Universidad Nacional de Rosario

Introducción al Aprendizaje Automatizado

Trabajo Práctico II

Villagra Martín

30 de mayo de 2017

Introducción

Este trabajo se centra sobre redes neuronales feedforward de 3 capas, utilizando el algoritmo de backpropagation estocástico para el entrenamiento.

Ejercicio a

En primer lugar se confeccionó la gráfica de la Figura 1. Para hacerlo se tomo la mediana del error en test de 20 redes distintas por cada posible configuración de los parámetros. En la gráfica se pueden observar zonas con errores altos y una gran cantidad de mínimos locales concentrados. También se notó que un mínimo local puede funcionar bien para el caso promedio pero para muchas semillas no es óptimo.

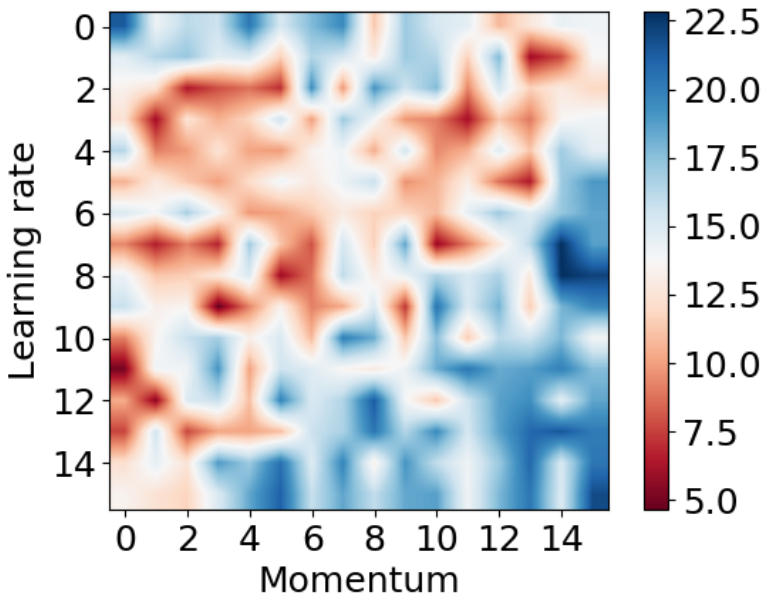


Figura 1: Error porcentual discreto en test en función del momentum y learning Rate.

Para entrenar la mejor red posible, se probaron varios valores esta vez buscando una semilla buena (sin promediar). Una vez que se encontró una semilla que produce una red con error bajo, se asumió que fijada la semilla (es decir sin promediar) la función de la Figura 1 no iba a ser tan compleja. Por lo cual se utilizó el algoritmo de optimización Nelder-Mead para encontrar el par momentum-learning rate que minimicen el error. Se logró obtener una red con error del 2.65 % en test utilizando

los parámetros: SEED=625922, μ=0.45, η=0.0703, su entrenamiento se puede ver en la Figura 2. Como plus, la red converge al error mínimo en apenas 3800 épocas. Si bien esta técnica es altamente efectiva, su utilidad práctica es reducida pues se está haciendo, de forma manual, un sobreajuste de los hiper-parámetros y peor aún, usando el conjunto de test.

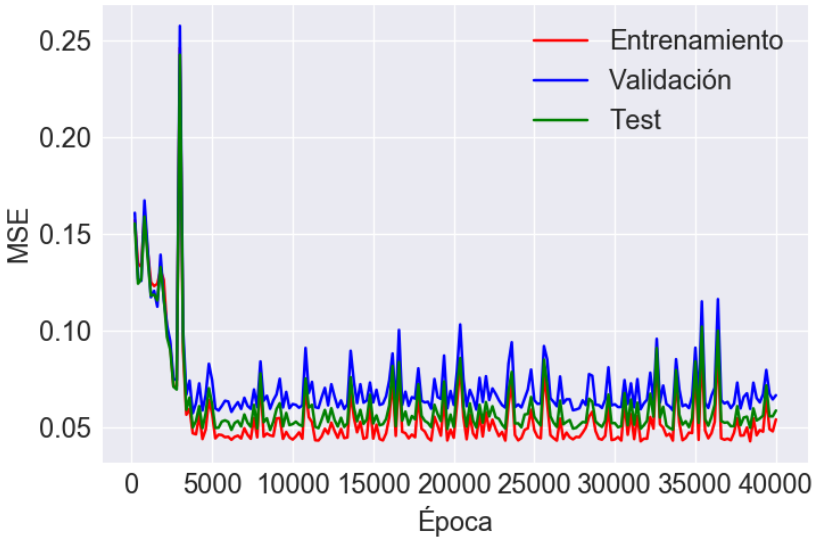
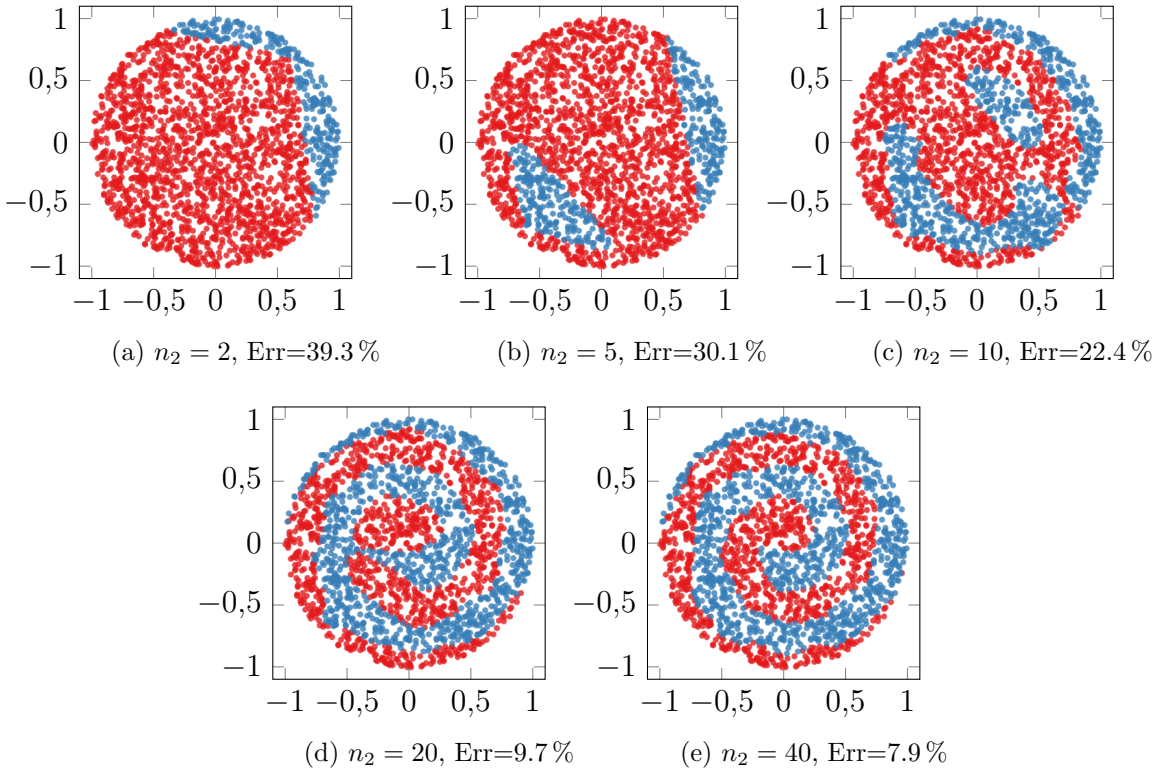


Figura 2: Función de error vs cantidad de épocas en el entrenamiento de la red para el dataset dos_elipses.

Ejercicio b

Lo primero que se observa en la Figura 3 es la mejora en las predicciones a medida que aumenta la cantidad de neuronas. En particular, cuando $n_2 = 40$, la red parece haber aprendido la "forma" de los espirales. Cuando la cantidad de neuronas es baja, los límites entre clases son rectas. En esto se evidencia la naturaleza casi lineal de las neuronas. Sin embargo, al aumentar la cantidad de neuronas la red tiene la capacidad de representar funciones no lineales más complejas, como la curva de los espirales.



(a) $n_2 = 2$, Err=39.3 % (b) $n_2 = 5$, Err=30.1 % (c) $n_2 = 10$, Err=22.4 %

(d) $n_2 = 20$, Err=9.7 % (e) $n_2 = 40$, Err=7.9 %

Figura 3: Predicciones sobre el conjunto de test variando la cantidad de neuronas ocultas.

Ejercicio c

Como resultado de las pruebas obtuvimos que cuanto menor es el conjunto de validación, menor es el error en test. De hecho, cuando se elimina completamente el error en test se obtienen redes con menor error. Esto puede deberse al bajo tamaño del conjunto de entrenamiento ($n = 200$). En este caso es más conveniente tener más datos de entrenamiento, aunque se haga sobreajuste. En la Figura 4 se muestra el entrenamiento de la red sin utilizar conjunto de validación obteniendo un error en test del 9.7 %.

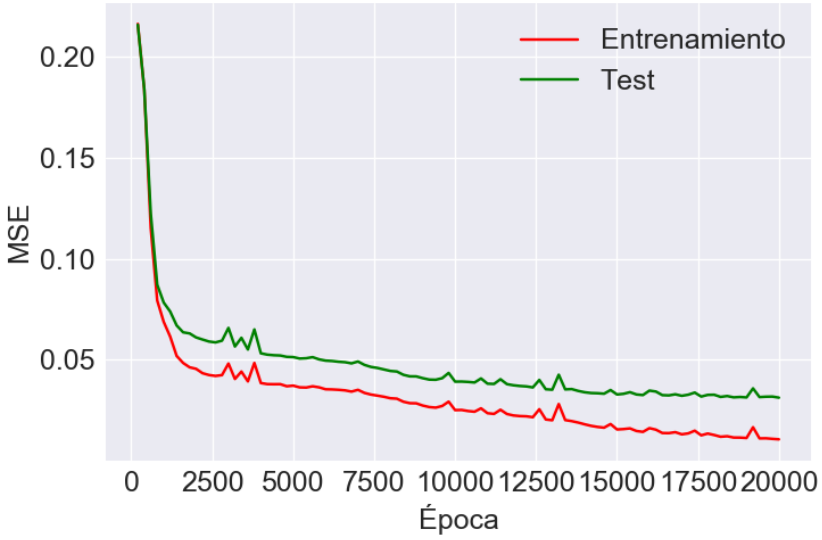
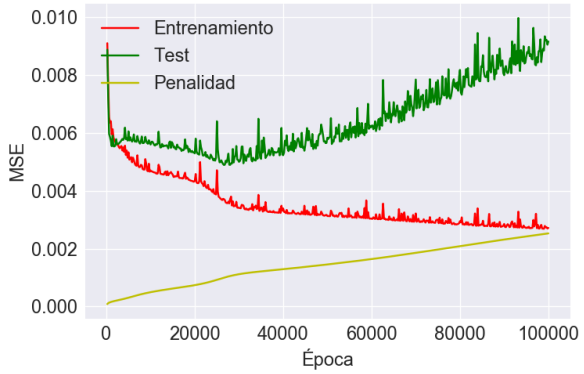


Figura 4: Error porcentual vs. cantidad de épocas de la red sobre el dataset ikeda.

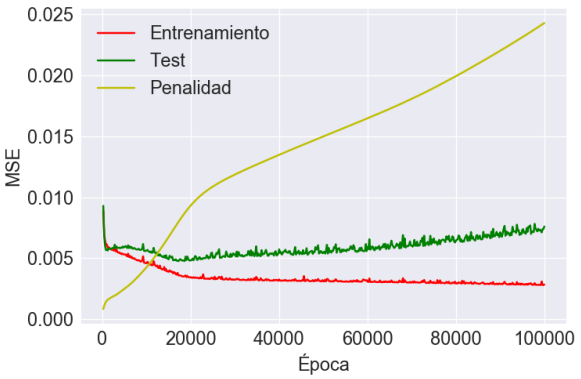
Ejercicio d

Se implementó weight-decay agregando γ como parámetro de entrada, modificando el calculo de la función de error y las correspondientes actualizaciones en sus derivadas (en la backpropagation ahora los pesos se multiplican por el término $1 - 2\gamma\eta$).

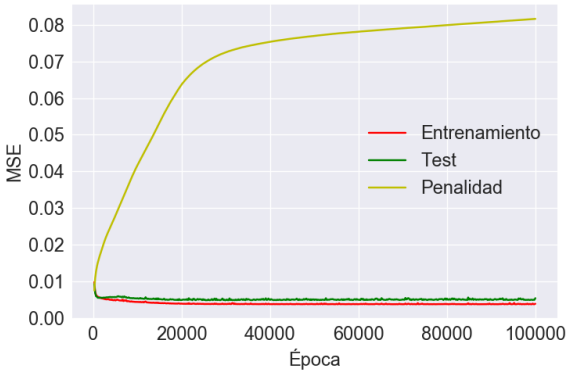
En la Figura 5 se resumen las diferentes pruebas realizadas. Cuando γ es muy bajo como en la Figura 5(a) la red realiza sobreajuste dando una error muy grande en test. Si γ es muy grande en la Figura 5(c) vemos que la penalidad es excesivamente grande y la red no entrena, esto es porque la penalidad impide que los pesos crezcan lo suficiente como para hacer algo útil. Encontramos valores aceptables con γ entre 10^{-6} y 10^{-5}, como se puede ver en la Figura 5(b). Con este γ la penalidad está en el mismo orden de magnitud que el MSE, aunque aún se aprecia sobreajuste.



(a) $\gamma = 10^{-7}$



(b) $\gamma = 10^{-6}$



(c) $\gamma = 10^{-1}$

Figura 5: MSE y penalidad vs cantidad de épocas utilizando diferentes γ para el dataset Sunspots.

También se observa como la penalidad aumenta a medida que pasan las épocas, esto es porque los pesos de las neuronas van en aumento indicando que se vuelven cada vez más sensibles a ciertos patrones de interés.

En conclusión el weight-decay evita la especialización excesiva que caracteriza al sobreajuste pero agrega un hiper-parámetro más que hay que fijar correctamente para que la red funcione.

Ejercicio e

Siguiendo el enunciado se obtuvo la Figura 6. En primera instancia vemos que para el conjunto diagonal la red neuronal tiene un error considerablemente menor que los árboles de decisión, y que sucede lo contrario con el paralelo.

El desempeño de la red neuronal es aproximadamente igual tanto para diagonal como para el paralelo. Esto indica que para la red ambos conjuntos son equivalentes. Los árboles de decisión sólo podían representar rectas paralelas a los ejes, pero vemos que esta restricción no existe en las redes neuronales por lo que una rotación del conjunto efectivamente no cambia mucho las predicciones.

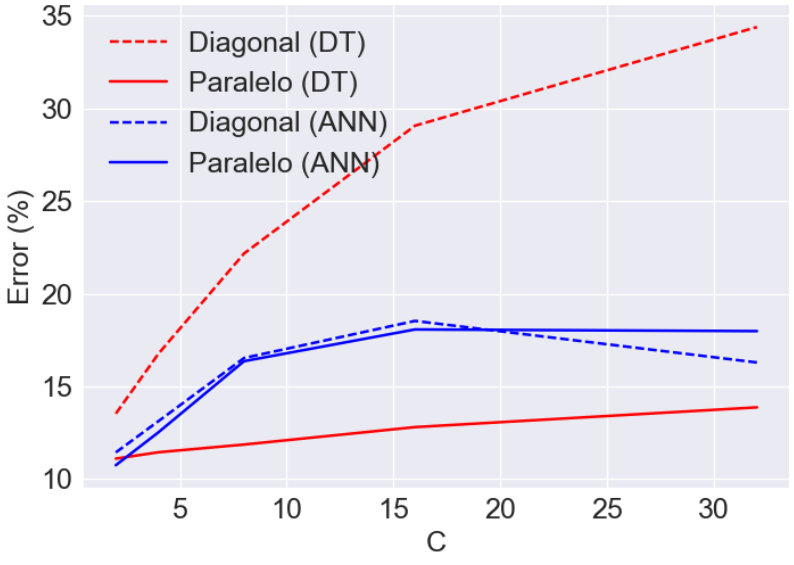


Figura 6: Error en conjunto de test vs. dimensión del input.

También se puede apreciar que la línea de las redes neuronales tienen pendientes menores que los árboles de decisión. Esto podría indicar una mayor resistencia al ruido, para el caso del paralelo.

Ejercicio f

Para admitir múltiples clases se eligió crear C neuronas de salida, donde C es la cantidad de clases. Luego cada clase i es representada por el vector canónico i (por ejemplo la clase 0 está representada por el vector (1 0 0 0)). La predicción de la red entonces está dada por la posición que tenga el valor más grande. Si bien podríamos haber usado una sola neurona de salida asignando valores equidistantes en el rango de [0, 1] para cada clase, este método es peor porque hay menos pesos para calibrar la salida. Utilizando la representación ya dicha contamos con una neurona especializada en reconocer cada clase. Otra ventaja de nuestro enconding es que los valores del vector de salida pueden usarse como un vector de probabilidad, lo que posibilita tener cierta medida de confiabilidad en la predicción.

Como desventaja tanto nuestra red como los samples crecen proporcionalmente en la cantidad de clases, lo que puede aumentar considerablemente el tiempo de entrenamiento si la cantidad de clases es elevada.

Un detalle es que en lugar de usar 0 y 1 para los vectores de salida se utilizó 0.9 y 0.1, de forma que estos valores sean alcanzables incluso si se usa una activación sigmoide en la última capa (aunque no es nuestro caso). Para implementar esta representación se transforman los inputs a medida que son leídos (Se toma N_3 como la cantidad de clases) y se añadieron algunas líneas para la nueva discretización de la predicción. Se adjunta con este informe la versión modificada del código bajo el nombre de bpcl.c

Dataset iris

Se consiguió un error en test del 0,0 % (es factible dado que sólo hay 45 muestras), con $\mu = 0{,}1$ y $\eta = 0{,}6$, 200 iteraciones y 4 neuronas ocultas. El entrenamiento se muestra en la Figura 7.

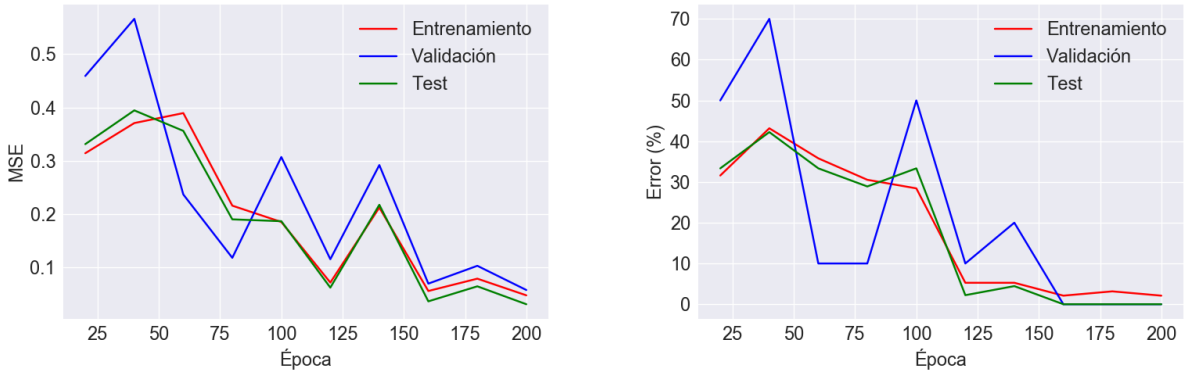


Figura 7: MSE y Error porcentual vs cantidad de épocas para el dataset iris.

Dataset faces

Se utilizó $\mu = 0{,}3$ y $\eta = 0{,}3$ con 100 iteraciones, los mismos valores usados en el libro. Dejando 10 muestras como validación se logró efectivamente un error del 10 %, igual que en el libro. La diferencia más notable radica en el poder de cómputo: según el libro el entrenamiento les tardó 5 minutos, mientras que en nuestro caso tarda menos de 1 segundo. En la Figura 8 se puede apreciar el entrenamiento de la red que llega a su mínimo en sólo 68 épocas. Otra diferencia es que en el libro se uso un descenso por gradiente completo, mientras que nosotros usamos un entrenamiento estocástico.

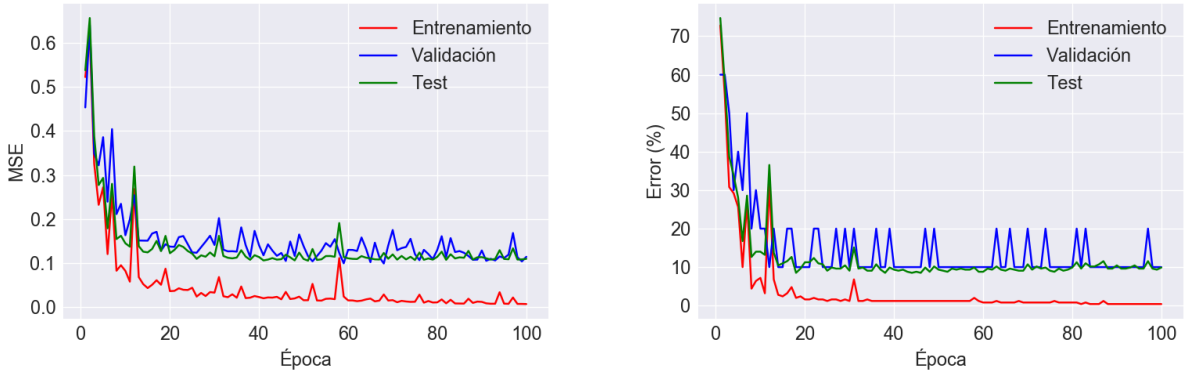


Figura 8: MSE y Error porcentual vs cantidad de épocas para el dataset faces.