Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-5438 Inteligencia Artificial II
Trimestre Septiembre - Diciembre 2023

Informe Proyecto 2

Estudiantes:

Br. Gamboa, Roberto

Br. Blanco, Luis

Carnés:

16-10394

17-10066

Profesor

Carlos Infante

24 de noviembre de 2023

Índice

Introducción

El algoritmo de backpropagation es un método de aprendizaje automático supervisado que se utiliza para entrenar redes neuronales artificiales. Este algoritmo funciona calculando la deriva de la función de pérdida con respecto a los pesos de la red neuronal. Luego, utiliza esta información para actualizar los pesos de la red en la dirección que reduce la función de pérdida.

Dicho algoritmo es muy poderoso y puede ser utilizado para entrenar redes neuronales con diversos fines, incluyendo clasificación, regresión y reconocimiento de patrones. Sin embargo, también puede ser un algoritmo complejo de implementar correctamente.

El objetivo principal de este proyecto es la implementación de dicho algoritmo, para luego realizar el entrenamiento con un set de datos y estudiar su precisión.

Link al repositorio del proyecto

El proyecto se encuentra alojado en el siguiente enlace:

`https://github.com/rcgamboan/CI5438-Proyecto2`

Implementación

El algoritmo fue implementado en el lenguaje de programación Python. Se siguió el pseudocódigo dado en clases y se complementó con los códigos que se encuentran en:

- "6- Implementing a neural network from scratch in Python"
- "7- Training a neural network: Backward propagation and gradient descent"

Se utilizó la librería numpy para realizar los cálculos requeridos en el algoritmo de una forma cómoda y sencilla. Además para realizar las gráficas se usó matplotlib.

El algoritmo de Back Propagation cuenta con un criterio de convergencia que compara el resultado de cada iteración con el de la iteración anterior, deteniendo su ejecución si la diferencia entre ambos es mínima. En cada iteración del algoritmo se calcula el costo de la iteración y se guarda en un arreglo, que posteriormente es utilizado para representar gráficamente la evolución del costo.

Para la implementación de la red neurona, se crea un objeto de tipo `Network` el cual necesita los siguientes parámetros para ser instanciado:

- `n_inputs`: representa el número de neuronas en la capa de entrada
- `n_hidden`: representa el número de neuronas en la(s) capas ocultas. Este parámetro es representado con un arreglo, si el arreglo es vacío no se tienen capas ocultas. Cada posición del arreglo representa una capa oculta y el valor de la casilla representa el número de neuronas en esa capa.
- `n_output`: representa el número de neuronas en la capa de salida

Por ejemplo, para crear una red neuronal con 6 neuronas en la capa de entrada, 2 capas ocultas con 4 y 3 neuronas respectivamente y 1 neurona en la capa de salida, la creación del objeto de tipo `Network` sería de la siguiente forma:

```
nombre_red = Network(4, [6, 3], 1)
```

Al ejecutar el algoritmo de backpropagation se "propaga" el error de la iteración actual desde las capas de salidas hasta las capas iniciales, con el fin de minimizarlo en cada iteración hasta que sea mínimo.

Entrenamiento del modelo

Iris Dataset

El entrenamiento del modelo se puede encontrar en el archivo `src/clasificacion.py`. Primero se obtienen los datos a ser utilizados para entrenar el modelo y se dividen en dos lotes: un lote de entrenamiento y un lote de prueba, los cuales conforman en 80 % y 20 % respectivamente del conjunto total de datos, al separar los datos en ambos lotes, se mantiene la proporción de valores de cada especie en ambas separaciones. Para separar los datos se utiliza el método train_test_split de la librería sklearn. Luego se establecen la tasa de aprendizaje a ser utilizada, la cantidad de iteraciones máxima y la tolerancia permitida para hacer la llamada al algoritmo de entrenamiento de la red. Para entrenar la red se itera los datos de entrenamiento proporcionados, se "propagan" desde las neuronas de la capa inicial hasta las de la capa final y se calcula el error comparando el valor obtenido al final de la red con el valor verdadero de la data de entrenamiento. Posteriormente se calcula el error cuadrático medio o MSE, se guarda en un arreglo para ser graficado posteriormente y se llama al algoritmo de backpropagation para propagar el error y actualizar los pesos de la red.

Para realizar las pruebas y el entrenamiento de la red, se han escogido 4 tasas de aprendizaje junto con 3 configuraciones para redes con una capa oculta y 2 configuraciones para redes con dos capas ocultas. Las tasas y configuraciones de red se muestran a continuación:

- **Tasas de aprendizaje:**
  - 0.1
  - 0.05
  - 0.005

- 0.0005

- **Configuraciones de red:**

  - Con una sola capa oculta:
    - `Network(4,[1],1)`
    - `Network(4,[2],1)`
    - `Network(4,[3],1)`
  - Con dos capas ocultas:
    - `Network(4,[2,1],1)`
    - `Network(4,[1,1],1)`

Para cada una de las tasas de aprendizaje, se realizó el entrenamiento de cada una de las redes neuronales planteadas. Para escoger la cantidad de neuronas encontradas en las capas ocultas se realizó la consulta del siguiente artículo: Choosing number of Hidden Layers and number of hidden neurons in Neural Networks.

Clasificadores binarios

Se crea una red neuronal para cada una de las clasificaciones disponibles en la data. Al tratarse de clasificadores binarios, se crean redes con 4 neuronas en la capa de entrada, ya que los datos tienen 4 características y 1 neurona en la capa de salida ya que se quiere obtener un valor booleano de respuesta. Al realizar clasificaciones con las redes, se redondea el valor obtenido para que este sea 1 o 0 y poder comparar con la data de prueba, a la cual previamente se le realiza una conversión donde se sustituye el nombre de la especie por 1 o 0 según concuerde con la especie correspondiente a la red.

Para cada una de las redes se realiza el entrenamiento, para luego calcular lo siguiente:

- Error medio, máximo y mínimo.
- Cantidad de falsos positivos
- Cantidad de falsos negativos

Se realizaron los experimentos solicitados en el enunciado del proyecto, obteniendo los siguientes resultados:

Para el entrenamiento de cada una de las redes neuronales se fijaron varias tasas de aprendizaje y un máximo de 10.000 iteraciones y en la mayoría de las ejecuciones el algoritmo converge alrededor de las 4.000 a 5.000 iteraciones. Las gráficas obtenidas para cada una de las tasas de aprendizaje y distintas configuraciones de la red neuronal se pueden encontrar en el directorio `img/clasificadores_binarios` del repositorio, junto con la información obtenida en cada uno de los entrenamientos de la red.

En los resultados obtenidos se puede apreciar que en la mayoría de los casos el algoritmo de backpropagation permite minimizar el error en la red y realizar clasificaciones con una buena precisión, de por lo menos el 60 %. En las pruebas donde se observa en la gráfica que el algoritmo no logra minimizar el error, se pudiera aumentar el número de iteraciones del algoritmo y disminuir el valor de tolerancia para la convergencia fijado en 1e-6 para las pruebas.

Como se utiliza información con pocas características y una sola posible salida, no es necesario hacer uso de tasas de aprendizaje tan pequeñas, en cambio, cuando se manejan mayores volúmenes de información de entrada, se hace necesario disminuir la tasa de aprendizaje para obtener clasificaciones acertadas con la red como se muestra posteriormente.

Clasificadores multiclase

Se crea la red neuronal a utilizar para realizar la clasificación requerida. Se realiza el entrenamiento de la red, para luego calcular el error medio, máximo y mínimo.

Al tratarse de un clasificador multiclase, la red neuronal contará con 3 neuronas en la capa de salida, 1 por cada clase posible. Al realizar clasificaciones con la red, se elige el valor más alto entre los 3 valores de salida y se selecciona como clase a la especie correspondiente a ese valor.

Se realizaron los experimentos solicitados en el enunciado del proyecto, obteniendo los siguientes resultados:

Para el entrenamiento de cada una de las redes neuronales se fijaron varias tasas de aprendizaje y un máximo de 10.000 iteraciones y en la mayoría de las ejecuciones el algoritmo converge alrededor de las 4.000 a 5.000 iteraciones. Las gráficas obtenidas para cada una de las tasas de aprendizaje y distintas configuraciones de la red neuronal se pueden encontrar en el directorio `img/clasificadores_multiclase` del repositorio, junto con la información obtenida en cada uno de los entrenamientos de la red.

En los resultados obtenidos se puede apreciar que, de manera similar a los resultados de la sección anterior, en la mayoría de los casos el algoritmo de backpropagation logra minimizar el error en la red. En todas las pruebas donde se utilizan redes con mas de una capa oculta, no se logra minimizar el error de igual medida que en los casos donde solo se usa una capa oculta o ninguna, esto se puede deber a que mientras mas capas ocultas se tengan, el gradiente debe propagarse por mas capas haciéndose cada vez mas pequeño, y pudiendo llegar a volverse 0. Otra razón puede ser el sobreajuste en la red, ya que mientras mas capas ocultas se tengan la red tiene mayor capacidad de adaptarse a las características de los datos de entrenamiento y no a las características en general del problema que se busca resolver.

Clasificación de Spam

El entrenamiento del modelo se puede encontrar en el archivo `src/spam.py`. Primero se obtienen los datos a ser utilizados para entrenar el modelo y se dividen en dos lotes: un lote de entrenamiento y un lote de prueba, los cuales conforman en 80 % y 20 % respectivamente del conjunto total de datos. Luego se establecen la tasa de aprendizaje a ser utilizada y la cantidad de iteraciones máxima para hacer luego crear la red neuronal a ser utlizada y realizar el entrenamiento de la misma. La red neuronal creada tiene 57 neuronas en la capa de entrada, ya que la data tiene 57 características y 1 neurona en la capa de salida ya que se trata como un clasificador binario donde si el valor de salida es 1, se clasifica el correo como spam y 0 si no es spam.

Para el entrenamiento de cada una de las redes neuronales se fijaron varias tasas de aprendizaje y un máximo de 10.000 iteraciones para algunas tasas y 100.000 para la útlima tasa probada. Con las primeras tasas de aprendizaje, el algoritmo converge alrededor de las 1.500

iteraciones; para el caso de última tasa utilizada, el algoritmo converge entre las 6.000 y 11.000 iteraciones. Las gráficas obtenidas para cada una de las tasas de aprendizaje y distintas configuraciones de la red neuronal se pueden encontrar en el directorio `img/spam` del repositorio, junto con la información obtenida en cada uno de los entrenamientos de la red.

El entrenamiento de las redes neuronales se realizó de manera similar al entrenamiento de las redes para el Iris Dataset, pero en este caso se variaron las tasas de aprendizaje y configuración de las redes en base a los datos para el clasificador de spam, quedando de la siguiente forma:

- **Tasas de aprendizaje:**
  - 0.00005
  - 0.000005
  - 0.0000005
- **Configuraciones de red:**
  - Con una sola capa oculta:
    - `Network(57,[7],1)`
    - `Network(57,[8],1)`
    - `Network(57,[9],1)`
  - Con dos capas ocultas:
    - `Network(57,[10, 9, 8],1)`
    - `Network(57,[9, 8, 7],1)`
    - `Network(57,[8, 7, 6],1)`

Al tratarse de un conjunto de datos con una gran cantidad de características, se ajustó la tasa de aprendizaje para lograr que el algoritmo de backpropagation minimice el error en la red, donde las tasas de aprendizaje usadas son varias magnitudes menores a las empleadas en los anteriores clasificadores. Se puede apreciar en los resultados que la red obtenida luego del entrenamiento puede realizar predicciones con una precisión de por lo menos el 60 % en la mayoría de los casos, sin embargo, se observa que si la tasa de aprendizaje se hace muy pequeña, el algoritmo falla al minimizar el error y por tanto no se logran predicciones tan acertadas.

Ejecución del proyecto

El proyecto fue realizado en el lenguaje de programación Python, en su versión 3.11.2. Para ejecutar el proyecto inicialmente se requiere instalar las librerías necesarias, especificadas en el archivo `requirements.txt` ubicado en la carpeta raíz del repositorio del proyecto, las mismas pueden instalarse mediante el comando

```
$ pip install -r requirements.txt
```

Para el Iris Dataset basta con acceder al directorio `src` y ejecutar el archivo `clasificacion.py` con el comando

```
$ python3 clasificacion.py <b|m>
```

donde:

- b: indica si se quieren usar los clasificadores binarios.
- m: indica si se quiere usar el clasificador multiclase.

Para el clasificador de spam basta con acceder al directorio `src` y ejecutar el archivo `spam.py` con el comando

```
$ python3 spam.py
```

Conclusión

Las redes neuronales pueden ser una herramienta muy efectiva para realizar tareas de clasificación, siempre y cuando se cuente con un conjunto de datos de entrenamiento lo suficientemente grande y diverso para poder aprender las características del problema que se está tratando de resolver. Es de suma importancia a su vez evaluar la cantidad de capas ocultas y la cantidad de neuronas por cada capa, ya que es un factor importante que afecta al rendimiento de la red de gran manera.

El algoritmo de backpropagation junto con el algoritmo de descenso de gradiente es una excelente opción para entrenar redes neuronales. Sin embargo, es importante realizar pruebas con diversas tasas de aprendizaje para encontrar el valor óptimo ya que si se utilizan tasas de aprendizaje no acordes al problema, puede fallar el algoritmo de backpropagation.

Recomendaciones

Para mejorar aún más el rendimiento de las redes neuronales, se pueden realizar las siguientes recomendaciones:

- Utilizar un conjunto de datos de entrenamiento más grande y diverso
- Experimentar con redes con diversas cantidades de capas ocultas y neuronas por cada capa
- Utilizar técnicas de regularización para evitar el sobreajuste a los datos de entrenamiento

En general, las redes neuronales son una herramienta poderosa que puede ser utilizada para una amplia gama de aplicaciones. Sin embargo, es importante tener en cuenta los factores mencionados anteriormente para garantizar que las redes neuronales se utilicen de manera efectiva y den resultados confiables.