CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

APRENDIZAJE INTELIGENTE

6° "A"

PRIMERA EVALUACIÓN PARCIAL

Profesor: Francisco Javier Luna Rosas

Alumnos:

Espinoza Sánchez Joel Alejandro

Gómez Garza Dariana

González Arenas Fernando Francisco

Fecha de Entrega: Aguascalientes, Ags.**, 2** de marzo de 2021

Primera Evaluación Parcial

Evidencias del Examen

1. Explicación de una red neuronal con propagación hacia atrás

Una red de neuronas artificial puede considerarse como un grafo dirigido ponderado en el que las neuronas artificiales son nodos y las hojas dirigidas y ponderadas son las conexiones entre salidas y entradas.

Dependiendo del patrón de conexión puede dividirse en:

- Redes de propagación hacia adelante (feedfoward): en las que los grafos no tienen bucles.
- Recurrentes o de retroalimentación (feedback), en las cuales los bucles ocurren debido a conexiones de retroalimentación

La propagación hacia atrás de errores o retro propagación (del inglés *backpropagation*) es un algoritmo de aprendizaje supervisado que se usa para entrenar redes neuronales artificiales. El algoritmo consiste en minimizar un error (comúnmente cuadrático) por medio de gradiente descendiente, por lo que la parte esencial del algoritmo es cálculo de las derivadas parciales de dicho error con respecto a los parámetros de la red neuronal.

Una red neuronal multicapa puede computar una salida continua en vez de una función escalonada. Una elección común es la renombrada función logística:

$$y = \frac{1}{1 + e^{-x}}$$

Con esta elección, la red de capa única es idéntica al modelo de regresión logística, ampliamente utilizado en modelado estadístico. La función logística es también conocida como función sigmoide. Tiene una derivada continua, la cual le deja ser utilizada en propagación hacia atrás.

Esta clase de redes consta de capas múltiples de unidades computacionales, normalmente interconectados de una manera prealimentada. Cada neurona en una capa tiene conexiones dirigidas a las neuronas de la capa siguiente. En muchas aplicaciones las unidades de estas redes realizan una función sigmoide como función de activación.

El *teorema de aproximación universal* para redes neuronales establece que cada función continua que mapea intervalos de números reales a algún intervalo de salida de números reales puede ser aproximado arbitrariamente por un perceptrón multicapa con una sola capa escondida.

Redes multicapa utilizan una variedad de técnicas de aprendizaje, entre las cuales la más popular es la de Propagación hacia atrás. Aquí, los valores de salida son comparados con la respuesta correcta para computar el valor de alguna función de error predefinida. Por medio de varias técnicas, el error es entonces retroalimentado a través de la red. Utilizando esta información, el algoritmo ajusta las ponderaciones de cada conexión para reducir el valor de la función de error en cantidades pequeñas.

Después de repetir este proceso por un número suficientemente grande de ciclos de entrenamiento, la red normalmente convergerá en algún estado donde el error de los cálculos sea pequeño. En este caso, uno diría que la red ha *aprendido* una función objetivo en específico. Para ajustar las ponderaciones correctamente, uno aplica un método general para optimización no lineal denominado descenso de gradiente. Para ello, la derivada de la función de error con respecto a las ponderaciones de la red es calculada y las últimas son modificadas de tal manera que el error disminuye (así pues, yendo en picada a lo largo de la superficie de la función de error). Por esta razón, a propagación hacia atrás sólo puede ser aplicada en redes con funciones de activación diferenciables.

Profundizando en el ámbito de las RNA, si se utiliza una de especial arquitectura llamada feedforward, el Dr. George Cybenko demostró que, con una configuración

de dos capas y un número suficiente de neuronas, se puede aproximar cualquier función continua a un grado de precisión arbitrario. Por lo tanto, no sorprende ver a lo largo de la literatura que el tipo de red neuronal artificial más utilizado sea el RNAf. Es de notarse que, si bien se han buscado en la inteligencia artificial nuevos enfoques para lidiar con los problemas de predicción e identificación de patrones, los profesores Warner y Misra han demostrado también que hay bastante similitud en los fundamentos sobre los que operan el análisis generalizado de regresión y las RNA del tipo multicapa feedforward (RNAf). Una RNAf se caracteriza por ser un conjunto de neuronas que reciben información multivariable, la procesan y dan una respuesta que puede ser multivariable también. En la arquitectura feedforward la topología del arreglo de neuronas y sus interconexiones hace fluir la información de forma unidireccional para que nunca pueda pasar más de una vez a través de una neurona antes de generarse la respuesta de salida. Tal como muestra la figura 1.1.

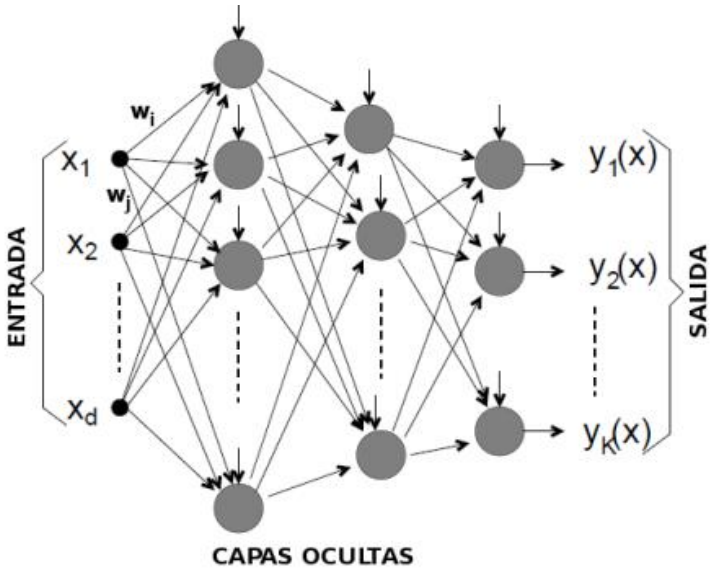

Figura 1.1 Diagrama de la estructura típica de una red neuronal artificial tipo *feedforward* (RNAf).

Las redes neuronales pueden utilizarse en un gran número y variedad de aplicaciones, tanto comerciales como militares.

Se pueden desarrollar redes neuronales en un periodo de tiempo razonable, con la capacidad de realizar tareas concretas mejor que otras tecnologías. Cuando se implementan mediante hardware (redes neuronales en chips VLSI), presentan una alta tolerancia a fallos del sistema y proporcionan un alto grado de paralelismo en el procesamiento de datos. Esto posibilita la inserción de redes neuronales de bajo coste en sistemas existentes y recientemente desarrollados.

Hay muchos tipos diferentes de redes neuronales; cada uno de los cuales tiene una aplicación particular más apropiada. Algunas aplicaciones comerciales son:

- Biología:
  - Aprender más acerca del cerebro y otros sistemas.
  - Obtención de modelos de la retina.
- Empresa:
  - Evaluación de probabilidad de formaciones geológicas y petrolíferas.
  - Identificación de candidatos para posiciones específicas.
  - Explotación de bases de datos.
  - Optimización de plazas y horarios en líneas de vuelo.
  - Optimización del flujo del tránsito controlando convenientemente la temporización de los semáforos.
  - Reconocimiento de caracteres escritos.
  - Modelado de sistemas para automatización y control.
- Medio ambiente:
  - Analizar tendencias y patrones.
  - Previsión del tiempo.
- Finanzas:
  - Previsión de la evolución de los precios.
  - Valoración del riesgo de los créditos.
  - Identificación de falsificaciones.
  - Interpretación de firmas.
- Manufacturación:
  - Robots automatizados y sistemas de control (visión artificial y sensores

de presión, temperatura, gas, etc.).
- Control de producción en líneas de procesos.
- Inspección de la calidad.

- Medicina:
  - Analizadores del habla para ayudar en la audición de sordos profundos.
  - Diagnóstico y tratamiento a partir de síntomas y/o de datos analíticos (electrocardiograma, encefalogramas, análisis sanguíneo, etc.).
  - Monitorización en cirugías.
  - Predicción de reacciones adversas en los medicamentos.
  - Entendimiento de la causa de los ataques cardíacos.
- Militares:
  - Clasificación de las señales de radar.
  - Creación de armas inteligentes.
  - Optimización del uso de recursos escasos.
  - Reconocimiento y seguimiento en el tiro al blanco

La mayoría de estas aplicaciones consisten en realizar un reconocimiento de patrones, como ser: buscar un patrón en una serie de ejemplos, clasificar patrones, completar una señal a partir de valores parciales o reconstruir el patrón correcto partiendo de uno distorsionado. Sin embargo, está creciendo el uso de redes neuronales en distintos tipos de sistemas de control.

2. Explicación y análisis

1. La explicación de las entradas de la red son las siguientes:

¿En que consiste los dataset de entrenamiento y sus valores target asociados?

El dataset de entrenamiento de una red neuronal artificial, son los datos que se le proporcionan a la red para que aprenda a realizar la tarea para la que fue creada. Estos datos van entrenando la red neuronal artificial poco a poco, disminuyendo el error en las tareas que realiza y de esa forma dar mejores resultados a datos ingresados posteriormente a la fase de entrenamiento.

El target asociado a los dataset de las redes neuronales, indican los valores esperados a los que se quiere llegar con el procesamiento de la información por las neuronas. Una vez hecho todo el proceso dentro de las neuronas y diferentes capas, se compara la salida resultante de la red con el target, es decir, con la salida deseada asociada a ese conjunto de datos ingresados al modelo.

La razón de aprendizaje de las redes neuronales Feed-Forward.

Las redes neuronales artificiales pueden ser entrenadas mediante un algoritmo de aprendizaje llamado regla delta, el cual calcula los errores entre la salida actual y la salida deseada, utilizando para esto algo llamado como descenso del gradiente. De esta forma las redes neuronales pueden aprender cada que se modifica el error a través del descenso del gradiente, disminuyendo el error en cada época.

El bias de las redes neuronales Feed-Forward.

El bias es el sesgo que tienen las neuronas, el cual indica que tan predispuesta esta la neurona a dar un resultado de 1 o 0 independientemente de los pesos de la neurona. Un sesgo alto, hace que la neurona requiera un peso mayor para los parámetros y así conseguir una salida de 1; un bias o sesgo bajo, favorece a la neurona para que pueda trabajar con pesos más bajos.

Los pesos de entrenamiento de las redes neuronales Feed-Forward.

Los pesos de entrenamiento de las redes neuronales, generalmente representados por w, son los pesos o influencia que tiene una determinada entrada x en el procesamiento de la neurona. Entre mayor peso tenga una entrada, mayor influencia tendrá en el resultado final y por consiguiente, mayor margen de error o responsabilidad al momento de calcular el error de la red.

2. La explicación de las salidas de la red se dan a partir de dos fórmulas. La primera es la siguiente:

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Podemos ejemplificarlo con las neuronas:

$$I_4 = (w_{14}O_1) + (w_{24}O_2) + (w_{34}O_3) + \theta_4 \qquad I_4 = (0.2 \times 1) + (0.4 \times 0) + (-0.5 \times 1) - 0.4$$

$$I_5 = (w_{15}O_1) + (w_{25}O_2) + (w_{35}O_3) + \theta_5 \qquad I_5 = (-0.3 \times 1) + (0.1 \times 0) + (0.2 \times 1) + 0.2$$

De esta forma obtenemos los valores que arroja la red en dicha capa, que son:

$$I_4 = -0.7$$

$$I_5 = 0.1$$

Y esto es realizado debido a que cada neurona funciona como una ponderación en forma de regresión lineal, que arroja los valores que se observan anteriormente.

Ahora hay que ajustar su valor para la entrada de la siguiente red, que es lo que se realiza con la siguiente fórmula:

$$O_j = \frac{1}{1 + e^{-I_j}}$$

Y los resultados son:

$$O_4 = 0.331812$$

$$O_5 = 0.524979$$

Y nuevamente se obtiene el valor de la red con la fórmula ya presentada:

$$I_6 = (w_{46}O_4) + (w_{56}O_5) + +\theta_6 \qquad I_6 = (-0.3 \times 0.331812) + (-0.2 \times 0.524979) + 0.1$$

$$I_6 = -0.104539$$

Y éste último también se ajusta:

$$O_6 = 0.473889$$

3. Ahora es necesario calcular el error:

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

$$Err_6 = 0.473889(1 - 0.473889)(1 - 0.473889)$$

$$Err_6 = 0.131169$$

$$Err_5 = 0.524979(1 - 0.524979)(1 - 0.524979)$$

$$Err_5 = 0.118458$$

$$Err_4 = 0.331812(1 - 0.331812)(1 - 0.331812)$$

$$Err_4 = 0.148145$$

4. Actualizamos finalmente los pesos y bias:

$$\Delta w_{ij} = (l) Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

$$\therefore w_{ij} = w_{ij} + (l) Err_j O_i$$

$$\Delta \theta_j = (l) Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

$$\therefore \theta_j = \theta_j + (l) Err_j$$

Por ejemplo:

$$w_{14} = 0.2 + (0.9)(0.148145)(1)$$

$$\therefore w_{14} = 0.133330$$

Este proceso de actualización se realiza con todos los pesos.

3. Implementación

El código fue desarrollado en Python 3 siguiendo estrictamente el diseño el examen. En un principio se declararon las variables de inicio como se observa en la siguiente imagen:

```python
import numpy

'''
Los valores de enter son:
    x1 x2 x3 w14 w15 w24 w25 w34 w35 w46 w56 t4 t5 t6
'''

enter = [1, 0, 1, 0.2, -0.3, 0.4, 0.1, -0.5, 0.2, -0.3, -0.2, -0.4, 0.2, 0-1]
Tj = 1
l = 0.9
O6 = 0

print(enter)
print(O6)
```

De la imagen se puede observar que se declaró el comienzo de los valores de la neurona; planteando un vector de nombre `enter`, donde cada posición representaba el siguiente valor:

$$enter = [x_1, x_2, x_3, w_{14}, w_{15}, w_{24}, w_{25}, w_{34}, w_{35}, w_{46}, w_{56}, \theta_4, \theta_5, \theta_6]$$

Después se crearon las variables `I4`, `I5`, y `I6` con el objetivo de representar las entradas hacia delante de cada neurona. De la misma forma existen las variables `O4`, `O5`, y `O6` para el cálculo de entradas y salidas de las neuronas.

```python
while (O6 < 1):
    #%% Calcular la red de entrada y salida

    # Calculamos para la unidad 4
    I4 = (enter[3] * enter[0]) + (enter[5] * enter[1]) + (enter[7] * enter[2]) + enter[11]
    O4 = 1/(1 + numpy.exp(-I4))

    # Calculamos para la unidad 5
    I5 = (enter[4] * enter[0]) + (enter[6] * enter[1]) + (enter[8] * enter[2]) + enter[12]
    O5 = 1/(1 + numpy.exp(-I5))

    # Calculamos para la unidad 6
    I6 = (enter[9] * O4) + (enter[10] * O5) + enter[13]
    O6 = 1/(1 + numpy.exp(-I6))
```

Posteriormente se calculó el error de cada neurona, usando las variables `E4`, `E5`, y `E6` con los datos ya obtenidos previamente:

```
    #%% Calcular el error

    # Calculamos para la unidad 6
    E6 = O6 * (1 - O6) * (Tj - O6)

    # Calculamos para la unidad 5
    E5 = O5 * E6 * enter[10]

    # Calculamos para la unidad 4
    E4 = O4 * E6 * enter[9]

```

Y finalmente, con estos datos, se actualizaron todos los valores dentro del vector:

```
    #%% Actualizamos pesos y bias
    enter[10] = enter[10] + (l * E6 * O5)
    enter[9] = enter[9] + (l * E6 * O4)
    enter[8] = enter[8] + (l * E5 * enter[2])
    enter[7] = enter[7] + (l * E4 * enter[2])
    enter[6] = enter[6] + (l * E5 * enter[1])
    enter[5] = enter[5] + (l * E4 * enter[1])
    enter[4] = enter[4] + (l * E5 * enter[0])
    enter[3] = enter[3] + (l * E4 * enter[0])
    enter[13] = enter[13] + (l * E6)
    enter[12] = enter[12] + (l * E5)
    enter[11] = enter[11] + (l * E4)

    print(enter)
    print(O6)
```

El código comienza

Tras un rato de ejecución, se han arrojado los siguientes resultados:

```
[1, 0, 1, 2.334032710677582, 6.239430989188243, 0.4, 0.1,
1.6340327106797072, 6.739430989188151, 2.091914818973689,
3.629763302449862, 1.7340327106796236, 6.739430989188151,
3.7652538608912622]
0.9999236396335215
```

Donde entre corchetes tenemos los valores actualizados del vector y específicamente un renglón después se imprime el valor que la neurona 6 está sacando, que, aunque el dato está dentro del objeto anterior, se imprime nuevamente para que sea más fácil de visualizar en el apartado de prueba del algoritmo.

De igual manera, el código se encuentra anexo a este documento en el anexo 1, así como entregable de manera externa en la misma entrega.

Conclusiones

Espinoza Sánchez Joel Alejandro: Personalmente creo que la aplicación de este examen me permitió aplicar los conocimientos de las redes neuronales junto a mi equipo, para desarrollar de mejor manera el algoritmo y que éste funcione correctamente en un tiempo de desarrollo mejor. Asimismo, esto me permitió despejar algunas dudas que tenía al discutir el algoritmo con mi equipo y tener una mejor comprensión, no sólo creo que personalmente, sino mencionándolo por todo el equipo, gracias al trabajo colaborativo.

Gómez Garza Dariana: En la realización de este examen pudimos reforzar muchos conocimientos antes vistos en clases, tanto teóricos como prácticos y ahora añadiéndole la parte de programación. Sinceramente al hacer el examen quedó mucho más claro el tema de las multicapas y de cómo funciona una red neuronal con propagación hacia atrás y realmente nos ayudó mucho. Me siento contenta, y yo creo que mi equipo también con el trabajo que realizamos.

González Arenas Fernando Francisco: Las redes neuronales prealimentadas (Feed-Forward), son un tipo de red neuronal que se entrena con un dataset de datos inicial para que aprendan a realizar la tarea para la que fueron creadas, en esta red neuronal todo el proceso es hacia adelante y no utiliza ciclos entre neuronas como en otros tipos de redes. Estos modelos son muy interesantes, porque además de ser uno de los modelos más conocidos, son usados para muchas aplicaciones como el reconocimiento de patrones, aproximación de funciones, entre muchas más. En lo personal las redes neuronales, comenzando por las Feed-Forward, considero que

tienen mucho potencial para resolver multitud de problemas que se nos presentan cotidianamente a los seres humanos y que durante los próximos años o décadas seguiremos descubriendo muchísimas más aplicaciones de esta tecnología en diferentes áreas como la exploración espacial, medicina, automatización de procesos, etc.

Referencias

1. D. Michie, D.J. Spiegelhalter, C.C. Taylor (eds). Machine Learning, Neural and Statistical Classification, 1994.

2. R. Rojas. *Neural Networks: A Systematic Introduction,* Springer, 1996.ISBN 3-540-60505-3.

3. Auer, Peter; Harald Burgsteiner; Wolfgang Maass (2008). A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Networks* **21**: 786-795.

4. Juan Pedro Vásquez López. (2014). Red neuronal feedforward como estimador de patrones de corrientes en el interior del puerto de manzanillo sujeto a la acción de tsunamis. Febrero 2021, de IMT, Instituto Mexicano del Transporte

5. Damián Jorge Matich. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. Febrero 2021, de Universidad Tecnológica Nacional – Facultad Regional Rosario.

Anexos

Anexo 1: Código en Python 3 de la red neuronal.

```
#%% Presentación
'''
   Universidad Autónoma de Aguascalientes

         Centro de Ciencias Básicas
 Departamento de Ciencias de la Computación
          Aprendizaje Inteligente
                   6° "A"

                Primera Evaluación Parcial

   Profesor: Francisco Javier Luna Rosas
  Alumnos:
      Espinoza Sánchez Joel Alejandro
      Gómez Garza Dariana
      González Arenas Fernando Francisco

  Fecha de Entrega: 2 de marzo del 2021

Descripción: Código correspondiente a la primera evaluación parcial de una
red neuronal
'''

#%% Inicio del código y preparación de la red
import numpy

'''
Los valores de enter son:
    x1 x2 x3 w14 w15 w24 w25 w34 w35 w46 w56 t4 t5 t6
'''
```

```
enter = [1, 0, 1, 0.2, -0.3, 0.4, 0.1, -0.5, 0.2, -0.3, -0.2, -0.4, 0.2,
0-1]
Tj = 1
l = 0.9
O6 = 0

print(enter)
print(O6)

while (O6 < 1):
    #%% Calcular la red de entrada y salida

    # Calculamos para la unidad 4
    I4 = (enter[3] * enter[0]) + (enter[5] * enter[1]) + (enter[7] *
enter[2]) + enter[11]
    O4 = 1/(1 + numpy.exp(-I4))

    # Calculamos para la unidad 5
    I5 = (enter[4] * enter[0]) + (enter[6] * enter[1]) + (enter[8] *
enter[2]) + enter[12]
    O5 = 1/(1 + numpy.exp(-I5))

    # Calculamos para la unidad 6
    I6 = (enter[9] * O4) + (enter[10] * O5) + enter[13]
    O6 = 1/(1 + numpy.exp(-I6))

    #%% Calcular el error

    # Calculamos para la unidad 6
    E6 = O6 * (1 - O6) * (Tj - O6)

    # Calculamos para la unidad 5
```

```
E5 = O5 * E6 * enter[10]

# Calculamos para la unidad 4
E4 = O4 * E6 * enter[9]

#%% Actualizamos pesos y bias
enter[10] = enter[10] + (l * E6 * O5)
enter[9] = enter[9] + (l * E6 * O4)
enter[8] = enter[8] + (l * E5 * enter[2])
enter[7] = enter[7] + (l * E4 * enter[2])
enter[6] = enter[6] + (l * E5 * enter[1])
enter[5] = enter[5] + (l * E4 * enter[1])
enter[4] = enter[4] + (l * E5 * enter[0])
enter[3] = enter[3] + (l * E4 * enter[0])
enter[13] = enter[13] + (l * E6)
enter[12] = enter[12] + (l * E5)
enter[11] = enter[11] + (l * E4)

print(enter)
print(O6)
```