Universidad Tecnológica Centroamericana
Facultad de Ingeniería

CC414 - Sistemas Inteligentes
Docente: Kenny Dávila, PhD

Tarea #4 – Redes Neurales Artificiales (4% Puntos Oro)

El objetivo de esta tarea es la implementación de redes neural artificiales con alimentación hacia adelante (feed-forward). Todas las arquitecturas a probar son totalmente conectadas (fully connected network). También se requiere implementar el algoritmo de propagación hacia atrás (Back-propagation) para entrenar dichas redes.

Nota. La Tercera edición del libro de la clase tiene un error en el pseudo código del algoritmo de back-propagation. El error es que la inicialización aleatoria de los pesos aparece dentro del ciclo de entrenamiento (al inicio de cada época). Sin embargo, esta inicialización **debe realizarse solamente una vez al inicio**, antes de la primera época.

Es requerido que implemente desde 0 los algoritmos más básicos de alimentación hacia adelante, y de propagación hacia atrás. **No se permite el uso de librerías de redes neurales salvo para operaciones matemáticas básicas**. Se recomienda el uso de numpy y la vectorización de operaciones.

Parte 1. Feed-Forward (1.0%)

El objetivo de este punto es la implementación de la estructura básica de una red neural artificial con alimentación hacia adelante. Deberá implementar los diferentes componentes de una red neural usando Python o Java. Se recomienda implementar al menos 3 clases: una para representar la red completa, otra para representar una capa de la red (ya sea oculta o de salida), y una para representar cada neurona. Todas las arquitecturas son completamente conectadas y usan la misma función de activación en todas las neuronas de una misma capa, pero dos capas distintas podrían usar funciones de activación diferentes. Implementaciones **vectorizadas** pueden prescindir de una clase neurona y usar directamente una matriz para representar todos los pesos de una capa dada.

Su implementación debe ser capaz de ser inicializada con pesos aleatorios con valores entre -1 y 1. También debe tener funcionalidad que le permita **cargar y guardar sus pesos usando archivos en formato JSON**. Dicho archivo debe contener un diccionario con un campo “entradas”, indicando el número de entradas que acepta la red, y un campo “capas” que contenga una lista de las capas en la red (se asume que siempre habrá al menos una capa). Cada una de estas capas será descrita usando un diccionario con un campo “neuronas” que contiene la lista de neuronas en la capa. Cada neurona se representa usando un diccionario con un campo “pesos” que contiene una lista de los pesos para cada conexión de entrada que tiene la neurona. El **primer peso de cada lista** siempre representará el peso del bias. Se le provee un archivo de ejemplo que implementa este formato (“part1_red_prueba.json”).

El objetivo de este ejercicio es implementar y validar la alimentación hacia adelante. Para esto, debe implementar una **función** "**evaluar**" dentro de su **clase Red Neural** donde se recibe como entrada el vector X con los valores de entrada y esta función debe retornar el vector de las salidas producidas por la capa de salida de la red. Todas las neuronas deben usar la función sigmoide como función de activación.

Se le pide utilizar su implementación para recrear la arquitectura del ejemplo visto en clase (2 capas con 2 neuronas por cada capa). **En su informe debe reportar las salidas producidas por la red** para los siguientes vectores: <0, 0>, <0, 1>, <1, 0> y <1,1>. Debe reportar dos versiones:

1) Resultados usando pesos aleatorios (incluya los pesos en el reporte)
2) Resultados usando los pesos provistos en "part1_red_prueba.json". **Debe cargar los pesos desde el archivo, no se permite usar pesos "hard coded"**.

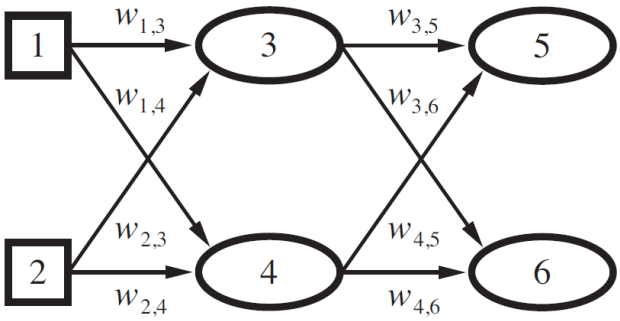

Figura 1. Arquitectura de red neural a Implementar. Los pesos del bias en cada neurona K ($w_{0,k}$) **no** están representados en esta imagen, pero son requeridos en su implementación.

Parte 2. Back-Propagation (1.5%)

En esta parte se le pide programar el algoritmo de propagación hacia atrás para entrenamiento de la red neural. Puede seguir el pseudo código que se provee en el libro de la clase. Solamente debe recordar que la inicialización de los pesos se hace solamente una vez al inicio y no dentro de cada ciclo de entrenamiento (época) como se muestra en el libro. El algoritmo de back-propagation se basa en épocas. Por cada época, la red es ajustada usando **todos los datos de entrenamiento** disponibles. En esta versión solamente usamos un ejemplo a la vez (no hay lotes). El algoritmo se detiene cuando se han realizado un número máximo de épocas o si se han hecho múltiples épocas continuas en las que el error en datos de validación no disminuye o incluso se incrementa. Al terminar el proceso de entrenamiento, **los pesos aprendidos deben guardarse en un archivo json** usando el formato especificado en la parte 1.

El algoritmo recibe como **entrada** los "datos de entrenamiento", "máximo número de épocas", "datos de validación", "valor épsilon" y "máximo de rondas sin decremento", donde solo los primeros dos parámetros son **mandatorios**. Al final de cada ronda de entrenamiento (época), el algoritmo debe **imprimir estadísticas** como el MSE entre los vectores de salida esperados (ground truth) y los vectores de salida de la red (output). El conjunto de datos de validación es un **parámetro opcional** y si se especifica entonces también se deben imprimir las mismas estadísticas para los datos de validación al finalizar cada

época. Note que es conveniente que se use un **formato de salida** que sea fácil de **tabular,** pero también es válido guardar directamente las estadísticas en un archivo csv para su posterior análisis. **Si se especifican** el valor épsilon y el máximo de rondas sin decremento, entonces deberá hacer un conteo al final de cada ronda donde verifique si la diferencia entre los valores MSE de validación de la ronda anterior y la ronda actual es menor al valor épsilon. Si se acumula un **total de épocas continuas** igual al valor "máximo de rondas sin decremento", entonces se detiene el proceso de entrenamiento.

Se le pide **entrenar 20 redes neurales usando la misma arquitectura simple de la parte 1** y la función suma binaria de dos entradas x1 y x2 usando los siguientes datos de entrenamiento en el archivo "part2_train_data.csv". Cada red neural deberá ser inicializada con **pesos aleatorios**. Cada red debe ser entrenada por un **total de 100 épocas**. Se le pide generar un gráfico de líneas con **3 líneas** mostrando lo siguiente: valor **máximo** de MSE por época, valor **mínimo** de MSE por época, valor **promedio** de MSE por época. **No hay datos de validación para este ejercicio**.

x_1	x_2	y_3 (carry)	y_4 (sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Datos de entrenamiento a usar

Parte 3. Clasificación (0.75%)

Una compañía encuestadora entrevisto mucha gente con el fin de colectar datos para ver si es posible determinar el tipo de comida favorita de una persona a partir de algunas preguntas básicas. Solamente se consideraron 5 platillos como posibles comidas favoritas: pizza, hamburguesa, arroz frito, ensalada y pollo horneado. A partir de las preguntas originales se generaron los siguientes puntajes que usaremos para la clasificación de los platillos favoritos:

1. Amor al queso. Da un puntaje a que tanto disfruta la persona aquellos platillos que contienen mucho queso.
2. Amor a la carne. Da un puntaje que determina que tanta carne consume la persona en su dieta.
3. Amor a los vegetales. Da un puntaje que determina que tantos vegetales consume la persona en su dieta.
4. Amor al arroz. Puntaje que determina que tanto arroz consume la persona en comparación a otros individuos.
5. Amor a lo frito. Puntaje que determina que tantas comidas fritas consume la persona a la semana.

Se le pide entrenar una serie de redes neurales con el fin de encontrar una arquitectura capaz de obtener buenos resultados en la clasificación. Es importante tomar en cuenta que las redes neurales son sensibles a los rangos de los valores de entrada. Por esta razón se recomienda usar normalización de los datos. Así mismo, se proveen las etiquetas como cadenas de texto. Es requerido usar codificación y convertir estos valores en vectores binarios ("one-hot encoding"), donde cada clase activa uno y solamente uno de los valores de salida para un total de 5 valores binarios. Al momento de realizar la clasificación usando la

salida de la red, se debe tomar la neurona con el valor de salida mas grande como la clase correspondiente.

Para cada arquitectura, se le pide entrenar la red usando los datos de entrenamiento provistos en el archivo “part3_data_train.csv”. Se deben entrenar la red por un **máximo** de 50 épocas. También se le proveen datos de validación en el archivo “part3_data_val.csv”, los cuales deberá combinar con un valor épsilon de 0.05 y un máximo de 3 épocas para determinar si el entrenamiento debe parar antes. Una vez completado el entrenamiento, entonces se le pide usar los datos de prueba en “part3_data_test.csv” para evaluar la red.

A diferencia de la parte 2, aquí estamos haciendo clasificación y por lo tanto se debe modificar el entrenamiento de la red para imprimir las siguientes métricas al final de cada época: accuracy (entrenamiento), promedio de F-1 por clase (entrenamiento), accuracy (validación), promedio de F-1 por clase (validación). Al final del entrenamiento, deberá imprimir las siguientes métricas para los datos de prueba: matriz de confusión, accuracy y promedio de F-1 por clase. Se le pide **resumir** las métricas sobre **datos de entrenamiento** y **validación** usando **gráficos de línea** en su **reporte**. **SE REDUCIRAN PUNTOS POR INCLUIR CAPTURAS DE PANTALLA**.

Se le pide probar las siguientes arquitecturas de red:

1. 5 entradas, una capa oculta con 4 neuronas, y una capa de salida con 5 neuronas.
2. 5 entradas, una capa oculta con 16 neuronas, y una capa de salida con 5 neuronas.
3. 5 entradas, una capa oculta con 32 neuronas, y una capa de salida con 5 neuronas.
4. 5 entradas, una capa oculta con 64 neuronas, y una capa de salida con 5 neuronas.
5. Una arquitectura definida por usted en base a los resultados anteriores.

Se le pide contestar las siguientes preguntas:

1. ¿Cuál de las 5 arquitecturas de red funciono mejor y por qué?
2. ¿En que se baso para crear su propia arquitectura de red neural?
3. ¿Cuál es la utilidad de los datos de validación?

Parte 4. Regresión (0.75%)

Se le pide modificar su implementación de redes neurales para poder soportar regresión. Por defecto, la función de activación sigmoide solamente puede producir valores entre 0 y 1. Sin embargo, al predecir cantidades más grandes, necesitamos que la red pueda producir un rango de valores menos limitado. Por esta razón, se debe crear una variación de neurona que utilice la función lineal como función de activación:

$$\boldsymbol{g(x) = x}$$

Donde la salida seria simplemente igual al valor de entrada. En este caso, esto seria el producto punto del vector de los valores de activación de la capa anterior por el vector de los pesos de la neurona actual. La derivada de esta función es simplemente:

$$\boldsymbol{g'(x) = 1}$$

Note que solamente usaríamos la función lineal en la capa de salida. Esto tendría un efecto al momento de calcular los valores delta del algoritmo de back-propagation, particularmente en las siguientes líneas del algoritmo:

$$\textbf{for each } \text{node } j \text{ in the output layer } \textbf{do}$$
$$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$$

Aplicación. Pokémon Go es un juego con múltiples fans alrededor del mundo. En dicho juego, cada Pokémon recibe un puntaje llamado Poder de Combate (CP en inglés) que indica lo fuerte que será en batalla. Dicho valor depende de una serie de factores incluyendo el nivel del Pokémon, las estadísticas base para su especie y sus valores individuales (IVs). Se le provee una base de datos ("part4_pokemon_go_train.csv") que contiene diferentes pokémones, con sus respectivas estadísticas de ataque, defensa, estamina, valores individuales y su respectivo valor de PC. Con estos datos, se le pide entrenar una serie de redes neurales para regresión, con el fin de producir una red que sea capaz de tomar las estadísticas de un Pokémon y prediga el respectivo PC del mismo. Se le provee una base de datos suplementaria ("part4_pokemon_go_validation.csv") que debe usar durante el entrenamiento para detener el entrenamiento si este deja de mejorar. Al terminar de entrenar cada red, debe evaluar el resultado usando la base de datos de prueba provista ("part4_pokemon_go_test.csv"). Las redes neurales no deben ser entrenadas por más de 50 épocas y debe usar criterios de detención temprana similares a los que se usaron en la Parte 3. En todos los casos es requerido ignorar el atributo nombre que solo se incluye con fines informativos. También podría resultar conveniente normalizar los datos de entrada.

Se le pide que entrene una serie de redes neurales. Para cada una de ellas, se le pide reportar resultados de entrenamiento y validación. En este caso la métrica a considerar sería el MSE entre la predicción del PC y los valores originales. Se le pide representar el resultado del entrenamiento de cada red usando un grafico de líneas. **SE REDUCIRAN PUNTOS POR INCLUIR CAPTURAS DE PANTALLA**. Al final también debe proveer el valor del MSE para los datos de prueba.

Se le pide probar las siguientes arquitecturas de red:

1. 7 entradas, **una capa oculta** con 16 neuronas y capa de salida con una sola neurona con activación lineal
2. 7 entradas, **una capa oculta** con 32 neuronas y capa de salida con una sola neurona con activación lineal
3. 7 entradas, **dos capas ocultas** con 16 neuronas cada una, y capa de salida con una sola neurona con activación lineal
4. 7 entradas, **dos capas ocultas** con 32 neuronas cada una, y capa de salida con una sola neurona con activación lineal
5. Una arquitectura definida por usted en base a los resultados anteriores.

Se le pide contestar las siguientes preguntas:

1. ¿Utilizó normalización de datos si o no, y por qué?
2. ¿Cuál de las 5 arquitecturas de red funciono mejor y por qué?
3. ¿En que se baso al momento de crear su propia arquitectura de red?

Fechas Importantes

1. Asignado: 5 de diciembre del 2020.
2. Entrega: 15 de diciembre del 2020.

Otras políticas

1. Esta tarea deberá trabajarse y entregarse **individual** o en **parejas.**
2. La entrega será **un solo archivo comprimido (.zip o .rar)**. Dentro de dicho archivo debe contener su respectivo reporte **en formato PDF**. También debe contener los scripts de Python que se usaron para contestar cada punto.
3. Si se hace en parejas, ambas personas deben subir el mismo archivo.
4. El **plagio** será penalizado de manera severa.
5. Los estudiantes que entreguen una tarea 100% original recibirán una nota parcial a pesar de errores existentes. En cambio, los estudiantes que presenten tareas que contenga material plagiado recibirán 0% automáticamente independientemente de la calidad.
6. Tareas entregadas después de la fecha indicada solamente podrán recibir la mitad de la calificación final. Por esta razón, es posible que **un trabajo incompleto pero entregado a tiempo termine recibiendo mejor calificación que uno completo entregado un minuto tarde**.