1. Desarrollo

1.1. Implementacion de la red neuronal

La implementación de la red neuronal se realizó en MatLab. Decidimos que era el entorno más apropiado para el trabajo, ya que por tratarse de un trabajo de experimentación no estábamos buscando performance y nos brinda diversas comodidades en operaciones con matrices.

Se utilizó la implementación de la red neuronal como se presentó en clase, y se agrupó cada parte de la misma en una clase definida en MatLab, bajo el nombre de **MyMultiPerceptron**. Puntualmente aplicamos aprendizaje incremental para el entrenamiento e implementamos la modificación del **momentum** para que la red pueda converger más rápidamente. Tanto el momentum (α) como el learning rate (γ) no se modifican durante el entrenamiento.

A modo de dejar bien asentada la implementación, la presentamos a continuación en formato de pseudocódigo. Sin embargo, no daremos mayores explicaciones ya que no presenta ninguna innovación. La función de activación fue reemplazada correspondientemente según el modo en el que la red neuronal estuviese corriendo, daremos más detalles en la sección que especifica sus opciones.

function FEEDFORWARD(w, x)
  y = propagateFeed(w, x)
  return $y_{|y|}$
end function

function PROPAGATEFEED(w, x)
  $y_1 = x$
  for $i = [1..|\text{w}|]$ **do**
    y_{i+1} =activation($[y_i\ 1] * \text{w}_i$)
  end for
  return y
end function

function TRAIN(w, data, err_{min}, ep_{max}, γ, α)
  $\Delta lw = \emptyset$
  while $e\ > \text{err}_{min} \wedge t\ < \text{ep}_{max}$ **do**
    $e = 0$
    $t = t + 1$
    for $x, z = [1..|\text{data}|]$ **do**
      y = propagateFeed(x)
      $[fe\ \Delta w]$ = correction(w, y, z, γ)
      w = adaptation(w, Δw, Δlw, α)
      $\Delta lw = \Delta w$
      $e = e + fe$
    end for
  end while
end function

function CORRECTION(w, y, z, γ)
  $\Delta w = \emptyset$
  $re = z - y_{|y|}$
  $fe = ||re||_1/|re|$
  for $i = [|w|.,1]$ **do**
    $re = re$.* activation'(y_{i+1})
    $\Delta w_i = \gamma\ *\ [y_i\ 1]'\ *\ re$
    $re = re * \text{w}_{[1..|w|-1]}$
  end for
  return $[fe\ \Delta w]$
end function

function ADAPTATION(w, Δw, Δlw, α)
  for $i = [1..|w|]$ **do**
    $\text{w}_i = \text{w}_i + \Delta w + \alpha * \Delta lw$
  end for
  return w
end function

1.2. Preprocesamiento de los datos

A fin de lograr que todas las características observadas tengan la misma influencia en la red (y evitar que la influencia de alguna característica con amplitud y valor absoluto en sus valores sea mayor que la de una con análogos menores), analizamos los resultados al hacer un preprocesamiento de cada caraterística en las bases de datos.

Como preprocesamiento elegimos realizar una normalización de cada característica numérica, incluyendo la salida en el segundo conjunto de datos.

1.3. Generación de instancias de prueba

Con el objetivo de evaluar diferentes arquitecturas de redes decidimos particionar el conjunto de datos disponibles (realizando la misma operación para cada dataset) utilizando k-fold cross-validation para disminuir los efectos de tomar conjuntos poco representativos y analizar el funcionamiento de una determinada selección de parámetros sobre distintas particiones de los datos. Sin embargo, considerando que debemos realizar una serie de pruebas en un tiempo acotado, decidimos acotarlas de la siguiente manera.

Particionamos la totalidad de los datos en cuatro partes, cada una considerando el 25 % de los mismos. Dada esa partición generamos 4 folds diferentes. Así, entrenamos con el 75 % y testeamos en el 25 % restante con cada una de las variantes.

1.4. Variantes consideradas

El desarrollo del trabajo consistió fundamentalmente en analizar distintas variantes para la red a fin de obtener parámetros que permitiesen obtener buenos resultados en cada problema. Para ello, cada uno de los mismos fue tratado de manera independiente y se realizaron distintas pruebas que se detallan a continuación.

1.4.1. Arquitectura adecuada al problema

Uno de los aspectos fundamentales sobre el funcionamiento de una red neuronal es su configuración respecto de la cantidad de capas y de neuronas en cada una de ellas. Durante el desarrollo consideramos distintas configuraciones, pero dado que los problemas son simples una única capa oculta deberia ser suficiente (aunque esto es cierto para la mayoria de los problemas[**?**]). Finalmente nos queda elegir la cantidad de neuronas en dicha capa.

Para tener un punto de referencia sobre el cual comenzar a evaluar los rendimientos de las arquitecturas consideramos una heurística que nos sugiere elegir una cantidad de neuronas en la capa oculta igual a la cantidad a la cantidad de entradas mas la cantidad de salidas sobre dos[**?**].

En el caso del conjunto de datos del problema 1, dado que hay 30 entradas y 1 salida evaluamos arquitecturas donde variamos las neuronas de la capa oculta en el conjunto $\{5, 10, 15, 20, 25\}$.

Para el problema 2, dado que hay 8 entradas y 2 salidas, evaluamos una capa oculta con cantidades en el conjunto $\{2, 4, 6, 8, 10, 12, 14\}$.

1.4.2. Épocas necesarias para obtener un buen resultados

Otro punto clave en el entrenamiento de la red es la cantidad de épocas a utilizar. Para evaluar este aspecto consideramos distintas cantidades para cada una de las arquitecturas mencionadas previamente. Los maximos de las mismas fueron elegidas a partir de que observamos que la red convergia suficientemente bien en dichos valores. El momentum durante estas pruebas se encontraba desactivado ($\alpha = 0$).

Para el problema 1, entrenamos usando $10, 20, 30, 40, 50, 60, 70, 80, 90, 100$ épocas y para el problema 2 fueron $500, 1000, 1500, 2000$.

1.4.3. Utilización de distintos valores de learning rate

Una vez escojidas arquitecturas y cantidades de iteraciones que permitieron obtener buenos resultados realizamos variaciones en el learning rate. Realizamos evaluaciones utilizando los valores $\gamma = 0{,}1, 0{,}2, 0{,}3, 0{,}4, 0{,}5$.

1.4.4. Normalización de los datos

En el primer problema solamente era posible normalizar los datos de entrada ya que la salida corresponde a una clasificacion. Sin embargo en el segundo problema, donde se trata de una regresion, la salida puede normalizarse para el entrenamiento, lo que puede producir cambios significativos en la red. Evaluamos este punto para corroborar si se encontraban diferencias significativas.

1.4.5. Momentum

Una vez elegida la arquitectura a utilizar, evaluamos la misma fijando una cantidad de épocas suficientes para asegurar su convergencia y variando el momentum para examinar sus efectos. Los valores elegidos para evaluar este punto fueron los contenidos en el conjunto

$$\alpha = \{i * 0{,}05 \mid i \in [1, 19] \subset N\}$$