Clasificación de Trayectorias de Movimiento usando un Perceptrón: Explicación y Fundamentación

Benjamín Alonso Fernández Andrade
Universidad de La Frontera, Facultad de Ingeniería y Ciencias
Departamento de Ciencias de la Computación e Informática
b.fernandez07@ufromail.cl
https://github.com/Mizuhar4/PERCEPTRON-2-PYTHON-PROYECT.git
Temuco, Chile

***Abstract*—En este trabajo se detalla la implementación de un modelo de perceptrón para la clasificación de trayectorias de movimiento en tres categorías: lineal, circular y aleatoria. Se explican las funciones del código, se justifica el uso de la función sigmoidal y se fundamenta la elección del diseño de la capa de entrada. El modelo demuestra un buen desempeño en la clasificación de patrones de movimiento.**

I. Introducción

El análisis y clasificación de trayectorias de movimiento son relevantes en aplicaciones que van desde la robótica hasta el monitoreo de comportamientos en sistemas de vigilancia. El objetivo de este proyecto es desarrollar un perceptrón que clasifique correctamente trayectorias como lineales, circulares o aleatorias. Para lograrlo, se implementó un modelo con una capa de entrada de 20 neuronas, una capa oculta de 5 neuronas y una capa de salida con 3 neuronas.

II. Metodología

A. Arquitectura del Modelo

El perceptrón se diseñó con las siguientes características:

- **Capa de entrada**: 20 neuronas que representan datos aplanados de 10 pasos de tiempo con coordenadas (x, y).
- **Capa oculta**: 5 neuronas, lo cual balancea la capacidad de modelado y la eficiencia computacional.
- **Capa de salida**: 3 neuronas, correspondientes a las categorías de movimiento.

B. Fundamentación Adicional de la Función Sigmoidal

La función de activación sigmoidal, definida como:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{1}$$

fue seleccionada en este modelo por varias razones que se relacionan con sus propiedades y beneficios específicos para redes neuronales pequeñas y para la tarea de clasificación de trayectorias de movimiento.

- **Rango de Salida y Representación de Probabilidades:** La función sigmoidal transforma cualquier entrada en un valor en el rango (0, 1), lo cual facilita interpretar las salidas de las neuronas como probabilidades. Esto es particularmente útil en problemas de clasificación, ya que cada neurona de salida puede representar una clase específica, y su valor puede interpretarse como la probabilidad de que la entrada pertenezca a dicha clase.
- **Adecuación para Redes de Tamaño Reducido:** La simplicidad y suavidad de la función sigmoidal son especialmente beneficiosas para redes pequeñas, como el perceptrón en este estudio. Este modelo no requiere arquitecturas complejas ni funciones de activación avanzadas, ya que trabaja con patrones relativamente sencillos de movimiento. La salida continua de la sigmoide en el rango (0, 1) ayuda a capturar patrones básicos sin aumentar innecesariamente la complejidad del modelo.
- **Propiedades Derivativas y Eficiencia en la Retropropagación:** La derivada de la función sigmoidal, expresada como:

  $$\sigma'(x) = \sigma(x)(1 - \sigma(x)), \tag{2}$$

  es crucial para el proceso de aprendizaje mediante retropropagación. Esta propiedad permite un cálculo eficiente de los gradientes en el ajuste de los pesos, facilitando una convergencia estable y rápida en redes de tamaño reducido. Al tener una derivada fácilmente computable, la función sigmoidal contribuye a minimizar el costo computacional del entrenamiento, manteniendo la estabilidad en la actualización de los pesos.

En resumen, la función sigmoidal se adapta adecuadamente al modelo planteado debido a su capacidad para producir salidas probabilísticas interpretables, su simplicidad en redes pequeñas, y su eficiencia computacional en el proceso de retropropagación. Esto contribuye a que el modelo mantenga un balance entre precisión y simplicidad en la clasificación de trayectorias de movimiento.

C. Evaluación del Diseño de la Entrada

Se evaluaron dos alternativas:

- **Opción 1**: Modelar cada par (x, y) como una unidad de entrada y delegar el cálculo de interacciones a la capa oculta. Esto introduce complejidad en la determinación de pesos.
- **Opción 2 (Seleccionada)**: Representar los datos aplanados, usando una entrada de 20 neuronas (10 pares de coordenadas). Esta elección permite simplificar la arquitectura y mantener consistencia en el procesamiento de las características de entrada.

La opción seleccionada reduce la complejidad del modelo y mejora la capacidad de generalización sin introducir una carga computacional excesiva en la capa oculta.

III. Explicación del Código

```python
import numpy as np

entrada_dim = 20
oculta_dim = 5
salida_dim = 3

pesos_entrada_oculta = np.random.normal(0, 0.1, (entrada_dim, oculta_dim))
pesos_oculta_salida = np.random.normal(0, 0.1, (oculta_dim, salida_dim))
sesgo_oculta = np.zeros(oculta_dim)
sesgo_salida = np.zeros(salida_dim)

def activacion_sigmoide(x):
    return 1 / (1 + np.exp(-x))

def derivada_sigmoide(x):
    return x * (1 - x)

def softmax(x):
    exps = np.exp(x - np.max(x))
    return exps / np.sum(exps, axis=0)

def paso_hacia_adelante(entrada):
    activacion_oculta = activacion_sigmoide(np.dot(entrada, pesos_entrada_oculta) + sesgo_oculta)

    salida = softmax(np.dot(activacion_oculta, pesos_oculta_salida) + sesgo_salida)
    return salida
```

Listing . Ejemplo de implementación del perceptrón.

- `entrada_dim`: Representa la dimensión de la capa de entrada, correspondiente a 10 pares de coordenadas (x, y) aplanados, resultando en 20 valores.
- `oculta_dim`: Número de neuronas en la capa oculta. Se eligen 5 neuronas para capturar patrones complejos con una arquitectura de baja complejidad.
- `salida_dim`: Número de neuronas en la capa de salida, una por cada clase de movimiento: lineal, circular y aleatorio.

- `pesos_entrada_oculta` y `pesos_oculta_salida`: Inicializados con valores aleatorios, estos matrices de pesos permiten al modelo aprender patrones complejos. La inicialización aleatoria evita problemas de simetría.
- `sesgo_oculta` y `sesgo_salida`: Inicializados en cero, los sesgos ajustan las salidas de las neuronas para mejorar el rendimiento del modelo.

Funciones de activación y retropropagación:

- `activacion_sigmoide`: Mapea las entradas a un rango de 0 a 1, lo que es útil para interpretar las salidas como probabilidades.
- `derivada_sigmoide`: La derivada de la sigmoide facilita el cálculo de gradientes durante la retropropagación.

Función `softmax`: Convierte la salida en una distribución de probabilidad, útil para clasificación multiclase.

Función de propagación hacia adelante:

- `paso_hacia_adelante(entrada)`: Esta función realiza el cálculo de la salida del perceptrón a partir de una entrada. Realiza los siguientes pasos:
  1) Calcula la activación de la capa oculta aplicando la función de activación sigmoidal a la multiplicación de la entrada con los pesos de la capa de entrada (`pesos_entrada_oculta`) y suma el sesgo de la capa oculta (`sesgo_oculta`). Esto permite al modelo capturar patrones no lineales en los datos de entrada.
  2) Calcula la salida aplicando la función softmax a la multiplicación de la activación de la capa oculta con los pesos de la capa de salida (`pesos_oculta_salida`) y suma el sesgo de la capa de salida (`sesgo_salida`). La función softmax transforma las salidas en probabilidades, facilitando la interpretación en problemas de clasificación.

IV. Resultados y Conclusión

El modelo fue evaluado con trayectorias generadas aleatoriamente, obteniendo una precisión promedio que demuestra la capacidad del perceptrón para identificar patrones de movimiento. La elección de la función sigmoidal y el diseño de entrada se justificaron al balancear la simplicidad y el rendimiento.

REFERENCES

[1] A. Mohanty, "Introduction to Activation Functions in Neural Networks," 2023. [En línea]. Disponible: https://www.datacamp.com/es/tutorial/introduction-to-activation-functions-in-neural-networks. [Accedido: 6 de noviembre de 2024].

[2] "Perceptrón: Qué es y para qué sirve," 2023. [En línea]. Disponible: https://datascientest.com/es/perceptron-que-es-y-para-que-sirve. [Accedido: 6 de noviembre de 2024].

[3] J. Francisco, "Redes Neuronales en Python - Perceptrón," 15 de marzo de 2022. [En línea]. Disponible: https://www.youtube.com/watch?v=0cyqjutbo9E. [Accedido: 6 de noviembre de 2024].

[4] "¿Qué son los pesos en las redes neuronales?" [En línea]. Disponible: https://es.quora.com/Qu%C3%A9-son-los-pesos-en-la-redes-neuronales. [Accedido: 6 de noviembre de 2024].

[5] Lluis Pelegrin, "Programando un clasificador Perceptrón en Python," 19 de abril de 2017. [En línea]. Disponible: https://www.llipe.com/2017/04/19/programando-un-clasificador-perceptron-en-python/. [Accedido: 6 de noviembre de 2024].