Aprendizaje Profundo y Redes Neuronales Artificiales
Informe de la Práctica 2

Denise S. Cammarota

27 de septiembre de 2020

Ejercicio 1

Ejercicio 1a

En la Figura 1 se presenta la resolución del ítem a del ejercicio 1.

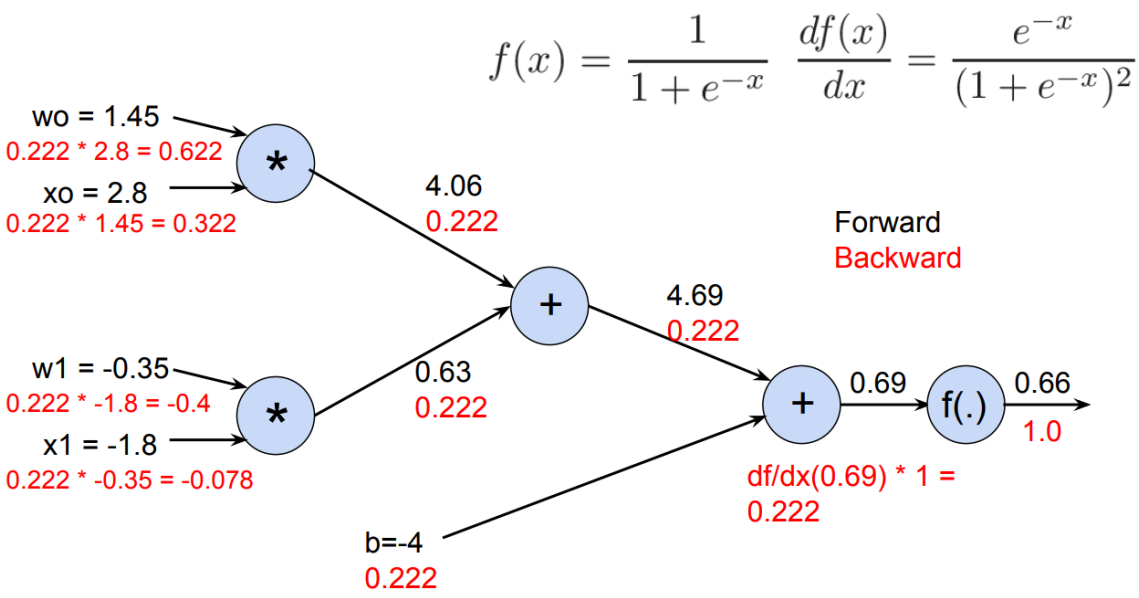


Figura 1: Resolución del ejercicio 1a.

Ejercicio 1b

En la Figura 2 se presenta la resolución del ítem b del ejercicio 1.

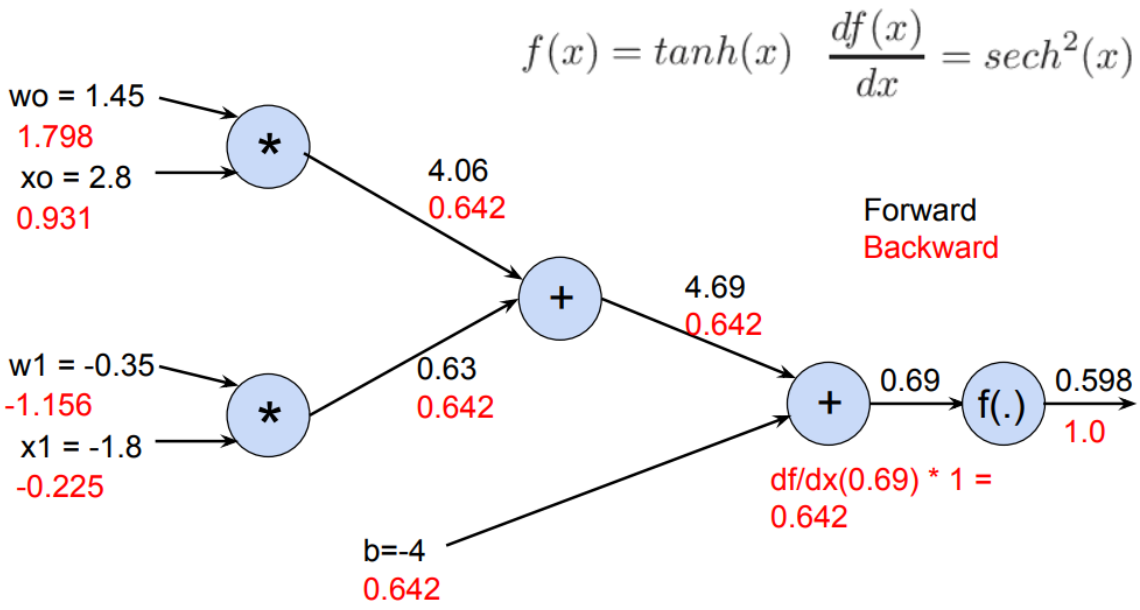


Figura 2: Resolución del ejercicio 1b.

Ejercicio 1c

En la Figura 3 se presenta la resolución del ítem c del ejercicio 1.

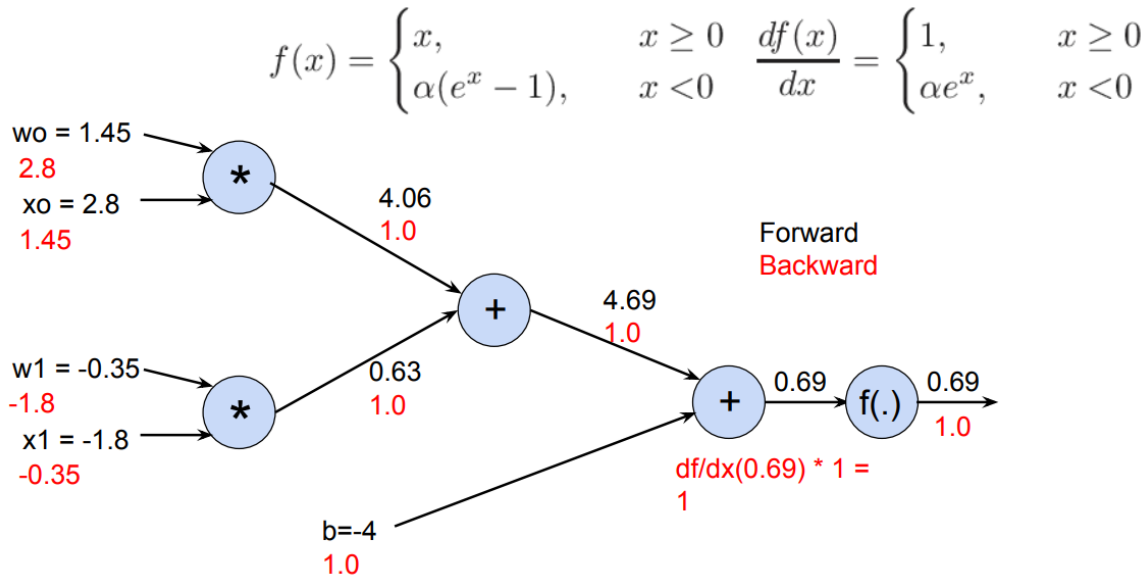


Figura 3: Resolución del ejercicio 1c.

Ejercicio 1d

En la Figura 5 se presenta la resolución del ítem d del ejercicio 1.

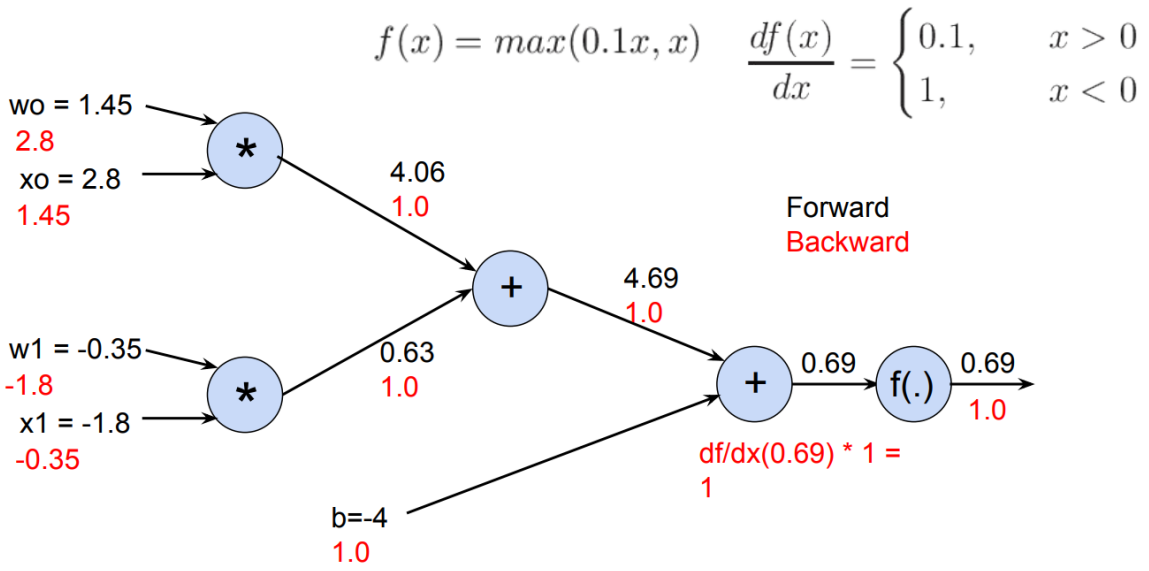


Figura 4: Resolución del ejercicio 1d.

Ejercicio 2

En la Figura 5 se presenta la resolución del ejercicio 2. Las cuentas auxiliares para este ejercicio se detallan en las imágenes que siguen a dicha figura.

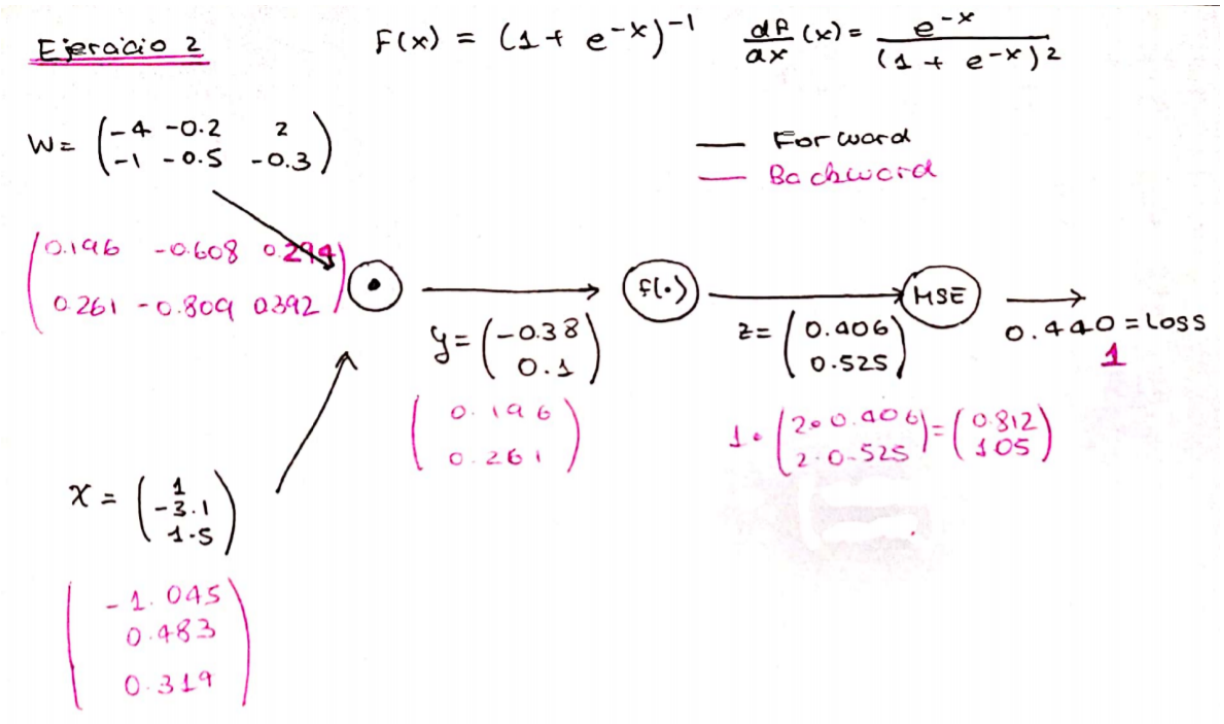


Figura 5: Resolución del ejercicio 2.

Cálculos auxiliares y números

- $f(-0.38) = 0.406$
- $f(0.1) = 0.525$
- $\text{Loss} = z_1^2 + z_2^2$
- $\frac{\partial \text{Loss}}{\partial z_i} = 2z_i$
- $\frac{\partial \text{Loss}}{\partial y_i} = \frac{\partial \text{Loss}}{\partial z_i} \frac{\partial z_i}{\partial y_i}$ pues $y_i \rightarrow f(y_i) = z_i$

  lo tengo para multiplicar → $\frac{\partial f(y_i)}{\partial y_i}$ como hacíamos en el 1 (a) básicamente

$$\frac{\partial \text{Loss}}{\partial y_1} = 0.812 \cdot \frac{\partial f}{\partial y}(-0.38) = 0.812 \cdot 0.241 = 0.196$$

$$\frac{\partial \text{Loss}}{\partial y_2} = 1.05 \cdot \frac{\partial f}{\partial y}(0.1) = 1.05 \cdot 0.249 = 0.261$$

- $\frac{\partial \text{Loss}}{\partial x_i} = \sum_k \frac{\partial \text{Loss}}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_i}$ $\quad y_k = \sum_j w_{kj} x_j \quad \frac{\partial y_k}{\partial x_i} = w_{ki}$

$$= \sum_k \frac{\partial \text{Loss}}{\partial y_k} \cdot w_{ki}$$

$$\nabla_x \text{loss} = W^T \cdot y = \begin{pmatrix} -4 & -1 \\ -0.2 & 2 \\ 2 & -0.3 \end{pmatrix} \begin{pmatrix} 0.196 \\ 0.261 \end{pmatrix} = \begin{pmatrix} -1.045 \\ 0.483 \\ 0.314 \end{pmatrix}$$

- $\frac{\partial \text{Loss}}{\partial w_{ij}} = \sum_k \frac{\partial \text{Loss}}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{ij}}$

$$y_k = \sum_l w_{kl} x_l$$

$$\frac{\partial y_k}{\partial w_{ij}} = \sum_l \delta_{ki} \delta_{jl} x_l = \delta_{ki} x_j$$

$$\frac{\partial \text{Loss}}{\partial w_{ij}} = \sum_k \frac{\partial \text{Loss}}{\partial y_k} \cdot \delta_{ki} x_j = \frac{\partial \text{Loss}}{\partial y_i} \cdot x_j$$

esto ya lo tengo del gradiente anterior → solo basta multiplicar así

paso anterior → $\begin{pmatrix} 0.196 \\ 0.261 \end{pmatrix}$

$W \rightarrow \begin{pmatrix} -4 & -0.2 & 2 \\ -1 & -0.5 & -0.3 \end{pmatrix}$

$x \rightarrow \begin{pmatrix} 1 \\ -3.1 \\ 1.5 \end{pmatrix}$

$$\nabla_W \text{loss} = \begin{pmatrix} 0.196 & -0.608 & 0.294 \\ 0.261 & -0.809 & 0.392 \end{pmatrix}$$

$$\nabla_W \text{loss} = \begin{pmatrix} 0.196 & -0.608 & 0.294 \\ 0.261 & -0.809 & 0.392 \end{pmatrix}$$

$$\frac{\partial \text{loss}}{\partial w_{11}} = 0.196 \cdot 1 = 0.196$$

$$\frac{\partial \text{loss}}{\partial w_{12}} = 0.196 \cdot -3.1 = -0.608$$

$$\frac{\partial \text{loss}}{\partial w_{13}} = 0.196 \cdot 1.5 = 0.294$$

$$\frac{\partial \text{loss}}{\partial w_{21}} = 0.261 \cdot 1 = 0.261$$

$$\frac{\partial \text{loss}}{\partial w_{22}} = 0.261 \cdot (-3.1) = -0.809$$

$$\frac{\partial \text{loss}}{\partial w_{23}} = 0.261 \cdot (1.5) = 0.392$$

Ejercicio 3

En la Figura 6 se presenta grafo computacional correspondiente a la red neuronal propuesta por este ejercicio para resolver cifar10. La primera capa de la misma consta de 100 neuronas con una activación sigmoide. En cambio, la segunda capa se compone de 10 neuronas con activación lineal. En cuanto a la función de costo, se toma una función de tipo MSE (Mean square error).

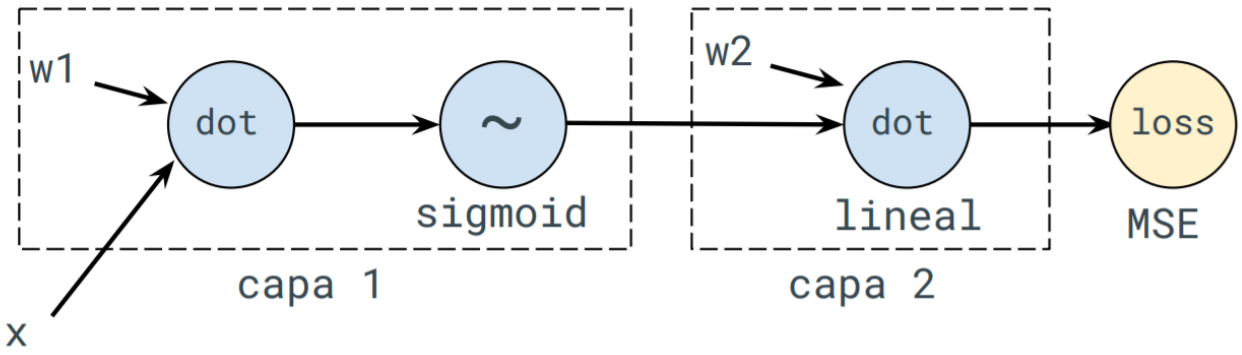


Figura 6: Grafo computacional correspondiente a la arquitectura planteada en el ejercicio 3.

El codigo para la resolución de este problema se encuentra en el archivo `p2-3.py`. No se utiliza para la misma el paradigma de programación orientada a objetos. En cambio, se definen funciones y, en base a las mismas, se resuelve el problema.

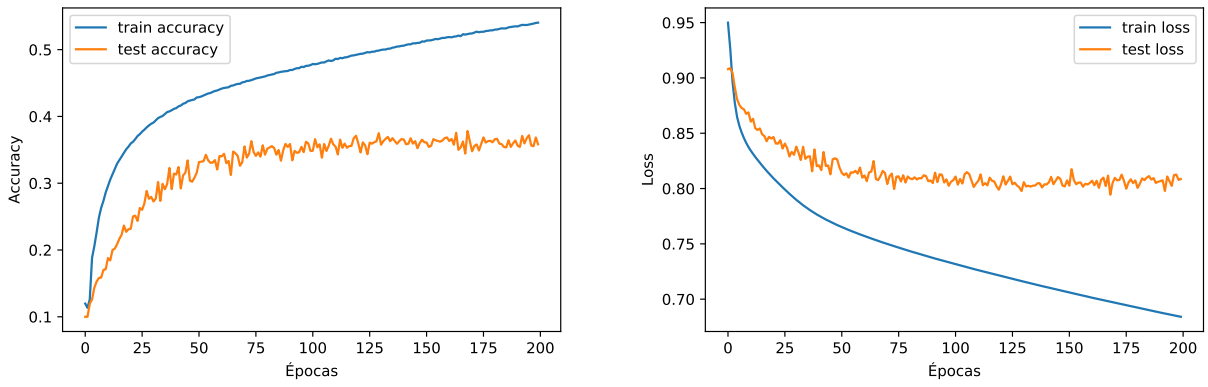
La función `reshapeImages` recibe las imágenes en el formato dado por cifar10 (32x32x3) y transforma a cada una de ellas en un vector fila de 3072 componentes. La función llamada `createRandomMatrix` recibe dimensiones `d1,d2` y crea una matriz de pesos aleatorios de tamaño `d1 x d2`. La función `randomizeMatrixRows` recibe datos `x,y`, donde `x` es la matriz de imagenes e `y` son sus scores, y los reordena de manera aleatoria. La función `addBias` recibe una matriz de imagenes y añade el bias como una primera columna de valor 1.

También se definen funciones como `accuracy` que recibe los scores verdaderos y los scores del modelo, y, como su nombre indica, calcula la accuracy del modelo. La función de costo recibe los mismos argumentos que la anterior y se implementa en la función `MSE`. Su gradiente se implementa en la función `grad_MSE`. Las ultima funciones auxiliar necesarias corresponden a la activacion y su gradiente, que se implementan como `sigmoid` y `grad_sigmoid` respectivamente.

Finalmente, la funcion `fit` es la que implementa la red neuronal. Esta recibe como argumentos las imagenes de train y test, al igual que sus scores, en el formato pedido por el ejercicio. Tambien recibe la cantidad de neuronas de ambas capas, el tamaño de batch, la cantidad de épocas, el learning rate y el parámetro para realizar una regularización de tipo L2. Esta funcion se sirve de `createRandomMatrix` para crear las matrices de pesos `w1` y `w2` utilizando la dimensión de los datos de train y test, y la cantidad de neuronas en cada capa. Luego, calcula el numero de iteraciones de batch por época según la cantidad de imágenes de train y el batch size. Luego, comienza a iterar por época, cambiando aleatoriamente el orden de las imagenes de train con `randomizeMatrixRows`. Entonces, dada una época, comienza la iteración de batch. Es decir, toma un batch de datos de train, hace forward en la red, calcula loss y accuracy para ese batch y los suma a variables `loss` y `acc`. Luego, realiza el camino backward en la red, calcula el gradiente y actualiza correspondientemente los pesos de las matrices `w1` y `w2`. Una vez recorridos todos los batchs para una época, hace promedios de `loss` y `acc` dividiendo por el numero de iteraciones, y los guarda en listas `train_loss` y `train_acc` respectivamente. Antes de comenzar la época siguiente, calcula loss y accuracy para los datos de test, y los guarda en listas `test_loss` y `test_acc`.

En el cuerpo principal del programa, se importan los datos de train y test de cifar10, y se corrige el formato de las imágenes utilizando la función `reshapeImages`. Luego, se hace un flatten de los datos de clases de train y test, y, utilizando fancy index, se los pasa al formato de scores pedido por el ejercicio. Este es tal que cada fila representa los scores de una imagen, y, para cada fila, hay un 1 en la columna de la categoría correcta y 0 en el resto de las columnas. Luego, se hace un ultimo preprocesado de las imágenes de train y test restando una imagen promedio de las imágenes de train a las mismas. Finalmente, se llama a la función fit. Para este caso, se utilizo un tamaño de batch `bs = 100`, 100 épocas, un learning rate `lr=5e-7` y un parametro de regularizacion `rg=2e-4`.

Los resultados de loss y accuracy en función de épocas para los datos de train y test se presentan en la Figura 7. Se observa en la Figura 7a el overfitting de los datos de training, cuya accuracy crece monótonamente con el numero de épocas, mientras que no sucede algo análogo para los datos de test. Se observa de la Figura 7b que la loss de los datos de train disminuye, mientras que la de los datos de test parece estancarse. Podría deberse a un learning rate alto.



(a) Accuracy en función del número de épocas. (b) Loss en función del número de épocas

Figura 7: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 3.

Finalmente, se busca comparar con los resultados de los clasificadores lineales del ejercicio 3 de la practica 1. Para ello, se volvieron a correr estos programas, cambiándose los parámetros utilizados en la práctica 1, con lo cual los valores de accuracy son distintos a los obtenidos anteriormente. Estos programas se incluyen en el archivo `LinearClassifiers.py`. Los gráficos de accuracy en función de épocas para estos clasificadores lineales se presentan junto con los resultados de este ejercicio para los datos de test en la Figura 8. Se observa que la accuracy alcanzada en este ejercicio es mayor que la obtenida por el clasificador SVM y menor que la obtenida por el SoftMax. Esto podría variar según los parámetros elegidos para cada modelo. Con los parámetros de la practica 1, por ejemplo, las accuracy de los clasificadores SoftMax y SVM para el problema cifar10 fue menor al 30 % (y por lo tanto menor a la accuracy obtenida en este ejercicio).

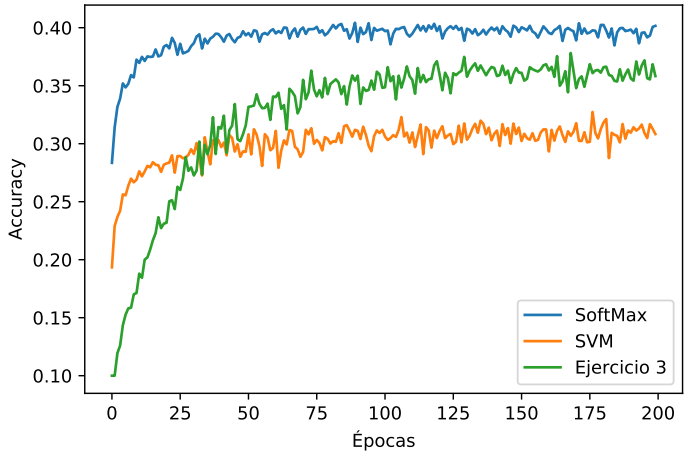


Figura 8: Accuracy para los datos de testing en función del numero de épocas para la arquitectura del ejercicio 3, y para los clasificadores lineales SoftMax y SVM implementados en la práctica 1.

Ejercicio 4

En la Figura 9 se presenta grafo computacional correspondiente a la red neuronal propuesta por este ejercicio para resolver cifar10. La primera capa de la misma consta de 100 neuronas con una activación sigmoide. En cambio, la segunda capa se compone de 10 neuronas con activación lineal. En cuanto a la función de costo, se toma una función de tipo CCE (Categorical Cross Entropy).

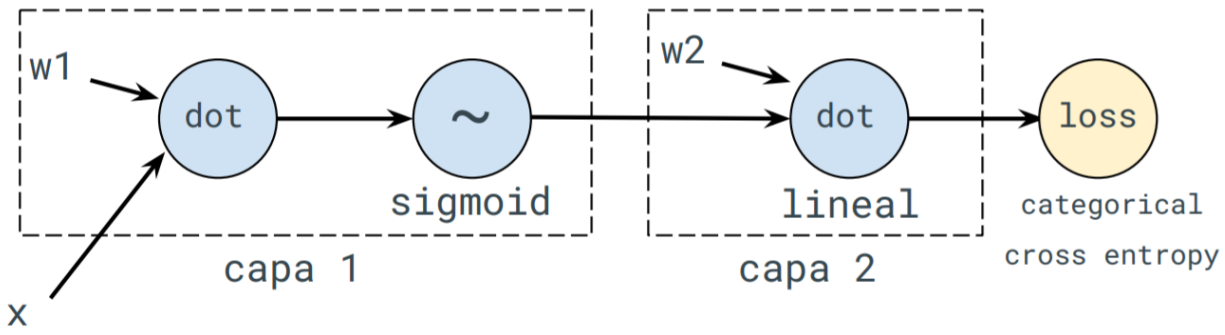
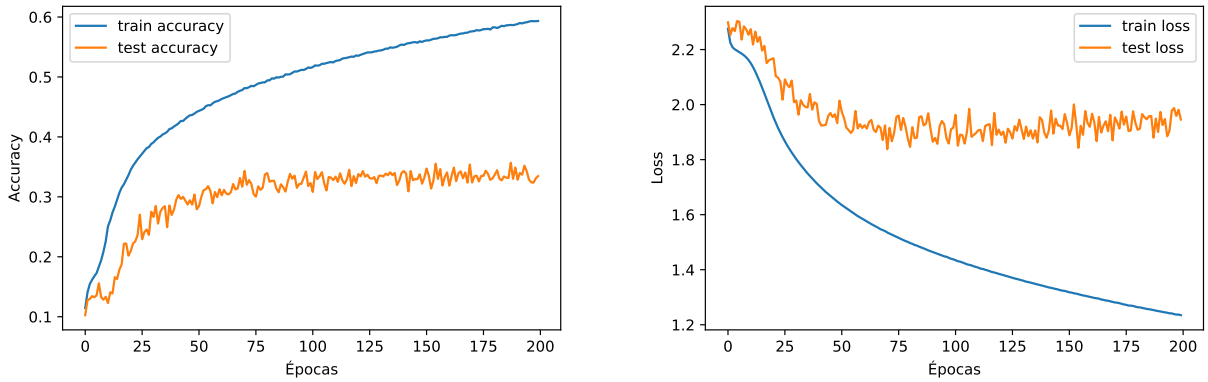


Figura 9: Grafo computacional correspondiente a la arquitectura planteada en el ejercicio 4.

La implementación de este ejercicio se encuentra en el archivo `p2-4.py` y es similar a la del ejercicio 3. Solamente es necesario sumar funciones `loss_softmax` y `grad_softmax` correspondientes a la función de costo y su gradiente.

En la Figura 10 se presentan los resultados de loss y accuracy para los datos de train y test. Se pueden hacer apreciaciones similares a las del ejercicio anterior con respecto a estas magnitudes.



(a) Accuracy en función del número de épocas.

(b) Loss en función del número de épocas.

Figura 10: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 4.

Finalmente, en la Figura 11 se visualizan los resultados de accuracy de los datos de test en función de las epocas para este ejercicio, el ejercicio anterior y los clasificadores lineales de la práctica 1. Se observa que la precisión alcanzada por la red neuronal de este ejercicio es menor que la del ejercicio anterior, aunque similar. Es mayor que la precisión alcanzada por el clasificador SVM y menor que el clasificador SoftMax.

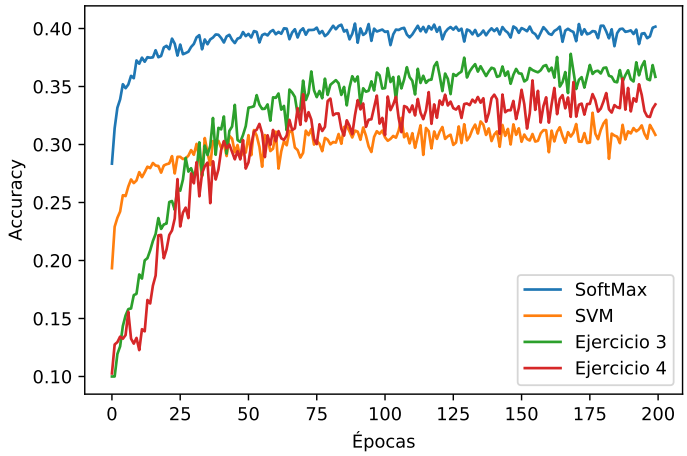


Figura 11: Accuracy para los datos de testing en función del numero de épocas para la arquitectura de los ejercicios 3 y 4, y para los clasificadores lineales SoftMax y SVM implementados en la práctica 1.

Ejercicio 5

Presentamos los resultados de este ejercicio separando en situaciones, segun los distintos grafos computacionales que se plantean. La situación 1 corresponde a la primera situación descripta en el enunciado, mientras que la segunda corresponde a la segunda descripción. Esta corresponde a cambiar la activación de la ultima capa por una activación lineal, mientras es sigmoide en la primera situación. La implementación de este ejercicio se basa en lo hecho en los ejercicios 3 y 4. Este ejercicio corresponde al archivo `p2-5.py`.

Situación 1

En la Figura 15 se presenta grafo computacional correspondiente a la primer red neuronal propuesta por este ejercicio para resolver cifar10. La primera capa de la misma consta de 100 neuronas con una activación ReLu. En cambio, la segunda capa se compone de 10 neuronas con activación sigmoide. Se utilizaron como funciones de costo la MSE y la CCE, implementadas anteriormente.

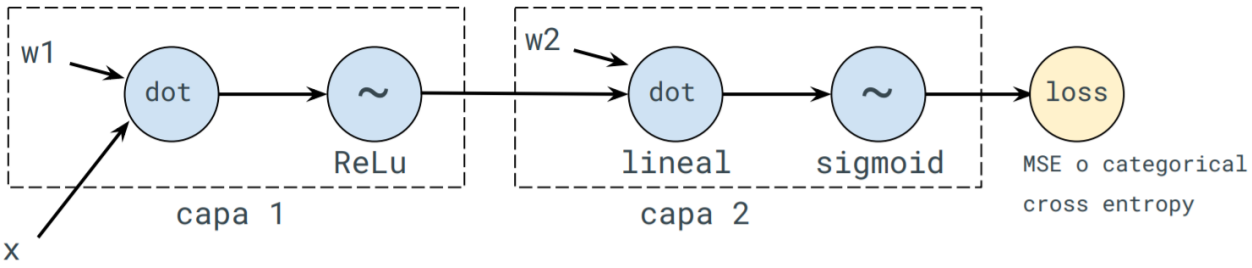
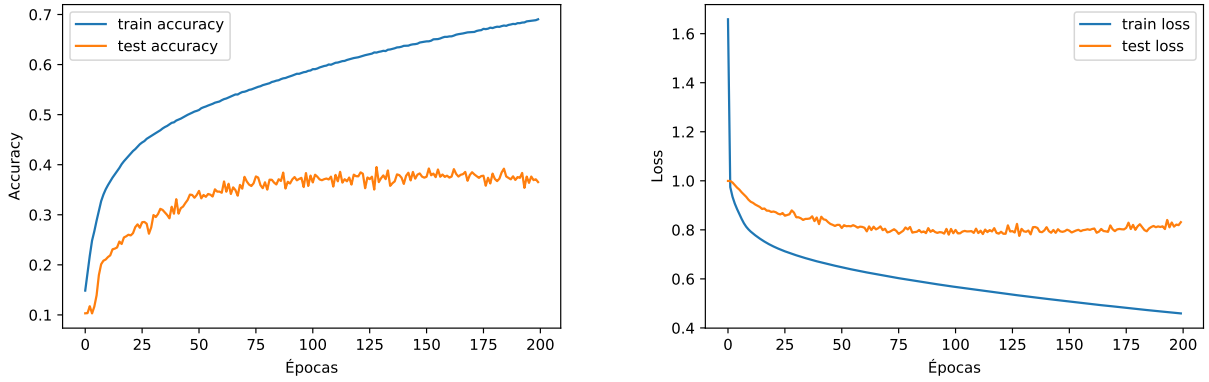


Figura 12: Grafo computacional correspondiente a la arquitectura planteada en la primera situación del ejercicio 5.

Función de costo MSE

En la Figura 13 se presentan los resultados obtenidos para la situación 1 del ejercicio, con una función de costo de tipo MSE.

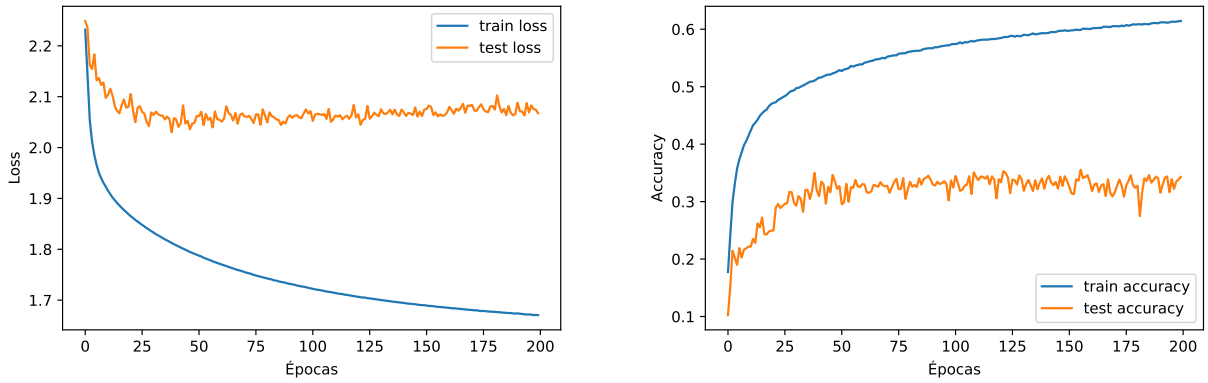


(a) Accuracy en función del número de épocas. (b) Loss en función del número de épocas.

Figura 13: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 5 - situación 1, con una función de costo MSE.

Función de costo CCE

En la Figura 13 se presentan los resultados obtenidos para la situación 1 del ejercicio, con una función de costo de tipo MSE.



(a) Accuracy en función del número de épocas.

(b) Loss en función del número de épocas.

Figura 14: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 5 - situación 1, con una función de costo CCE.

Situación 2

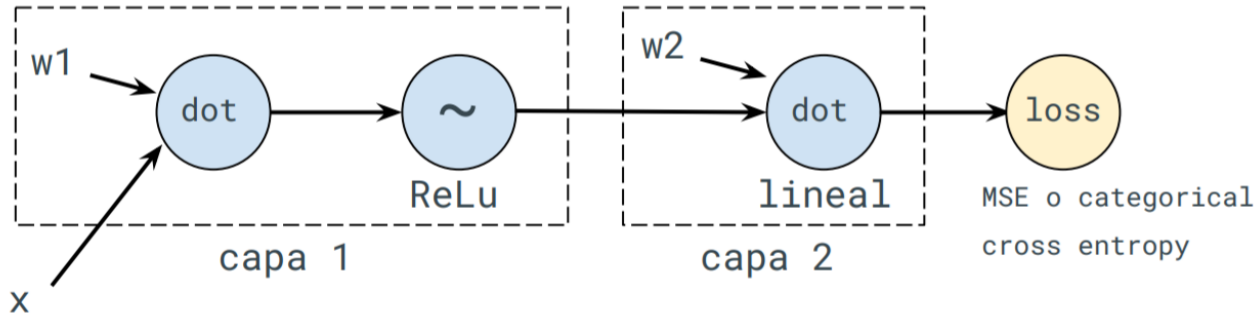
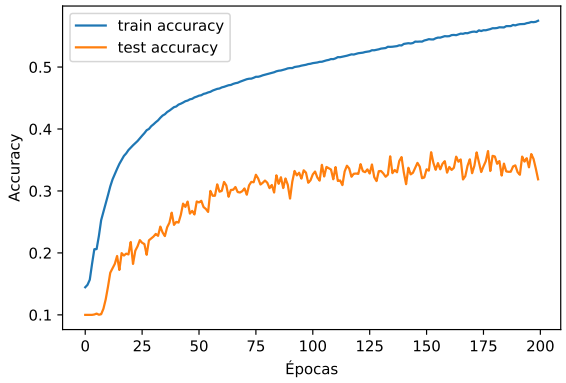
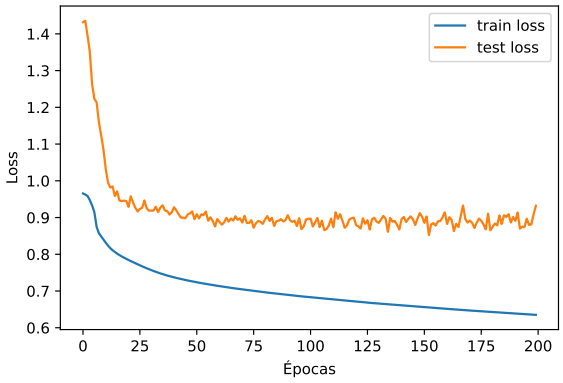


Figura 15: Grafo computacional correspondiente a la arquitectura planteada en la segunda situación del ejercicio 5.

Función de costo MSE



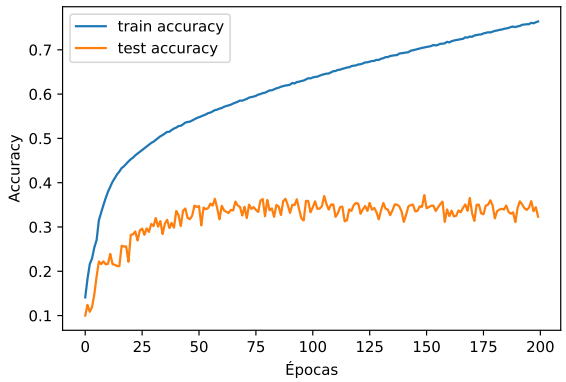
(a) Accuracy en función del número de épocas.



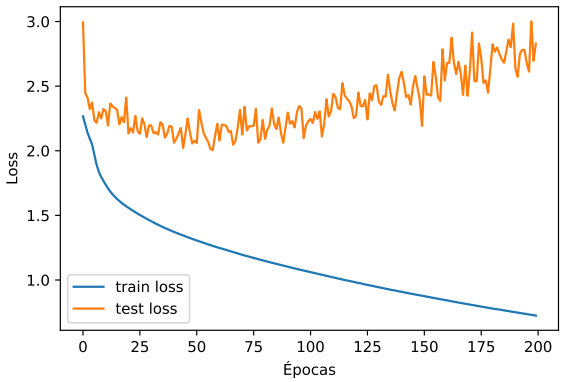
(b) Loss en función del número de épocas.

Figura 16: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 5 - situación 2, con una función de costo MSE.

Función de costo Categorical Cross Entropy



(a) Accuracy en función del número de épocas.



(b) Loss en función del número de épocas.

Figura 17: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 5 - situación 2, con una función de costo de tipo Categorical Cross Entropy.

Ejercicio 6

Situación 1

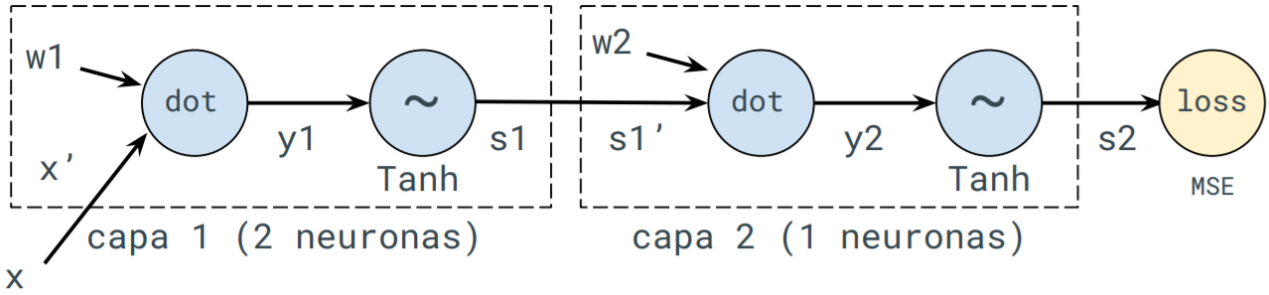
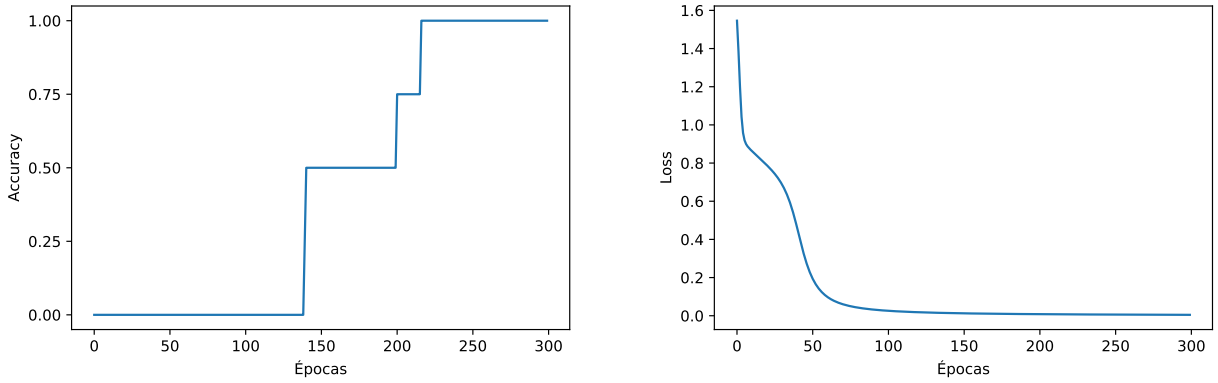


Figura 18: Grafo computacional correspondiente a la arquitectura planteada en la segunda situación del ejercicio 6.



(a) Accuracy en función del número de épocas.

(b) Loss en función del número de épocas.

Figura 19: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 6 - situación 1, con una función de costo de tipo MSE.

Situación 2

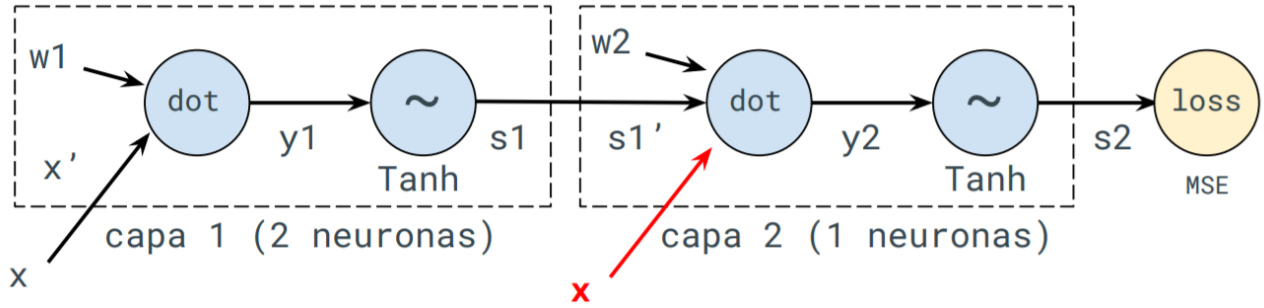


Figura 20: Grafo computacional correspondiente a la arquitectura planteada en la segunda situación del ejercicio 6.

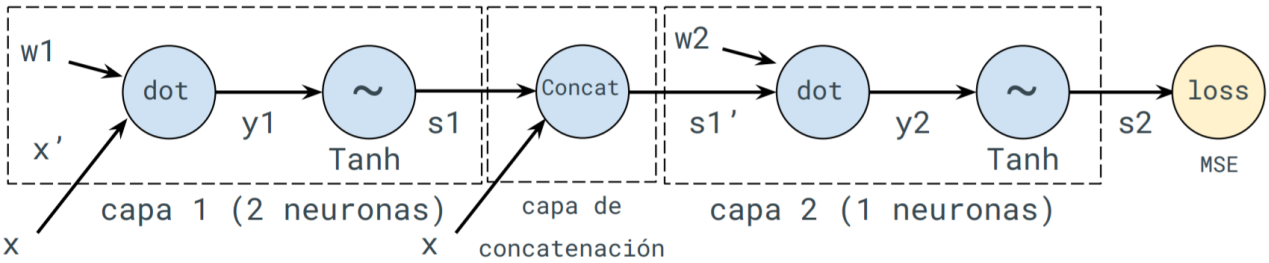
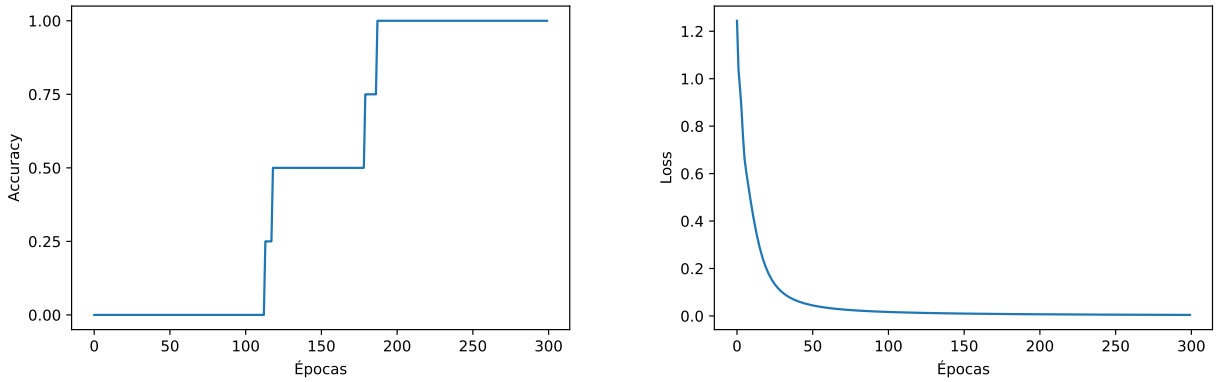


Figura 21: Grafo computacional alternativo correspondiente a la arquitectura planteada en la segunda situación del ejercicio 6, realizado para la implementación.



(a) Accuracy en función del número de épocas.

(b) Loss en función del número de épocas.

Figura 22: Accuracy y loss en función del numero de épocas para la arquitectura del ejercicio 6 - situación 2, con una función de costo de tipo MSE.

Ejercicio 7

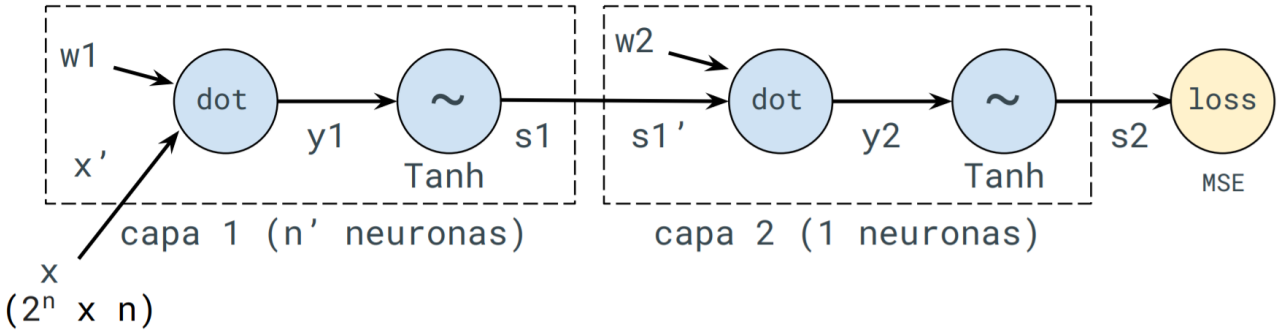


Figura 23: Grafo computacional correspondiente a la arquitectura planteada en la segunda situación del ejercicio 7.

Ejercicio 8

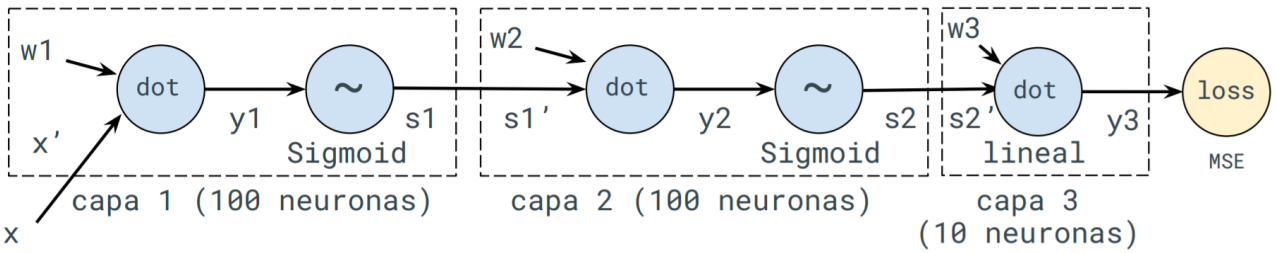


Figura 24: Grafo computacional correspondiente a la arquitectura planteada en la segunda situación del ejercicio 8.