2.- Utilizando el código desarrollado en clase como base, investigue cómo entrenar una red neuronal de perceptrones muticapa. Entrene el modelo para la clasificación de la base de datos Iris y calcule su precisión. Use el 80% de los datos para el entrenamiento y el 20% para su evaluación, agregue al informe cuál es el error calculado de la clasificación de la red y muestre cómo seleccionó el 80% de los datos de entrenamiento (60%).

1. Entregar:
   1. Informe: Añadir al informe de la tarea, una explicación breve de cómo entrenó el modelo y los parámetros utilizados. Describa cómo seleccionó el 80% de los datos para entrenamiento. Agregue cuál fue el error calculado de la clasificación. Pruebe dos arquitecturas distintas y discuta qué efecto tiene la arquitectura con respecto al error de clasificación.
   2. Código: Incluya los scripts utilizados al igual que los modelos entrenados en dos archivos pkl.

En clase se estudió la base de un perceptrón y para poder resolver este ejercicio es importante conocer su funcionamiento como base. Este tipo de red se basa en el uso de varias capas de neuronas artificiales es decir varios perceptrones

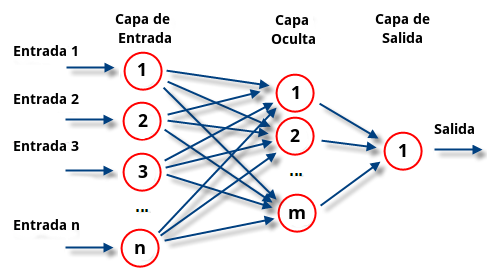


Figura 5. Percpetrones multicapa[2]

Esto permitió a las redes neuronales aprender las variaciones no lineales de los distintos tipos de ambientes, que en su mayoría, presentan variaciones del tipo no lineal. Como un ejemplo el XOR.

Cuando se aborda un problema utilizando el perceptron multicapa, uno de los primeros pasos a realizar es el diseño de la arquitectura de la red. Este diseño implica la determinación de la función de activación a emplear, el número de neuronas y el número de capas de la red [3].

En lo que respecta al número de neuronas y capas, algunos de estos parámetros vienen dados por el problema y otros deben ser elegidos por el diseñador. Así, por ejemplo, tanto el número de neuronas en la capa de entrada, como el número de neuronas en la capa de salida, vienen dados por las variables que definen el problema.

No existe un método o regla que determine el número óptimo de neuronas ocultas para resolver un problema dado. En la mayor parte de las aplicaciones prácticas, estos parámetros se determinan por prueba y error.

Para este ejercicio se programaron varias funciones, las principales se detallan a continuación

Una función llamada initialize\_network() que crea una nueva red neuronal lista para el entrenamiento. Acepta tres parámetros, el número de entradas, el número de neuronas que debe tener en la capa oculta y el número de salidas. Cabe resaltar que en el programa lo valores de salidas se los asocia a un nuevo vector que inicializa desde cero, es decir si la base de datos tiene como salida valores [5,7,9], el programa los asociará como [0,1,2].

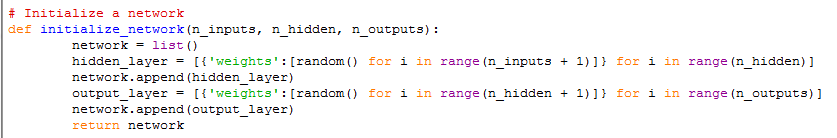


Figura 6. Código implementado

Como se mencionó, la red necesita de un número de época. Esto implica que el primer bucle tendrá un número fijo de épocas y con cada época se actualizará la red usando la fila de los datos. Debido a que se hacen actualizaciones para cada patrón de entrenamiento, este tipo de aprendizaje se llama aprendizaje en línea.

A continuación se muestra una función que implementa el entrenamiento de la red neuronal ya inicializada con un conjunto de datos de entrenamiento dado, tasa de aprendizaje, número fijo de épocas y un número esperado de valores de salida.

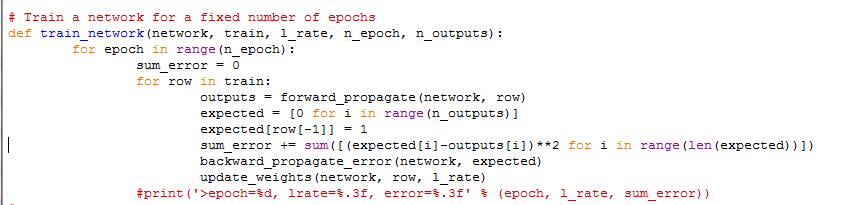


Figura 7. Código implementado

El número esperado de valores de salida se utiliza para transformar valores de clase en los datos de entrenamiento. Ése es un vector binario con una columna para cada valor de la clase para emparejar la salida de la red. Esto es necesario para calcular el error de la capa de salida.

Entonces para este ejercicio se tiene una base de datos “iris\_data.psv” es importante seleccionar los datos adecuadamente para no tener problemas al entrenar la red y luego predecir los valores. Para esto se usó el método de muestreo cuadrático estratificado, ya que esta base de datos es no lineal, está dividida en subgrupos (tipos de iris) y necesitamos seleccionar porciones de los datos que sean efectivos de cada subgrupo.

Para esto se usó la herramienta RapidMiner

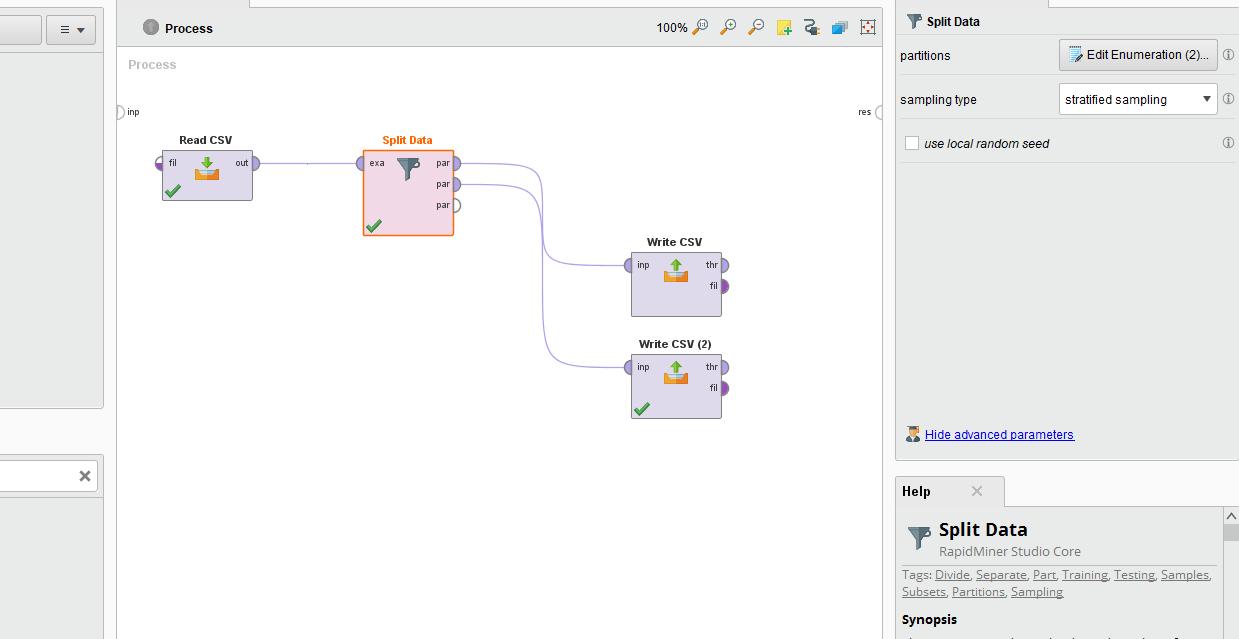


Figura 8. Selección de datos RapidMiner

La cual nos da dos nuevas bases datos, el 80% de la base de datos llamada “data802.csv” y el 20% llamada “data202.csv”.

Siguiente paso es tratar de entrenar la red con los datos, como se mencionó anteriormente el investigador es el que trata de ajustar cuantas épocas, neuronas y salidas debe tener su red. Para este caso luego de algunos entrenamientos se determinó los siguientes parámetros

* Tasa de aprendizaje: 0.3
* Épocas: 900
* Capas ocultas: 6

Con estos valores nos da como resultado los pesos “W”, y sus errores

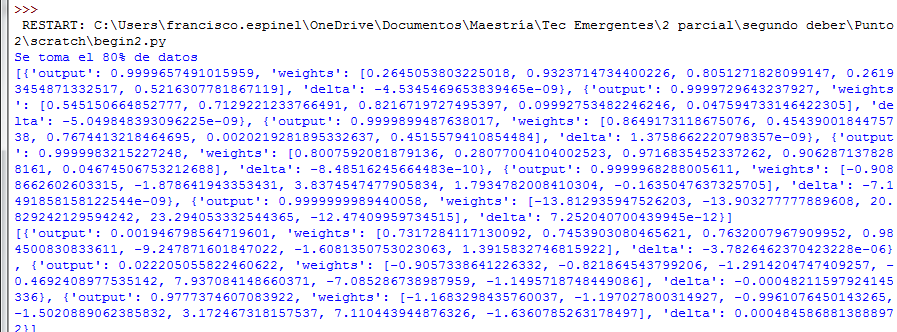
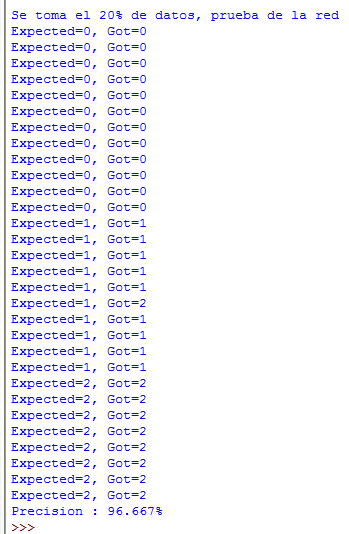


Figura 9. Red entrenada, programa ejecutado con el 80% de datos

Finalmente ponemos a prueba la red, ingresando el 20% de los datos con lo cual nos da un error del 3.33%



Conclusiones:

Cuando se aborda cualquier caso de perceptrones multicapa, el primer paso a realizar es saber que arquitectura usar de la red, es mejor saber a profundidad el problema para conocer qué datos se pueden omitir o que podría estar fallando

Para este caso con una se debió correr 10 veces e ir colocando valores a azar a la red para llegar a un error aceptable.