamiento. También se observa que no hay diferencia entre las diferentes etapas de entrenamiento, lo cual, como veremos en la predicción de la frase, recae en una mala eficiencia para resolver el problema. Aun así, queremos destacar que, a diferencia de la función lineal, sólo utilizó 87 iteraciones para llegar a la red final, lo cual es muy positivo porque denota que al menos llegó a un resultado y no se le terminaron las iteraciones posibles.

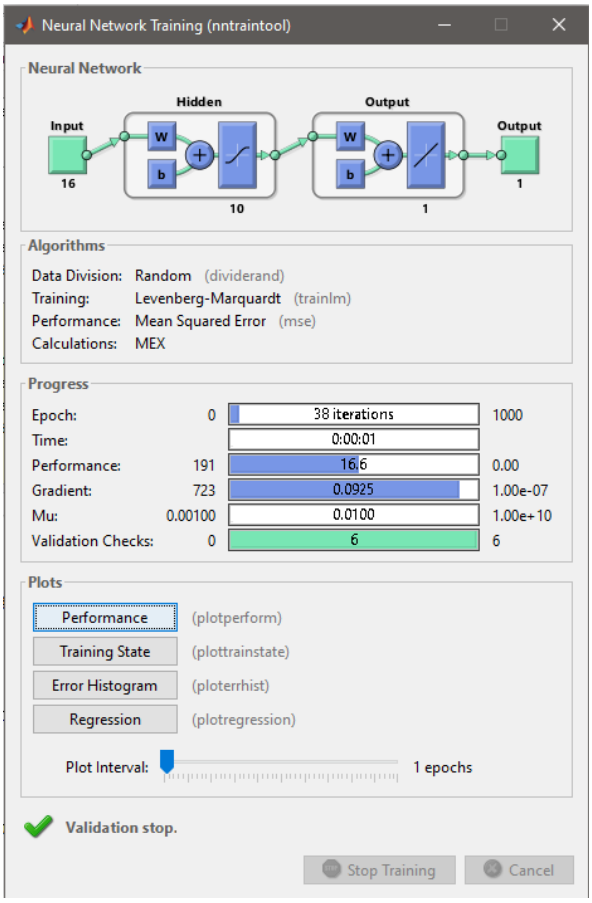


Fig.4 - Ventana de training de Matlab

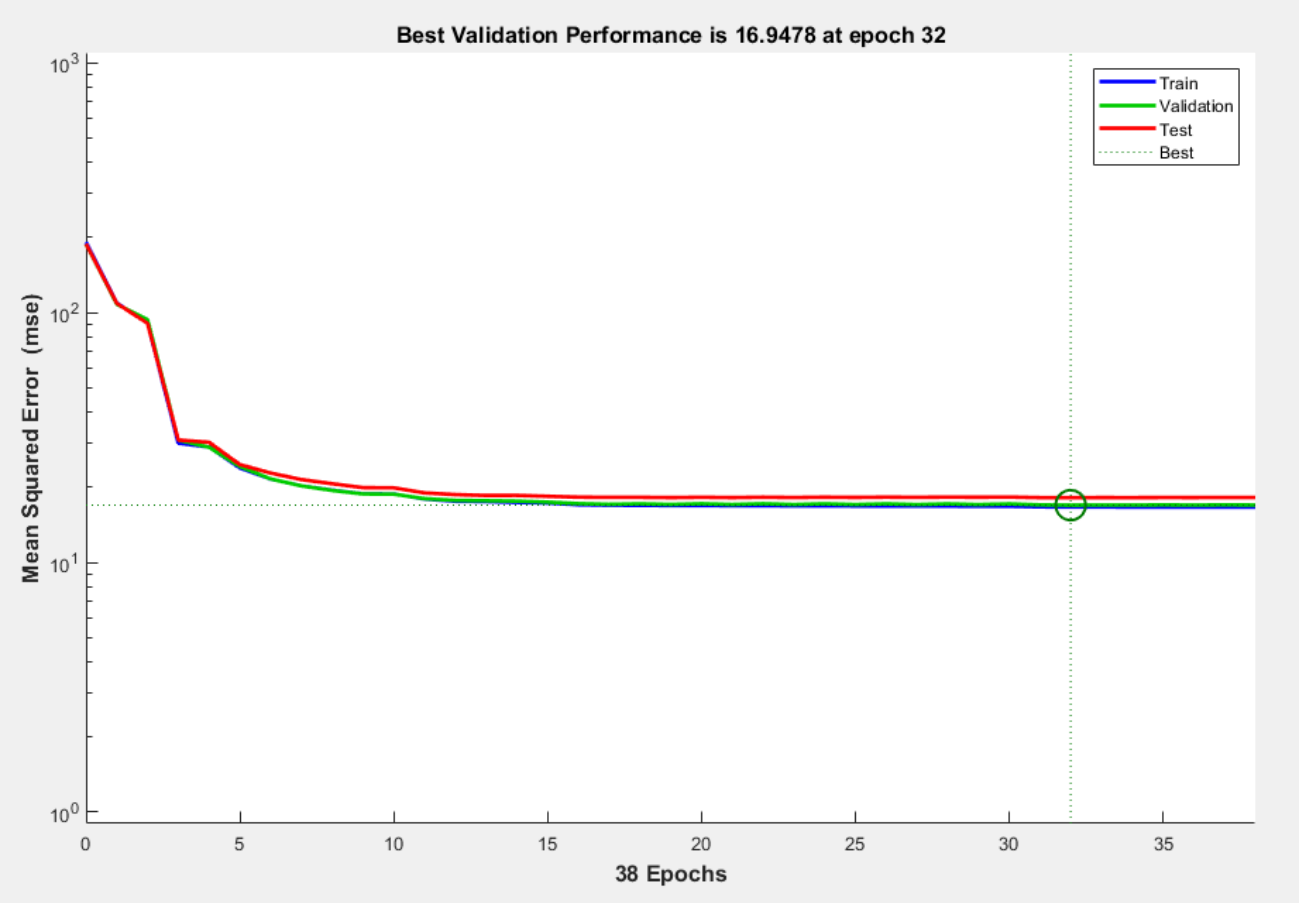


Fig.5 - Performance Plot de la red neuronal

La predicción de la frase fue:

'KJMKNPOKSLKOMPMBMIIKOGUKUOFIBBLOKOFIY'

Si bien no podemos identificar una razón clara para este problema, decidimos probar con una mayor cantidad de neuronas. Si bien las iteraciones aumentaron en cantidad y empezamos a ver una notable diferencia en la performance, los resultados fueron pésimos.

30 neuronas: 'JGNJNNMJQLJMUNUCNIIJMITJTMDICCLMJMDIY'

50 neuronas: "'LKQLJPRLURLRRPR@QGKLQHULUQFK@@RRLQFK_'"

100 neuronas: "LLSLPOQLUQLQQOQCSFMLNISLSNHMCCQQLNHMX"

Se hace evidente que un incremento en el número total de neuronas no da un buen resultado, debido a que se está pretendiendo que la red sea lo suficientemente precisa como para que presente a su salida un número entre 0 y 26 que, al ser dicha salida de tipo double, debe estar acotado a un error absoluto de +-0.5 para evitar errores por redondeo.

Una aproximación más eficiente es pensar en una neurona por salida válida, es decir, para este caso 26 neuronas de salida cada una presentando un valor de “1” si la entrada corresponde a la letra que se le atribuye a dicha salida o “0” si no.

Para conseguir esto, se cargaron nuevamente los datos y se acondicionaron de forma que los targets están almacenados en una matriz cuyas filas representan a las 26 letras (una por cada atributo) y cada columna está compuesta de ceros, excepto en la fila correspondiente a la letra en cuestión.

Utilizamos el siguiente código:

```
y = full(ind2vec(double(cell2mat(targetsLetras())-64)'));
```

Creamos una red de tipo "Patternet" , capaz de reconocer el patrón dado, utilizando 10 neuronas y obtuvimos la performance que se muestra a continuación.

Performance = 0,0276, número que adelanta una mejor solución a las vistas hasta aquí.

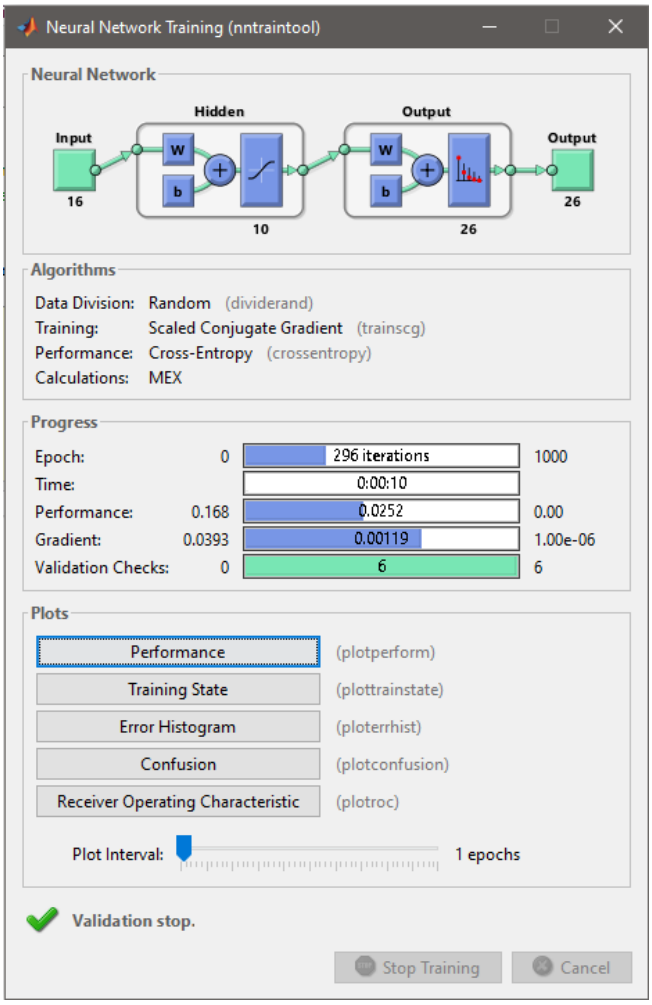


Fig. 6 - Ventana de training de Matlab

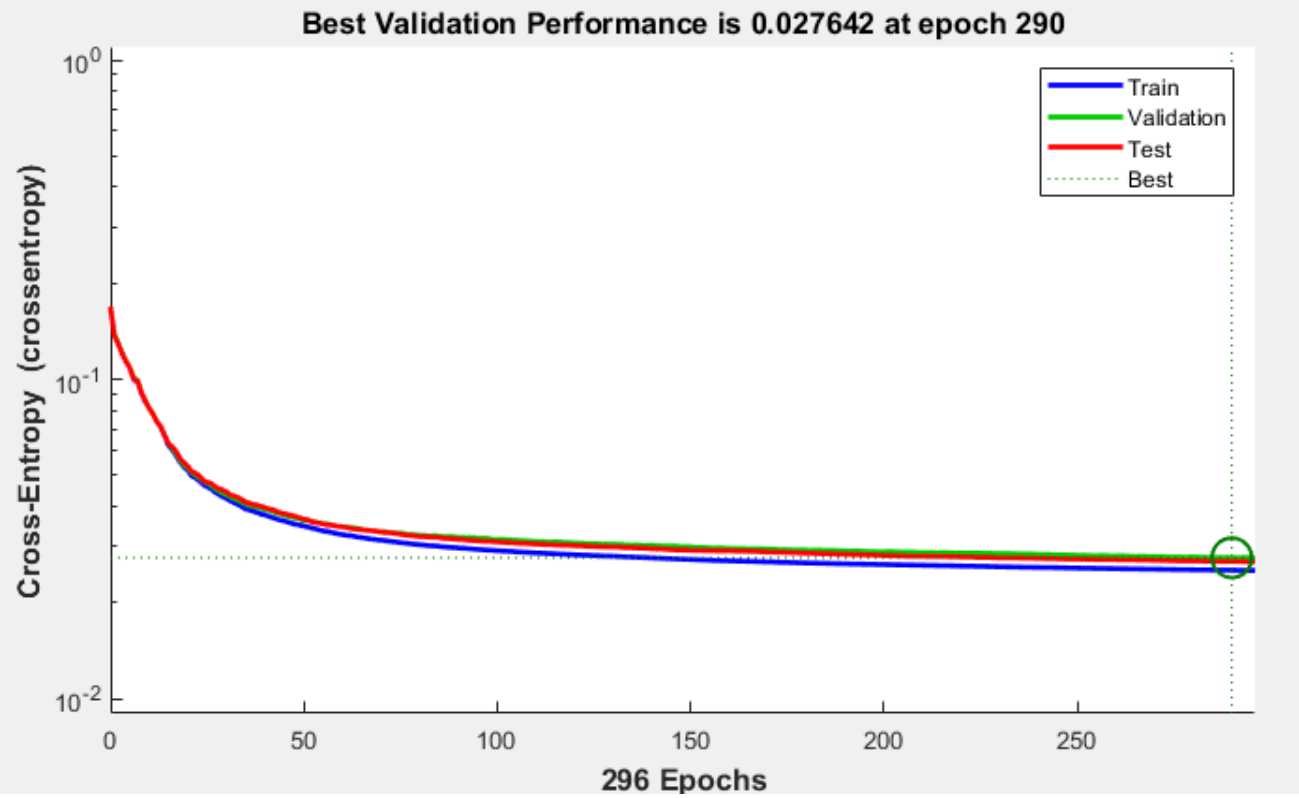


Fig. 7 - Performance plot de la red neuronal

Luego realizamos la predicción de la frase y reacondicionamos la respuesta con el código posterior.

```
Predic_PR = net_pr(frase')
frase = char(vec2ind(Predic_PR)+64)
```

Encontramos un resultado con 100% de efectividad, es decir, se predijo perfectamente la frase.

'ELMEJOREXPERTOTAMBIENFUEUNDIAAPRENDIZ'

Concluimos que el buen resultado se debe a la gran capacidad de esta herramienta y que lo logramos con un análisis puntual del problema a resolver, en base a los resultados de las primeras redes. Puntualmente, el buen resultado se debe al reconocimiento de patrones, que sin una cantidad de neuronas excesiva, fue suficiente para decodificar la frase dada. No podemos dejar de mencionar que el acondicionamiento de los datos es fundamental para obtener un buen resultado. Además vale aclarar que intentamos utilizar distintas cantidades de capas, con distintas cantidades de neuronas en redes de Backpropagation, pero el tiempo de entrenamiento se volvía excesivo para resultados que eran mejores.

5. Conclusiones

En primer lugar, queremos resaltar que la realización del Trabajo Práctico fue fundamental para afianzar los conocimientos propuestos. El desarrollo del mismo en búsqueda de una solución satisfactoria nos proveyó de varias herramientas, pero más que nada de experiencia para poder elaborar ideas concretas que apunten a la buena resolución de problemas.

En términos de las herramientas utilizadas, el software Matlab resultó de suma utilidad por su practicidad y gran librería de funciones. Los algoritmos de Aprendizaje Automatizado fueron, en definitiva, satisfactorios para resolver el problema. Si bien los ADs no dieron solución por completo a la consigna, sí fue de gran utilidad para adentrarnos en la problemática planteada. Además, el trabajo práctico nos permitió visibilizar las limitaciones, pero también gran potencial por ser sencillo y de corto tiempo de entrenamiento.

Por su parte, las RNA marcaron una gran superioridad, no sólo por lograr resolver los planteado, sino también por la gran versatilidad que tienen existiendo una amplia variedad de funciones de transferencia para aplicar, sumado a la posibilidad de variar la cantidad de neuronas por capa oculta y la cantidad de capas ocultas.