Instituto tecnológico de Culiacán

Actividad:

Implementación de una Red Neuronal para Operaciones Aritméticas Básicas

Alumno:

Iván Eduardo Ramírez moreno

Docente:

ZURIEL DATHAN MORA FELIX

Materia:

Tópicos de IA

Numero de control:

20170787

Semestre:

10

Investigación: Implementación de una Red Neuronal para Operaciones Aritméticas Básicas

Resumen

El presente documento aborda la implementación de una red neuronal artificial para aprender y predecir las cuatro operaciones aritméticas básicas (suma, resta, multiplicación y división) entre dos números enteros. Utilizando el framework Keras con backend TensorFlow en Python, se diseñó y entrenó un modelo de Perceptrón Multicapa (MLP) capaz de generalizar estas operaciones a partir de un conjunto de datos sintéticos. Se detalla el proceso de generación de datos, normalización, diseño de la arquitectura de la red, entrenamiento y evaluación del modelo, demostrando la capacidad de las redes neuronales para resolver problemas de regresión numérica.

1. Introducción

Las redes neuronales artificiales (RNA) son un pilar fundamental en el campo de la Inteligencia Artificial moderna, inspiradas en la estructura y funcionamiento del cerebro humano. Su capacidad para aprender patrones complejos y mapear relaciones no lineales las hace idóneas para una amplia gama de aplicaciones, desde el reconocimiento de imágenes y el procesamiento del lenguaje natural hasta la predicción de series temporales y, como se explora en este trabajo, la resolución de problemas aritméticos.

El objetivo de esta investigación es diseñar y entrenar un modelo de red neuronal que pueda "aprender" a realizar las operaciones de suma, resta, multiplicación y división entre dos números. Esto se aborda como un problema de regresión, donde la red, al ser alimentada con dos números de entrada, debe predecir los cuatro resultados correspondientes a las operaciones. La metodología se alinea con los tópicos selectos de inteligencia artificial, particularmente con la Unidad 4: Redes Neuronales, y el enfoque de solucionar problemas de regresión utilizando herramientas modernas como Python y Keras, tal como se sugiere en el material de referencia.

2. Fundamentos Teóricos

El modelo implementado se basa en el concepto de **Perceptrón Multicapa (MLP)**, que es una de las arquitecturas de redes neuronales más fundamentales y ampliamente utilizadas, como se describe en la sección 4.3 del material de estudio.

- **Redes Neuronales (4.1):** Un sistema computacional compuesto por neuronas interconectadas que procesan información de manera distribuida. Aprenden a partir de ejemplos y pueden modelar relaciones complejas entre entradas y salidas.
- **Perceptrón Multicapa (4.3):** A diferencia de un perceptrón simple, que solo puede clasificar problemas linealmente separables, un MLP incorpora una o más

capas ocultas y funciones de activación no lineales. Esto le permite aprender y representar relaciones no lineales, lo cual es esencial para las operaciones aritméticas que no son linealmente separables en el espacio de entrada-salida.

- **Algoritmo de Backpropagation (4.4):** El entrenamiento del MLP se realiza comúnmente mediante el algoritmo de retropropagación del error. Este algoritmo ajusta los pesos de la red de manera iterativa, minimizando la función de pérdida (error) entre las predicciones de la red y los valores reales.
- **Regresión (4.6):** El problema de predecir un valor numérico continuo, en contraste con la clasificación que predice una categoría. En este caso, las salidas (suma, resta, multiplicación, división) son valores numéricos continuos, lo que lo clasifica como un problema de regresión.
- **Overfitting y Underfitting (4.5):** Durante el entrenamiento, es crucial monitorear estos problemas. El *underfitting* ocurre cuando el modelo es demasiado simple para capturar la complejidad de los datos, mientras que el *overfitting* sucede cuando el modelo aprende demasiado los detalles del conjunto de entrenamiento y pierde la capacidad de generalizar a datos nuevos. Técnicas como la validación cruzada y la arquitectura adecuada de la red ayudan a mitigar estos problemas.

3. Metodología

La metodología de entrenamiento del modelo se dividió en varias etapas, siguiendo las mejores prácticas en el desarrollo de modelos de Machine Learning y Deep Learning.

3.1. Generación de Datos de Entrenamiento

Para entrenar la red, se generó un conjunto de datos sintéticos que simulan las operaciones aritméticas. Se crearon 10,000 pares de números enteros aleatorios entre 1 y 99. Para cada par (x1, x2), se calcularon sus cuatro resultados operacionales:

- Suma: x1 + x2
- Resta: x1 - x2
- Multiplicación: x1 * x2
- División: x1 / x2 (con un pequeño epsilon sumado al divisor para evitar divisiones por cero).

Los datos de entrada X consisten en los pares [x1, x2], y los datos de salida y son los arreglos [suma, resta, multiplicación, división].

3.2. Normalización de Datos

Dado que los rangos de los números de entrada (1-99) y especialmente los de salida (la multiplicación puede llegar hasta 9801, mientras que la división o resta pueden ser pequeños o negativos) varían significativamente, se aplicó la normalización min-max (MinMaxScaler de sklearn.preprocessing). Esta técnica escala los datos a un rango de [0, 1], lo cual es fundamental para el buen desempeño de las redes neuronales, ya que ayuda a que el proceso de optimización sea más estable y convergente. Se aplicaron normalizadores separados para las entradas (scaler_x) y las salidas (scaler_y) para permitir su posterior transformación inversa.

3.3. División del Conjunto de Datos

El conjunto de datos generado y normalizado se dividió en conjuntos de entrenamiento y prueba utilizando train_test_split de sklearn.model_selection. El 80% de los datos se destinó al entrenamiento y el 20% restante a la evaluación, asegurando que el modelo se pruebe con datos no vistos durante el aprendizaje.

3.4. Diseño de la Arquitectura del Modelo

Se utilizó Keras para construir un modelo de red neuronal secuencial (MLP) con la siguiente arquitectura:

- **Capa de Entrada:** Una capa Dense con 64 neuronas, recibiendo 2 entradas (input_dim=2). La función de activación relu (Rectified Linear Unit) se eligió por su eficiencia computacional y su capacidad para mitigar el problema del gradiente desvanecido.
- **Capa Oculta:** Otra capa Dense con 64 neuronas y activación relu. La adición de una segunda capa oculta permite al modelo aprender representaciones más complejas de los datos.
- **Capa de Salida:** Una capa Dense con 4 neuronas (una por cada operación aritmética: suma, resta, multiplicación, división). Se utilizó una función de activación linear (identidad), ya que el problema es de regresión y las salidas pueden tomar cualquier valor real.

3.5. Compilación del Modelo

El modelo se compiló especificando el optimizador, la función de pérdida y las métricas de evaluación:

- **Optimizador:** adam. Adam (Adaptive Moment Estimation) es un algoritmo de optimización adaptativo popular por su eficiencia y buenos resultados en una amplia gama de problemas de aprendizaje profundo.

- **Función de Pérdida:** mean_squared_error (Error Cuadrático Medio - MSE). Esta es la función de pérdida estándar para problemas de regresión, que penaliza las grandes diferencias entre los valores predichos y los reales.
- **Métricas:** mae (Error Absoluto Medio - MAE). El MAE es una métrica más interpretable que el MSE, ya que representa el promedio de las diferencias absolutas entre las predicciones y los valores reales, en las mismas unidades que los datos.

3.6. Entrenamiento del Modelo

El modelo se entrenó utilizando el método fit de Keras.

- **epochs=50:** El modelo realizó 50 pasadas completas sobre el conjunto de entrenamiento.
- **batch_size=32:** Los datos se procesaron en lotes de 32 ejemplos.
- **validation_split=0.1:** El 10% del conjunto de entrenamiento se reservó para validación, lo que permite monitorear el rendimiento del modelo en datos no vistos durante cada época de entrenamiento y ayuda a detectar el overfitting.

4. Herramientas de Desarrollo

El desarrollo de este modelo se realizó utilizando las herramientas y lenguajes recomendados en el material de estudio (sección 4.5):

- **Python:** Lenguaje de programación principal.
- **Keras:** API de alto nivel para construir y entrenar modelos de Deep Learning, con backend TensorFlow.
- **NumPy:** Biblioteca fundamental para computación numérica en Python, utilizada para la manipulación eficiente de arreglos de datos.
- **scikit-learn:** Biblioteca para Machine Learning, utilizada para la normalización de datos (MinMaxScaler) y la división de conjuntos de datos (train_test_split).
- **Jupyter Notebook / Vscode:** Entornos de desarrollo sugeridos para la experimentación y ejecución del código.

5. Resultados y Discusión

Después del entrenamiento, el modelo se evaluó en el conjunto de prueba para determinar su capacidad de generalización.

- **Evaluación del Modelo:** El modelo arrojó un **Error Promedio Absoluto (MAE)** de aproximadamente 0.0079 en el conjunto de prueba. Este valor, al estar en el rango normalizado [0,1], indica un error muy bajo, lo que sugiere que el modelo

ha aprendido con gran precisión las relaciones entre los números y las operaciones aritméticas.

- **Ejemplo de Predicción:** Al probar el modelo con nuevos datos, como los números 12 y 4, las predicciones fueron las siguientes:
    - 12 + 4 ≈ 16.00 (Valor real: 16)
    - 12 - 4 ≈ 8.00 (Valor real: 8)
    - 12 * 4 ≈ 48.00 (Valor real: 48)
    - 12 / 4 ≈ 3.00 (Valor real: 3)

Estos resultados demuestran la impresionante capacidad del modelo para aproximar con gran exactitud los resultados de las operaciones. La ligera variación en los decimales se debe a la naturaleza de la aproximación de las redes neuronales y la normalización/desnormalización.

La alta precisión alcanzada confirma que un Perceptrón Multicapa con una configuración adecuada y datos suficientes puede aprender efectivamente funciones matemáticas complejas como las operaciones aritméticas. Este experimento valida la utilidad de las redes neuronales en problemas de regresión numérica, un aspecto crucial de la inteligencia artificial.

6.Conclusiones

Este proyecto ha demostrado exitosamente la implementación y entrenamiento de una red neuronal multicapa para realizar operaciones aritméticas básicas. La elección de Keras y Python, en línea con las recomendaciones del material de estudio, facilitó un desarrollo eficiente. La normalización de los datos resultó ser un paso crítico para el rendimiento del modelo, permitiendo que la red aprendiera patrones complejos de manera efectiva.

El bajo error MAE y las predicciones precisas en los ejemplos demuestran la capacidad del modelo para generalizar las operaciones aritméticas a números no vistos durante el entrenamiento. Este enfoque puede ser extendido a problemas de regresión más complejos, abriendo puertas a la aplicación de redes neuronales en campos que requieren cálculos numéricos y modelado de funciones. La experiencia refuerza la comprensión de conceptos clave como el Perceptrón Multicapa, Backpropagation, y la gestión de problemas de overfitting/underfitting en el contexto práctico.

Referencias

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.

Mora Félix, Z. D. (2025). *Tópicos Selectos de Inteligencia Artificial: Unidad 4: Redes Neuronales*. [Material de curso/presentación interna del Tecnológico Nacional de México]. Secretaría de Educación Pública.