Tarea N°3

Redes Neuronales Convolucionales

Cristian Herrera Badilla

COM4402 – Introducción a Inteligencia Artificial

Escuela de Ingeniería, Universidad de O'Higgins

Sábado 11 de Noviembre 2023

Abstract*— En este informe, se exploraron dos arquitecturas de redes neuronales, net_1 y net_2, para la clasificación de imágenes en los conjuntos de datos MNIST y CIFAR-10. Los modelos demostraron un sólido rendimiento en MNIST, superando el 90% de precisión, indicando su eficacia en la clasificación de dígitos escritos a mano. Sin embargo, en CIFAR-10, la precisión fue más desafiante, situándose en el rango del 60-65%, revelando la complejidad de clasificar imágenes más variadas y detalladas. La comparación entre net_1 y net_2 sugiere que la primera puede tener una ligera ventaja en CIFAR-10. Además, la experimentación con hiperparámetros destacó la influencia del tamaño de lote y la importancia de ajustar la arquitectura para obtener un rendimiento óptimo en conjuntos de datos específicos. Este análisis contribuye a la comprensión de las mejores prácticas para diseñar modelos de aprendizaje automático en tareas de clasificación de imágenes.

I. Introducción

Este informe aborda el análisis y desarrollo de modelos de aprendizaje automático, centrándose en la clasificación de imágenes utilizando el conjunto de datos MNIST y CIFAR-10. Se explora la eficacia de dos arquitecturas de redes neuronales, net_1 y net_2, evaluando su rendimiento en la tarea de reconocimiento de dígitos escritos a mano y en la clasificación de imágenes más complejas de CIFAR-10. A través de la experimentación con hiperparámetros y la comparación detallada de resultados, se busca proporcionar una comprensión exhaustiva de cómo diferentes configuraciones afectan el rendimiento de las redes neuronales en estas tareas específicas. Este análisis contribuye a la comprensión de las mejores prácticas para el diseño y ajuste de modelos de aprendizaje automático en contextos de clasificación de imágenes.

II. Marco teórico

Se presentan conceptos clave importantes para este informe relacionados con el aprendizaje automático, la clasificación de imágenes, y sus definiciones.

A. El Dataset MNIST

La base de datos MNIST (Modified National Institute of Standards and Technology database) es un problema de clasificación multiclase en el que se nos pide que clasifiquemos un dígito (0−9) a partir de una imagen en escala de grises de 28×28:

La base de datos MNIST de dígitos escritos a mano, disponible en esta página, tiene un conjunto de entrenamiento de 60.000 ejemplos y un conjunto de prueba de 10.000 ejemplos. Es un subconjunto de un conjunto más amplio disponible en NIST. Los dígitos han sido normalizados en tamaño y centrados en una imagen de tamaño fijo.

Es una buena base de datos para las personas que quieren probar técnicas de aprendizaje y métodos de reconocimiento de patrones en datos del mundo real, dedicando un esfuerzo mínimo al preprocesamiento y al formateo.

B. Redes Neuronales Artificiales (RNAs):

Las redes neuronales artificiales son modelos computacionales inspirados en la estructura y el funcionamiento de las redes neuronales biológicas. Están compuestas por capas de unidades de procesamiento llamadas neuronas o nodos. Cada neurona realiza una combinación lineal de las entradas, seguida de una función de activación no lineal. Las RNAs se utilizan para aproximación de funciones y tareas de aprendizaje automático, como la clasificación de imágenes.

C. Redes Neuronales Profundas:

Las redes neuronales profundas, comúnmente denominadas redes neuronales profundas o Deep Learning, son una clase de redes neuronales con múltiples capas ocultas. La profundidad de la red permite aprender representaciones jerárquicas de datos, lo que es especialmente beneficioso para tareas de visión por computadora, como la clasificación de imágenes.

Componentes de una Red Neuronal:

I. Neuronas (Nodos): Las neuronas son unidades de procesamiento que realizan operaciones matemáticas en las entradas. Cada neurona tiene pesos asociados que controlan la influencia de las entradas en la salida. La suma ponderada de las entradas se pasa a través de una función de activación.

II. Capas Ocultas: Las capas ocultas son capas intermedias entre la capa de entrada y la capa de salida. Cada capa oculta consta de múltiples neuronas. La combinación de múltiples capas ocultas permite a la red neuronal aprender representaciones jerárquicas de los datos.

III. Funciones de Activación: Las funciones de activación introducen no linealidad en la red. Comunes funciones de activación incluyen ReLU (Rectified Linear Unit), Sigmoide y Tangente Hiperbólica (Tanh). Estas funciones permiten que la red aprenda relaciones no lineales en los datos.

IV. Pesos y Sesgos: Cada conexión entre neuronas tiene un peso que controla la fuerza de la conexión. Los sesgos (biases) se utilizan para ajustar la salida de una neurona. Durante el entrenamiento, los pesos y los sesgos se ajustan para minimizar la función de pérdida.

D. Convolutional Neural Networks):

Las redes neuronales convolucionales (CNN) son una clase de red neuronal profunda, aplicada con mayor frecuencia al análisis de imágenes visuales. También se conocen como redes neuronales artificiales invariantes al desplazamiento o al espacio, basadas en la arquitectura de pesos compartidos de los filtros de convolución que se deslizan a lo largo de las características de entrada, y proporcionan respuestas equivalentes a la traslación conocidas como feature maps. Las CNN tienen aplicaciones en el reconocimiento de imágenes y videos, sistemas de recomendación, clasificación de imágenes, segmentación de imágenes, análisis de imágenes médicas, procesamiento del lenguaje natural, interfaces cerebro-ordenador, y series temporales financieras, entre otras.

E. Conjuntos de Datos:

En este proyecto, se trabajó con tres conjuntos de datos:

Conjunto de Entrenamiento: Contiene imágenes de dígitos escritos a mano utilizados para entrenar los modelos.

Conjunto de Validación: Utilizado para ajustar hiperparámetros y evaluar el rendimiento del modelo durante el entrenamiento.

Conjunto de Prueba: Se utiliza para evaluar el rendimiento final del modelo en datos no vistos.

F. One-Hot Encoding

Es una técnica utilizada en el procesamiento de datos y en el aprendizaje automático para representar categorías o etiquetas categóricas como vectores binarios. Se utiliza comúnmente cuando se trabaja con algoritmos que requieren variables numéricas como entrada y no pueden manejar directamente etiquetas categóricas.

En el proceso de One-Hot Encoding, cada categoría única se representa mediante un vector binario con todos sus elementos a cero, excepto uno que corresponde a la posición de esa categoría en la lista de todas las categorías únicas. Esto crea una

representación "caliente" (con un "1") para la categoría específica y "fría" (con "0") para todas las demás.

La fórmula para realizar One-Hot Encoding de una variable categórica con n categorías únicas es la siguiente:

$$One - Hot\ Encoding(x_i) = [0, 0, \dots, 1, \dots, 0]$$

Donde x_i es la categoría específica que se está codificando y la posición del "1" en el vector corresponde al índice de esa categoría en la lista ordenada de todas las categorías únicas.

Esta representación facilita el manejo de variables categóricas en algoritmos de aprendizaje automático, ya que convierte la información categórica en un formato numérico que puede ser procesado más fácilmente.

G. Funciones de Pérdida:

En este proyecto, se utilizó la función de pérdida de entropía cruzada (Cross-Entropy Loss). Esta función de pérdida mide la discrepancia entre las predicciones del modelo y las etiquetas verdaderas del conjunto de entrenamiento.

H. Optimización:

Para entrenar modelos de redes neuronales, se utilizan algoritmos de optimización. En este caso, se empleó el optimizador Adam. El optimizador ajusta los pesos del modelo para minimizar la función de pérdida.

I. Precisión (Accuracy):

La precisión es una métrica fundamental utilizada para evaluar el rendimiento de un modelo de clasificación. Se define como la proporción de predicciones correctas realizadas por el modelo en relación con el número total de muestras en el conjunto de datos de prueba. La precisión se expresa como un porcentaje y varía entre 0% y 100%. Una alta precisión indica un rendimiento sólido en la clasificación, mientras que una baja precisión sugiere la necesidad de mejoras en el modelo.

J. Convolucion:

Operación matemática que combina dos funciones para producir una tercera. En el caso de imágenes, se aplica un filtro (también llamado kernel) a la imagen de entrada para obtener un mapa de características. Este filtro es una pequeña matriz que se desliza a lo largo de la imagen, multiplicando sus valores con los píxeles correspondientes y sumando los productos para obtener un valor en el mapa de características. Este proceso se repite para cada posición del filtro en la imagen.

K. Sliding Windows:

Se refiere a la práctica de aplicar la convolución mediante el desplazamiento de un pequeño filtro a lo largo de la imagen de entrada. En cada paso, se calcula la convolución en la región actual y se forma el mapa de características.

Este proceso se repite en toda la imagen para cubrir todas las ubicaciones posibles. La "ventana" es el área de la imagen que está siendo examinada en cada paso.

El uso de ventanas deslizantes en CNNs permite a la red aprender patrones locales y su disposición espacial en las imágenes, lo que es fundamental para el reconocimiento de objetos y la extracción de características relevantes.

III. METODOLOGÍA

1. Adquisición y preprocesamiento de datos

Para llevar a cabo el análisis y desarrollo de un modelo de aprendizaje automático, se inicia con la adquisición de los datos de entrada, los cuales consisten en un conjunto correspondientes a la base de datos de manuscritos MNIST y la base de datos de imágenes pequeñas CIFAR-10. Se utilizan dos conjuntos de datos: uno para entrenamiento y validación, y otro para pruebas.

2. Implementar una Función de Normalización de Imágenes

La normalización es un paso importante del preprocesamiento de datos, se sabe que las imágenes que se utilizaran son de tamaño 8 bits de [0,255], entonces el objetivo es que este intervalo sea [0,1], ya que de esta forma se mantendrá una escala consistente en los datos de entrada y también al reducir la magnitud de los valores de pixeles logra hacer que el proceso de optimización converja rápidamente.

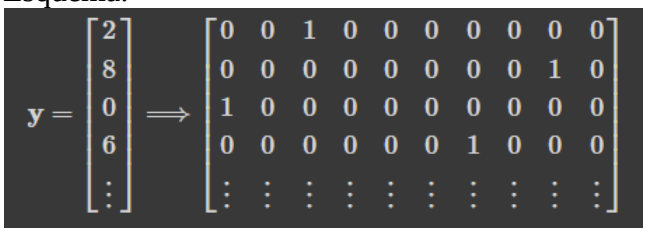
3. Ampliar la dimensión de entrada

En el conjunto de datos MNIST, cada dígito está representado por una matriz de tamaño (28,28). La red neuronal artificial que se va a construir utiliza el concepto de canales de color y mapas de características incluso para las imágenes en escala de grises. Esto significa que se tiene que transformar (28,28) en (28,28,1) utilizando la función: "np.expand_dims".

4. Implementación de función One-Hot Encoding

One-hot encoding es una solución robusta y sencilla para representar objetivos multicategóricos.

Esquema:



Ejemplo de One-hot Encoding

En el lado izquierdo tenemos un vector de etiquetas objetivo y con K=10 número de clases. En el lado derecho podemos ver la versión codificada one-hot del vector y donde cada elemento ∈y se ha transformado en un vector de filas K -dimensional. Sólo el elemento yi se ha fijado en 1, el resto son 0. Por ejemplo, el primer elemento de y es 2, lo que significa que el vector codificado en un punto será todo ceros excepto la posición 2 (indexación 0). Del mismo modo, el tercer elemento de y es 0, lo que significa que el vector codificado one-hot será todos los ceros a excepción de la posición 0.

La idea central es transformar los datos multicategóricos en una combinación de varias clases simples. Al hacer esto podemos, para cada ejemplo, ver si pertenece a alguna clase, donde 1 indica que sí y 0 en caso contrario.

Esta función crea una matriz de ceros de tamaño vector * number_classes. Luego utilizando un bucle for que recorrerá todas las columnas de dicha matriz e ira asignando el valor "1" en la posición correspondiente según los valores del vector. Finalmente se retorna la matriz con la función "np.array".

5. Implementación de una CNN

Importación de Librerías:

Se importa la biblioteca Keras y sus componentes necesarios. Keras es una interfaz de alto nivel para construir y entrenar modelos de aprendizaje profundo.

Definición de la Función net_1:

Se define una función llamada net_1 que toma dos parámetros: sample_shape (forma de la muestra, es decir, las dimensiones de la entrada) y nb_classes (número de clases para la tarea de clasificación).

Definición de la Entrada de la Red:

Se define la entrada de la red con las dimensiones especificadas por sample_shape.

Se construye la red neuronal con varias capas:

Dos capas de convolución con activación ReLU y padding 'same'.

Una capa de max-pooling para reducir las dimensiones del feature map.

Una capa de aplanado para convertir el feature map en un vector plano.

Una capa totalmente conectada (fully-connected) con activación ReLU.

Una capa completamente conectada de salida con activación softmax para la clasificación.

Definición del Modelo:

Se define el modelo utilizando la entrada (input_x) y la salida (probabilities) que se han especificado en las capas anteriores.

Retorno del Modelo:
Finalmente, la función devuelve el modelo construido.

6. Implementación de una CNN sin max-pooling

Definición de la Función net_2:
Similar a la función net_1, la función net_2 toma dos parámetros: sample_shape (forma de la muestra, es decir, las dimensiones de la entrada) y nb_classes (número de clases para la tarea de clasificación).

Definición de la Entrada de la Red:
Al igual que en net_1, se define la entrada de la red con las dimensiones especificadas por sample_shape.

Construcción de la Red:
Se construye la red neuronal con las siguientes capas:
Una capa de convolución con 32 filtros y activación ReLU.
Otra capa de convolución con 64 filtros, activación ReLU, padding 'same', y un stride de (2, 2), lo que resulta en una reducción de tamaño debido al stride.
Luego, se aplica la operación de aplanado para convertir el feature map en un vector plano.
A continuación, hay una capa totalmente conectada (fully-connected) con 128 dimensiones y activación ReLU.
Finalmente, hay una capa completamente conectada de salida con activación softmax para la clasificación.

Definición del Modelo:
Similar a net_1, se define el modelo utilizando la entrada y salida especificadas en las capas anteriores.

Retorno del Modelo:
Al igual que en net_1, la función devuelve el modelo construido.

7. Definición de los hiperparametros y entrenamiento de los modelos

Utilizando prueba y error, se definen los hiperparametros para el entrenamiento de los modelos con el objetivo de maximizar su “accuracy” y minimizar su “loss”.
Hiperparamentros utilizados:

Para net_1():
batch_size = 600
epochs = 50

batch_size = 1000
epochs = 50

batch_size = 300
epochs = 50

batch_size = 100
epochs = 50

batch_size = 50
epochs = 50

batch_size = 50
epochs = 25

batch_size = 50
epochs = 75

batch_size = 100
epochs = 100

Para net_2():
batch_size = 600
epochs = 50

batch_size = 300
epochs = 50

batch_size = 100
epochs = 50

batch_size = 100
epochs = 25

batch_size = 100
epochs = 75

batch_size = 200
epochs = 75

batch_size = 100
epochs = 100

8. Implementar una CNN para CIFRAR10

Utilizando los recursos antes creados y mencionados, se implementa una tercera CNN para los datos CIFRAR10:

a) Codificación One-Hot para los labels
b) Normalizar las imágenes
c) Crear una CNN basada en "net_1()" y otra CNN basada en "net_2()"

9. Entrenamiento de los nuevos CNN

Los modelos anteriores se entrenan utilizando "epochs = 30", "batch_size = 128" y "validation_split = 0.2"

```
history = model.fit(x_train, y_train,
                    epochs=30,
                    batch_size=128,
                    validation_split=0.2)
```

IV. Resultados

Tabla de resultados:

net_1	loss	accuracy
batch_size = 600 epochs = 50	0.3594	0.9021
batch_size = 1000 epochs = 50	0.5023	0.8718
batch_size = 300 epochs = 50	0.2952	0.9175
batch_size = 100 epochs = 50	0.2261	0.9352
batch_size = 50 epochs = 50	0.1920	0.9427
batch_size = 50 epochs = 25	0.2559	0.9251
batch_size = 50 epochs = 75	0.1195	0.9649
batch_size = 100 epochs = 100	0.1474	0.9581

net_2	loss	accuracy
batch_size = 600 epochs = 50	0.3588	0.9014
batch_size = 300 epochs = 50	0.3079	0.9144
batch_size = 100 epochs = 50	0.2055	0.9416
batch_size = 100 epochs = 25	0.2876	0.9161
batch_size = 100 epochs = 75	0.1986	0.9422
batch_size = 200 epochs = 75	0.2418	0.9325
batch_size = 100 epochs = 100	0.1655	0.9524

CNN net_1 CIFRAR10
Rendimiento en el conjunto de pruebas: Pérdida en pruebas: 2.964906692504883 Precisión en pruebas: 0.6563000082969666

CNN net_2 CIFRAR10
Rendimiento en el conjunto de pruebas: Pérdida en pruebas: 3.2239928245544434 Precisión en pruebas: 0.6279000043869019

V. Análisis

MNIST Dataset:

Ambos modelos (net_1 y net_2) muestran un rendimiento bastante bueno en el conjunto de pruebas de MNIST, con precisión alcanzando valores superiores al 90%. Esto sugiere que la arquitectura de la red y los hiperparámetros seleccionados son adecuados para la tarea de clasificación de dígitos escritos a mano.

CIFAR-10 Dataset:

En el conjunto de pruebas de CIFAR-10, los resultados son menos satisfactorios. La precisión está en el rango del 60-65%, lo que indica que hay espacio para mejorar el rendimiento. Este conjunto de datos es más desafiante debido a la naturaleza más

compleja de las imágenes (32x32 a color) y la diversidad de clases.

Comparación entre net_1 y net_2:

Ambos modelos tienen un rendimiento similar en MNIST. Sin embargo, en CIFAR-10, net_1 parece tener un rendimiento ligeramente mejor en términos de precisión.

Hiperparámetros:

La búsqueda de hiperparámetros muestra que, en general, un tamaño de lote (batch size) más pequeño parece beneficiar al rendimiento. Además, el número óptimo de épocas puede variar para diferentes tamaños de lote.

Ajustar los hiperparámetros y la arquitectura de la red puede ser crucial para obtener un rendimiento óptimo en conjuntos de datos específicos.

Sería útil probar otras arquitecturas y técnicas de regularización para mejorar el rendimiento en CIFAR-10.

VI. Conclusión

El análisis y desarrollo de modelos de aprendizaje automático utilizando el conjunto de datos MNIST y CIFAR-10 ha proporcionado una visión profunda de cómo diferentes configuraciones afectan el rendimiento de la red neuronal en tareas específicas de clasificación de imágenes.

Ambos modelos, net_1 y net_2, han mostrado un rendimiento satisfactorio en la clasificación de dígitos escritos a mano en el conjunto de datos MNIST. La precisión superando consistentemente el 90% sugiere que la arquitectura y los hiperparámetros elegidos son adecuados para esta tarea.

Experimentación con Otras Arquitecturas:

Dada la complejidad de CIFAR-10, podría ser beneficioso experimentar con arquitecturas de red más avanzadas, como redes neuronales convolucionales más profundas.

Exploración de Técnicas de Regularización:

La aplicación de técnicas de regularización, como la disminución del sobreajuste mediante la introducción de capas de abandono (dropout), podría mejorar el rendimiento en CIFAR-10.

Ajuste Detallado de Hiperparámetros:

Continuar con la búsqueda de hiperparámetros óptimos, incluyendo tasas de aprendizaje, funciones de activación y otros parámetros específicos de la arquitectura, puede ser crucial para mejorar el rendimiento en ambos conjuntos de datos.

Se proporciona una base sólida para entender la importancia de la elección de arquitecturas y hiperparámetros en el rendimiento de las redes neuronales. Ajustar estos aspectos de manera específica para cada conjunto de datos y tarea de clasificación es esencial para lograr resultados óptimos. La experimentación continua y la optimización de la arquitectura y los hiperparámetros son prácticas fundamentales en el desarrollo de modelos de aprendizaje automático.

Referencias

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[2] Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer.

[3] Aggarwal, C. (2018). Neural Networks and Deep Learning: A Textbook. Springer.

[4] Hastie, Trevor, Robert Tibshirani, Jerome H. Friedman, and Jerome H. Friedman. The elements of statistical learning: Data Mining, Inference, and Prediction. Vol. 2. New York: Springer, 2009.

[5] Francois Chollet, Deep learning with Python, Second Edition

[6] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[7] Simon Prince. Computer Vision: Models, Learning, and Inference. Cambridge University Press 2012