MASTER INTELIGENCIA ARTIFICIAL

MODULO I – FUNDAMENTOS DE IA Y MACHINE LEARNING

1.1- FUNDAMENTOS DE LA IA

1.1.1- CONCEPTOS GENERALES

La inteligencia artificial (IA) es una rama de la ciencia de la computación que se enfoca en la creación de sistemas y programas capaces de realizar tareas que normalmente requieren inteligencia humana. Estas tareas pueden incluir el razonamiento, el aprendizaje, la percepción, el reconocimiento de patrones, la toma de decisiones, el procesamiento del lenguaje natural y muchos otros aspectos cognitivos y conductuales.

A continuación, se presentan algunos conceptos generales fundamentales relacionados con la inteligencia artificial:

Aprendizaje automático (Machine Learning): Es una subdisciplina de la IA que se centra en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender de datos sin ser programadas explícitamente. El aprendizaje automático se puede clasificar en tres categorías principales: aprendizaje supervisado, no supervisado y por refuerzo.
Aprendizaje supervisado: En este enfoque, se proporcionan algoritmos ejemplos de entrada y salida para que puedan aprender a mapear entradas a salidas conocidas. Por ejemplo, mostrar imágenes etiquetadas como "perro" o "gato" para que el modelo pueda aprender a reconocerlos en nuevas imágenes.
Aprendizaje no supervisado: Aquí, el algoritmo se expone solo a datos de entrada y se le pide que encuentre patrones o estructuras ocultas en los datos sin ninguna guía explícita. Es útil para segmentar datos y descubrir agrupamientos naturales.
Aprendizaje por refuerzo: En este caso, un agente interactúa con un entorno y recibe retroalimentación en forma de recompensas o castigos según sus acciones. El agente aprende a tomar decisiones para maximizar la recompensa a lo largo del tiempo.
Redes neuronales artificiales: Son modelos matemáticos inspirados en la estructura y función del cerebro humano. Estas redes consisten en capas de neuronas interconectadas que procesan la información y se utilizan ampliamente en problemas de aprendizaje profundo (deep learning).

Aprendizaje profundo (Deep Learning): Es una rama especializada del aprendizaje automático que se basa en el uso de redes neuronales profundas con múltiples capas ocultas. El aprendizaje profundo ha impulsado muchos avances en el procesamiento de imágenes, el procesamiento del lenguaje natural y otras áreas.

Procesamiento del lenguaje natural (PLN): Se enfoca en permitir que las computadoras comprendan, interpreten y generen lenguaje humano de manera natural. Esto se aplica en chatbots, sistemas de traducción automática, análisis de sentimientos y mucho más.

Visión por computadora: Es una rama de la IA que se ocupa de enseñar a las computadoras a interpretar y comprender el contenido visual, como imágenes y videos. Se utiliza en aplicaciones como reconocimiento facial, detección de objetos y conducción autónoma.

Robótica autónoma: Combina la inteligencia artificial con la ingeniería robótica para desarrollar robots capaces de realizar tareas complejas de manera independiente, adaptándose al entorno en tiempo real.

Ética y seguridad en IA: Con el avance de la IA, surgen importantes consideraciones éticas sobre el uso responsable de esta tecnología, la privacidad de los datos y los posibles sesgos que pueden surgir en los algoritmos. La seguridad en IA también se vuelve crítica para evitar vulnerabilidades y posibles consecuencias negativas.

Estos son solo algunos de los conceptos generales fundamentales en inteligencia artificial. La IA es un campo en constante evolución y expansión, y su aplicación abarca diversas industrias, desde la atención médica hasta el transporte, pasando por la industria del entretenimiento y muchas otras áreas de la vida cotidiana.

1.1.2.- TIPOS DE IA

Inteligencia Artificial Basada en Aprendizaje Automático (Machine Learning): Esta es una de las formas más comunes de inteligencia artificial, en la que los sistemas aprenden a través de datos y experiencias para mejorar su rendimiento en tareas específicas.

Inteligencia Artificial Basada en Aprendizaje Profundo (Deep Learning): Es una subcategoría del aprendizaje automático que utiliza redes neuronales profundas para procesar y aprender representaciones complejas de datos, especialmente útiles en tareas de procesamiento de imágenes, reconocimiento del habla y procesamiento del lenguaje natural.

1.1.3.- TAXONOMIA DE LA IA

La taxonomía de la inteligencia artificial se puede dividir de muchas maneras, pero aquí hay una versión simplificada que incluye las categorías más reconocibles:

Por el tipo de funcionamiento:

- **IA débil:** También conocida como IA estrecha, es una IA diseñada y entrenada para una tarea específica, como un algoritmo de recomendación de música. No tiene conciencia ni comprensión propia y no puede hacer nada más allá de la tarea específica para la que fue programada.
- **IA fuerte:** También conocida como IA general, es una IA que tiene la capacidad de entender, aprender, adaptarse y aplicar su inteligencia a diferentes contextos, de la misma manera que un humano. Aunque es un objetivo a largo plazo para muchos investigadores de IA, no existía tal tecnología hasta mi última actualización en septiembre de 2021.
- **IA superinteligente:** Es una forma hipotética de IA que no solo iguala, sino que supera las capacidades humanas en la mayoría de las tareas económicamente valiosas. La superinteligencia está más allá del alcance de la tecnología actual y está en el ámbito de la especulación y la teoría en este momento.

Por el método de aprendizaje:

- **Aprendizaje supervisado:** Los algoritmos de IA se entrenan con conjuntos de datos etiquetados, donde las entradas correctas y las salidas se les proporcionan explícitamente.
- **Aprendizaje no supervisado:** Los algoritmos de IA se entrenan con datos no etiquetados y deben encontrar patrones y relaciones en los datos por sí mismos.
- **Aprendizaje por refuerzo:** Un tipo de IA que aprende a través de la prueba y error, recibiendo "recompensas" o "castigos" para determinadas acciones, y ajustando su comportamiento en consecuencia.

Por la técnica utilizada:

- **Redes neuronales:** Se trata de sistemas de IA que imitan la forma en que el cerebro humano funciona para aprender de los datos de entrada.
- **Aprendizaje profundo:** Una subcategoría de las redes neuronales que utiliza redes neuronales con múltiples capas (o profundas) para el aprendizaje de características de alto nivel a partir de los datos.

- **Lógica difusa:** Esta técnica se utiliza para interpretar datos que son inciertos, imprecisos o ambiguos, similar a la forma en que los humanos toman decisiones basadas en información incompleta o incierta.
- **Algoritmos genéticos:** Se inspiran en la teoría de la evolución y utilizan conceptos como la "selección natural" para optimizar la solución a un problema.

1.2.- MACHINE LEARNING

1.2.1.Repaso de fundamentos de probabilidad y estadística

1.2.1.1.- **Conceptos Básicos de Estadística Descriptiva:**

- **Media, Mediana, Moda:** Estas son medidas de tendencia central en un conjunto de datos.
- **Varianza y Desviación Estándar:** Son medidas de dispersión que indican cuánto varían los datos respecto a la media.
- **Cuartiles y Percentiles:** Son medidas que dividen el conjunto de datos en partes iguales.

1.2.1.2- Probabilidad:

- **Definiciones y Propiedades de Probabilidad:** Incluyen conceptos como eventos, espacio de muestras, probabilidad condicional, probabilidad conjunta, etc.
- **Teorema de Bayes:** Es un concepto importante en probabilidad que describe la probabilidad de un evento basado en el conocimiento previo de las condiciones que pueden estar relacionadas con el evento. Es fundamental en aprendizaje automático.
- **Distribuciones de Probabilidad:** Las distribuciones como la Normal, Bernoulli, Binomial, Poisson, etc., son muy útiles en diferentes técnicas de aprendizaje automático. Debes entender cómo funcionan y cuándo aplicar cada una.

1.2.1.3.- Estadística Inferencial:

- **Estimación Puntual y por Intervalos:** Esto implica la estimación de parámetros desconocidos de la población a partir de una muestra.
- **Pruebas de Hipótesis:** Las pruebas de hipótesis son fundamentales para verificar las suposiciones o comparar diferentes algoritmos de IA.

1.2.1.4.- Conceptos de Correlación y Regresión:

- **Correlación:** Mide el grado en que dos variables están relacionadas linealmente.
- **Regresión Lineal:** La regresión es la piedra angular de muchas técnicas de aprendizaje supervisado y se utiliza para predecir un valor continuo basándose en una o más variables de entrada.

1.2.1.5- Aprendizaje de Máquinas Estadístico:

- **Funciones de Pérdida y Optimización:** Comprender cómo funcionan las funciones de pérdida y los métodos de optimización es crucial para entrenar modelos de IA eficaces.
- **Validación Cruzada y Overfitting:** Estos conceptos son esenciales para evaluar la eficacia de un modelo y prevenir el sobreajuste.

1.2.2.- Tipos de aprendizaje: supervisado, no supervisado, semisupervisado, reinforcement learning

Otros tipos de sistemas de ML

Si pueden aprender incrementalmente sobre la marcha: online versus batch learning

Si trabajan simplemente comparando nuevos puntos de datos con puntos de datos ya conocidos o en cambio detectando patrones en la data de entrenamiento y construyendo un modelo predictivo

1.2.2.1.- Aprendizaje Supervisado

En el aprendizaje supervisado, el algoritmo de inteligencia artificial se entrena con un conjunto de datos etiquetados, lo que significa que cada ejemplo de datos tiene una salida esperada correspondiente.

Por ejemplo, imagina que tienes un conjunto de imágenes de gatos y perros, y cada imagen está etiquetada con la palabra "gato" o "perro". A través del proceso de aprendizaje supervisado, el algoritmo aprende a distinguir gatos de perros. Una vez entrenado, puedes darle al algoritmo una nueva imagen de un gato o un perro y, basándose en lo que ha aprendido, el algoritmo será capaz de identificar si la imagen es de un gato o de un perro.

1.2.2.2.- Aprendizaje No Supervisado

En el aprendizaje no supervisado, el algoritmo se entrena con un conjunto de datos no etiquetados. Esto significa que los datos de entrada no vienen con una salida correspondiente. El objetivo del aprendizaje no supervisado es encontrar patrones o estructuras subyacentes en los datos.

Un ejemplo de esto es la segmentación de clientes en marketing. Imagina que tienes datos sobre los hábitos de compra de miles de clientes, pero no tienes una clasificación predefinida de tipos de clientes. Al usar técnicas de aprendizaje no supervisado, como el clustering, puedes agrupar a los clientes en diferentes segmentos basándote en sus hábitos de compra. Por ejemplo, podrías encontrar un grupo de clientes que compra productos costosos regularmente, otro grupo que solo compra artículos en oferta, etc.

1.2.2.3.- Aprendizaje Semi-Supervisado

El aprendizaje semi-supervisado es una combinación de aprendizaje supervisado y no supervisado. Aquí, una gran cantidad de datos sin etiquetar se utiliza junto con una pequeña cantidad de datos etiquetados para entrenar a un algoritmo de IA.

Un ejemplo de la vida real de esto podría ser un motor de recomendación de música. Imagina que solo algunas canciones en tu biblioteca están etiquetadas con su género, pero la mayoría no. A través del aprendizaje semi-supervisado, el algoritmo de recomendación podría aprender a asociar características de las canciones etiquetadas a géneros específicos y luego aplicar este conocimiento para inferir los géneros de las canciones no etiquetadas. Luego, el sistema podría recomendarte canciones basándose en los géneros que pareces preferir.

1.2.2.4.- Aprendizaje por refuerzo (Reinforcement learning)

Se centra en cómo un agente debe actuar en un entorno para maximizar alguna noción de recompensa acumulativa. Se basa en la idea de que si un agente toma una acción que resulta ser beneficiosa, debería ser más probable que tome esa acción nuevamente en el futuro.

El proceso típico en un sistema de Reinforcement Learning es el siguiente: el agente observa el estado actual del entorno, toma una acción basada en ese estado, luego recibe una recompensa y llega a un nuevo estado. Este proceso se repite muchas veces, y el agente aprende a través de la experiencia de prueba y error.

Por ejemplo, supongamos que tienes un agente que está aprendiendo a jugar al ajedrez. El 'entorno' es el tablero de ajedrez, los 'estados' son las diferentes configuraciones del tablero, las 'acciones' son los posibles movimientos que puede hacer el agente, y las 'recompensas' son las consecuencias de esos movimientos (ganar, perder o empatar el juego).

El objetivo del agente es aprender una política, que es básicamente una estrategia que le dice qué acción tomar en cada estado. Idealmente, esta política debería permitirle maximizar la recompensa total que recibe a lo largo del tiempo.

1.2.3.- Tipos de problemas: regresión, clasificación, agrupamiento

1.2.3.1.- Regresión

La regresión se utiliza para predecir un valor continuo en función de uno o más valores de entrada. Un ejemplo de esto sería predecir el precio de una casa basado en características como su ubicación, el número de habitaciones, el tamaño del lote, etc. Los algoritmos de regresión intentan trazar una línea (o una superficie, en el caso de múltiples variables de entrada) que modela la relación entre las entradas y la salida. Los modelos de regresión más comunes incluyen la regresión lineal y la regresión logística.

1.2.3.2.- Clasificación

La clasificación es un tipo de problema en el que el objetivo es predecir una categoría o clase a partir de uno o más valores de entrada. Un ejemplo común es el filtro de spam de un proveedor de correo electrónico, que clasifica los correos electrónicos como "spam" o "no spam" basándose en una serie de características del correo electrónico. Algunos de los algoritmos más comunes para los problemas de clasificación incluyen las máquinas de vectores de soporte, los árboles de decisión y las redes neuronales.

1.2.3.3.-Agrupamiento

El agrupamiento, o "clustering", es una técnica de aprendizaje no supervisado que se utiliza para encontrar grupos o clusters de puntos de datos similares en un conjunto de datos. En lugar de predecir una clase o un valor continuo, el objetivo del agrupamiento es identificar la estructura subyacente de los datos. Un ejemplo común de uso del agrupamiento es en la segmentación de clientes para marketing. Podrías tener datos sobre los hábitos de compra de tus clientes y utilizar un algoritmo de agrupamiento para identificar grupos de clientes con comportamientos similares. Los algoritmos de agrupamiento más comunes incluyen K-means,

agrupamiento jerárquico y DBSCAN.

Cada uno de estos tipos de problemas tiene su lugar en el aprendizaje automático, y la elección de cuál utilizar depende de la naturaleza de tus datos y de lo que estás tratando de lograr.

1.2.4.- Etapas de trabajo en Machine Learning

a.- Definición del problema: Todo proyecto de Machine Learning comienza con una clara definición del problema. Esto puede implicar definir la pregunta de negocio o investigación, identificar los objetivos del proyecto, y decidir qué tipo de solución de Machine Learning (por ejemplo, clasificación, regresión, agrupamiento, etc.) puede ser la más adecuada.

b.- Recolección de datos: En esta etapa, se recopilan los datos que se utilizarán para entrenar y probar el modelo de Machine Learning. Los datos pueden provenir de diversas fuentes, como bases de datos, archivos de texto, hojas de cálculo, feeds de datos en tiempo real, etc.

c.- Preprocesamiento de los datos: Antes de que los datos puedan ser usados en un modelo, a menudo necesitan ser limpiados y transformados. Esto puede implicar manejar valores perdidos, eliminar duplicados, cambiar el tipo de datos, normalizar o escalar características, codificar variables categóricas, etc.

d.- Análisis exploratorio de datos (EDA): En esta etapa, se analizan los datos para entender su estructura, relaciones y patrones. Esto puede implicar la utilización de estadísticas descriptivas, visualizaciones, pruebas de correlación, etc.

e.- Selección y entrenamiento del modelo: Se elige el modelo de Machine Learning que mejor se ajusta al problema y a los datos, y se entrena utilizando el conjunto de datos de entrenamiento. Se dividen los datos en training data (80%) and test data(20%)

f.- Evaluación del modelo: Una vez que el modelo ha sido entrenado, se evalúa su rendimiento utilizando el conjunto de datos de prueba. Existen **varias** métricas para evaluar el rendimiento de un modelo, y las más adecuadas dependen del tipo de tarea de Machine Learning (por ejemplo, precisión, recall, AUC-ROC para tareas de clasificación, error cuadrático medio para tareas de regresión, etc.).

g.- Ajuste de hiperparámetros: En esta etapa, se ajustan los hiperparámetros del modelo para mejorar su rendimiento. Los hiperparámetros son los parámetros del modelo que se establecen antes del entrenamiento y no se aprenden a partir de los datos.

h.- Implementación y monitoreo: Una vez que se ha obtenido un modelo satisfactorio, se implementa en el sistema de producción. Después de la implementación, el modelo debe ser monitoreado regularmente para asegurar que continúa funcionando como se esperaba.

1.2.5.- Técnicas de carga y preprocesamiento de datos

En un proyecto de Machine Learning, el manejo y preprocesamiento de los datos son pasos fundamentales que tienen un gran impacto en el rendimiento final de los modelos. Aquí te proporciono una descripción de las principales técnicas de carga y preprocesamiento de datos:

Carga de datos:

Lectura de archivos CSV/Excel: Estos son formatos comunes para almacenar datos. Las bibliotecas como Pandas en Python tienen funciones para leer estos archivos directamente en DataFrames, que son estructuras de datos tabulares.
Conexión a bases de datos: Muchos datos se almacenan en bases de datos relacionales o NoSQL. Python tiene bibliotecas como SQLAlchemy, sqlite3, PyMongo, entre otros, que pueden conectar tu código con estas bases de datos y cargar los datos en tu entorno de trabajo.
Web Scraping: Si los datos que necesitas están en páginas web, puedes utilizar técnicas de web scraping para extraerlos. Bibliotecas como Beautiful Soup o Scrapy en Python pueden ser útiles para esta tarea.
APIs: Muchas empresas y organizaciones ofrecen APIs (Application Programming Interfaces) que permiten acceder a sus datos. Por lo general, tendrás que enviar una solicitud HTTP a la API y luego procesar la respuesta, que a menudo está en formato JSON o XML.

Preprocesamiento de datos:

Limpieza de datos: Los datos crudos a menudo contienen errores, valores faltantes, duplicados o información irrelevante. La limpieza de los datos implica identificar y tratar estos problemas. Por ejemplo, puedes llenar los valores faltantes con la media o la mediana, o puedes eliminar las filas o columnas que contienen valores faltantes.
Transformación de datos: Esto puede incluir la normalización o estandarización de las características, es decir, ajustar los datos a una escala común. También puede incluir la codificación de variables categóricas (por ejemplo, convertir una característica de categoría en múltiples características binarias).
Selección de características: No todas las características en tu conjunto de datos pueden ser útiles para tu modelo de Machine Learning. Algunas pueden tener poco impacto en la variable objetivo, otras pueden ser redundantes. Las técnicas de selección de características pueden ayudarte a identificar las características más relevantes.
Ingeniería de características: Esta es la creación de nuevas características a partir de las existentes. Por ejemplo, puedes combinar dos características existentes para crear una nueva, o puedes transformar una característica (por ejemplo, tomando su logaritmo) para cambiar su relación con la variable objetivo.
División de los datos: Para evaluar correctamente tu modelo, necesitarás dividir tus datos en un conjunto de entrenamiento y un conjunto de prueba. A veces, también se usa un conjunto de validación para ajustar los hiperparámetros.

1.2.6.- Selección de algoritmos de Machine Learning

La selección de un algoritmo de Machine Learning depende de una variedad de factores, que incluyen el tipo de problema que estás tratando de resolver, la naturaleza de tus datos y tus requisitos de rendimiento y precisión. Aquí te dejo algunos de los algoritmos más comunes y cuándo podrías querer usarlos:

Regresión Lineal: Es uno de los algoritmos más simples. Se utiliza cuando la variable objetivo es continua y existe una relación lineal entre las variables de entrada y de salida. Por ejemplo, puedes usar regresión lineal para predecir el precio de una casa en función de su tamaño y ubicación.
Regresión Logística: Se utiliza cuando la variable objetivo es categórica y se quiere predecir la probabilidad de que una observación pertenezca a una categoría específica. Es comúnmente usado en problemas de clasificación

binaria, como predecir si un correo electrónico es spam o no.
Árboles de Decisión y Bosques Aleatorios: Estos algoritmos son útiles para problemas de clasificación y regresión. Los árboles de decisión son fáciles de entender y visualizar, pero pueden ser propensos al sobreajuste. Los bosques aleatorios, que son conjuntos de árboles de decisión, pueden ayudar a evitar este problema.
Máquinas de Vectores de Soporte (SVM): Son algoritmos de aprendizaje supervisado que se utilizan para problemas de clasificación y regresión. SVM es útil para clasificar datos no lineales o en problemas de alta dimensionalidad.
K-Nearest Neighbors (KNN): Es un algoritmo de aprendizaje supervisado que se puede utilizar para problemas de clasificación y regresión. KNN clasifica una observación basándose en las observaciones más cercanas en el espacio de características.
Redes Neuronales y Deep Learning: Son útiles para una amplia gama de problemas, incluyendo clasificación de imágenes, procesamiento de lenguaje natural, y predicción de series de tiempo. Requieren una gran cantidad de datos para entrenarse y pueden ser computacionalmente intensivas.

Estos son solo algunos ejemplos de los muchos algoritmos de machine learning que existen. La elección del algoritmo correcto dependerá de tu problema específico, así como de factores como el tamaño de tus datos, la calidad de tus datos, y tus requisitos de rendimiento y precisión.

1.2.7.- Entrenamiento y validación de modelos

El entrenamiento y la validación de modelos son dos pasos fundamentales en el proceso de desarrollo de un modelo de Machine Learning.

Entrenamiento de Modelos: Aquí es donde se "enseña" al modelo sobre los datos. Para hacer esto, se usa un conjunto de datos conocido como conjunto de entrenamiento. Este conjunto de datos contiene las entradas (a veces llamadas características) y las salidas correctas (a veces llamadas etiquetas o objetivos). El objetivo del entrenamiento es ajustar los parámetros del modelo para minimizar el error entre las predicciones del modelo y las salidas correctas. Dependiendo del algoritmo de machine learning que se utilice, este proceso puede implicar encontrar la línea de mejor ajuste (en el caso de la regresión lineal), la construcción de un árbol de decisión (para un modelo de árbol de decisión) o ajustar los pesos y sesgos en una red neuronal (para modelos de deep learning).

Validación de Modelos: Una vez que el modelo ha sido entrenado, es importante verificar cómo de bien se generaliza a datos no vistos. Para esto, se utiliza un conjunto de datos separado, conocido como conjunto de validación. Este conjunto de datos se utiliza para ajustar los hiperparámetros del modelo (como la tasa de aprendizaje en una red neuronal o la profundidad de un árbol de decisión) y para verificar la capacidad del modelo para generalizar a partir de los datos de entrenamiento. Si un modelo tiene un buen rendimiento en el conjunto de entrenamiento pero un rendimiento pobre en el conjunto de validación, esto puede ser un signo de sobreajuste, lo que significa que el modelo ha aprendido tan bien los datos de entrenamiento que no puede generalizar bien a los datos no vistos.

Es importante tener en cuenta que los conjuntos de entrenamiento y validación deben ser representativos de los datos que el modelo verá en la "vida real". Además, nunca se deben usar los datos de validación para entrenar el modelo.

Una técnica comúnmente utilizada en la validación de modelos es la validación cruzada (cross-validation). En la validación cruzada, el conjunto de datos se divide en 'k' subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto ('k-1') como datos de entrenamiento. El proceso de validación cruzada es repetido durante 'k' iteraciones, con cada uno de los subconjuntos de datos utilizados exactamente una vez como datos de prueba. La media de los resultados de las 'k' pruebas se toma como el resultado.

Estos dos pasos, junto con la prueba final del modelo en un conjunto de prueba separado, son cruciales para desarrollar modelos de machine learning eficaces y precisos.

1.3.- Optimización

1.3.1.- Algoritmos de optimización: descenso de gradiente

El descenso de gradiente es un algoritmo de optimización ampliamente utilizado en machine learning y deep learning para minimizar una función de pérdida. A menudo, el objetivo de un modelo de aprendizaje automático es encontrar los parámetros del modelo que minimicen la función de pérdida.
El descenso de gradiente se basa en el hecho de que la función de pérdida disminuye más rápidamente en la dirección del gradiente negativo. Por lo tanto, en cada paso del algoritmo, actualizamos los parámetros del modelo moviéndonos en la dirección del gradiente negativo de la función de pérdida.

Aquí está el algoritmo básico de descenso de gradiente:

1. Inicializa los parámetros del modelo con algún valor inicial. Esto puede ser un valor aleatorio o algún valor predefinido.
2. Calcula el gradiente de la función de pérdida con respecto a los parámetros del modelo.
3. Actualiza los parámetros del modelo moviéndote en la dirección del gradiente negativo. Esto se hace restando el gradiente (multiplicado por un factor de aprendizaje, a menudo llamado "learning rate") de los parámetros actuales.
4. Repite los pasos 2 y 3 hasta que el gradiente sea muy pequeño (es decir, estás cerca del mínimo de la función de pérdida) o hasta que alcances un número máximo de iteraciones.

Existen varias variantes del algoritmo básico de descenso de gradiente, que incluyen:

- Descenso de gradiente por lotes (Batch Gradient Descent): Se utiliza todo el conjunto de datos de entrenamiento para calcular el gradiente de la función de pérdida en cada paso.
- Descenso de gradiente estocástico (Stochastic Gradient Descent, SGD): Se utiliza una sola instancia de entrenamiento seleccionada al azar para calcular el gradiente en cada paso. Es generalmente más rápido que el descenso de gradiente por lotes, pero sus actualizaciones son más ruidosas, lo que puede resultar en una convergencia más lenta.
- Descenso de gradiente de mini lotes (Mini-batch Gradient Descent): Es un compromiso entre el descenso de gradiente por lotes y el estocástico. Se utiliza un pequeño lote aleatorio de instancias de entrenamiento para calcular el gradiente en cada paso.

También hay técnicas avanzadas de descenso de gradiente que incluyen términos de momento (como el gradiente descendiente con impulso) o que ajustan dinámicamente la tasa de aprendizaje (como Adagrad, RMSprop y Adam).

1.3.2.- Métricas de evaluación

La evaluación de un modelo de machine learning es un paso esencial para entender cómo de bien está funcionando. Dependiendo de si tu modelo es de clasificación, regresión o algún otro tipo de modelo, hay diferentes métricas que puedes usar. Aquí te dejo algunas de las métricas más comunes:

Para Clasificación:

- Precisión (Accuracy): Es la proporción de predicciones correctas sobre el total de predicciones. Es una métrica útil cuando las clases están bien balanceadas.
- Recall (Sensitivity o True Positive Rate): Es la proporción de verdaderos positivos que se identificaron correctamente. Es útil en situaciones donde los falsos negativos son más preocupantes que los falsos positivos.
- Precision: Es la proporción de verdaderos positivos entre todas las predicciones positivas. Es útil en situaciones donde los falsos positivos son más preocupantes que los falsos negativos.
- F1 Score: Es la media armónica de Precision y Recall. Intenta equilibrar ambas métricas y es más útil que la precisión cuando tienes una distribución de clases desequilibrada.
- Area Under the ROC Curve (AUC-ROC): Es la probabilidad de que un clasificador ordene un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio. Un AUC de 1 indica una clasificación perfecta, mientras que un AUC de 0.5 indica un clasificador aleatorio.

Para Regresión:

- Mean Absolute Error (MAE): Es la media del valor absoluto de los errores. Da una idea de cuánto te estás equivocando en tus predicciones.
- Mean Squared Error (MSE): Es la media de los cuadrados de los errores. Da más peso a los errores grandes.
- Root Mean Squared Error (RMSE): Es la raíz cuadrada de la media de los cuadrados de los errores. También da más peso a los errores grandes y tiene la misma unidad que la variable objetivo.
- R-squared (Coeficiente de Determinación): Proporciona una medida de cuánto de la variabilidad en la variable objetivo puede ser explicada por las características del modelo. Un valor de 1 indica que el modelo explica toda la variabilidad, mientras que un valor de 0 indica que el modelo no explica nada de la variabilidad.

Para Clustering:

- Silhouette Score: Esta métrica se usa para medir cuán bien se han agrupado los datos en los clusters.
- Davies-Bouldin Index: Este índice indica la similitud media entre los clusters. Los valores más bajos indican una mejor partición.

Estas son solo algunas de las métricas disponibles. La elección de la métrica depende en gran medida de tu problema y de lo que más te importe en tus predicciones.

1.3.3.- Tasa de aprendizaje

La tasa de aprendizaje es un hiperparámetro crucial en muchos algoritmos de Machine Learning que influye en cómo se actualizan los modelos en respuesta a los datos de entrenamiento. En el aprendizaje supervisado, la tasa de aprendizaje determina cuánto cambian los parámetros de un modelo en respuesta a la información obtenida en cada iteración.

En el contexto del descenso de gradiente, la tasa de aprendizaje se utiliza para multiplicar el gradiente de la función de pérdida con respecto a los parámetros del modelo. Este producto, que es un vector en el espacio de los parámetros, se resta luego de los parámetros actuales para actualizarlos.

La tasa de aprendizaje tiene un impacto importante en la convergencia del algoritmo:

- Si la tasa de aprendizaje es demasiado alta, el algoritmo puede saltarse el mínimo de la función de pérdida y nunca converger.
- Si la tasa de aprendizaje es demasiado baja, el algoritmo puede tardar mucho en converger, o incluso quedarse atrapado en un mínimo local en lugar de encontrar el mínimo global.

No hay una fórmula mágica para seleccionar la tasa de aprendizaje perfecta. A menudo se selecciona a través de prueba y error, probando diferentes valores y viendo cuál funciona mejor en un conjunto de validación, o se utiliza un método de ajuste de hiperparámetros como la búsqueda de cuadrícula o la búsqueda aleatoria.

Algunos optimizadores, como Adam o RMSprop, ajustan dinámicamente la tasa de aprendizaje durante el entrenamiento, lo que puede hacer que la elección de la tasa de aprendizaje inicial sea menos crítica.

Es importante recordar que la tasa de aprendizaje es sólo uno de los muchos hiperparámetros que pueden necesitar ser ajustados en un modelo de Machine Learning, y que estos hiperparámetros pueden interactuar de formas complejas.

1.3.4.- Overfitting y underfitting

Overfitting y underfitting son dos problemas comunes que pueden ocurrir al entrenar un modelo de machine learning.

Overfitting (Sobreajuste): Esto ocurre cuando un modelo aprende el conjunto de datos de entrenamiento "demasiado bien". Se adapta tanto a los datos de entrenamiento que se ajusta incluso al ruido y a las peculiaridades en los datos de entrenamiento que no son relevantes para la tarea en sí. Como resultado, el modelo puede tener un rendimiento muy bueno en los datos de entrenamiento, pero puede no generalizar bien a los datos no vistos o de prueba. En otras palabras, es como si el modelo hubiera memorizado las respuestas del examen en lugar de entender realmente las preguntas. El sobreajuste puede ocurrir cuando un modelo es demasiado complejo, como un modelo de aprendizaje profundo con demasiados parámetros en comparación con el número de observaciones en los datos de entrenamiento.
Underfitting (Subajuste): Esto ocurre cuando un modelo no puede aprender adecuadamente los patrones en los datos de entrenamiento. A diferencia del sobreajuste, en este caso, el modelo no se ajusta suficientemente bien incluso a los datos de entrenamiento, y por lo tanto tampoco se desempeña bien en los datos de prueba. El subajuste puede ocurrir cuando un modelo es demasiado simple (por ejemplo, un modelo lineal que se utiliza para describir datos no lineales) y/o cuando no se entrena el modelo lo suficiente.

Aquí hay algunas estrategias para combatir el sobreajuste y el subajuste:

- Recopilación de más datos: En general, entrenar un modelo con más datos puede ayudar a mejorar su capacidad para generalizar y puede reducir la posibilidad de sobreajuste.
- Regularización: Técnicas como la regularización L1 y L2 introducen una penalización en la función de pérdida del modelo para reducir la complejidad del modelo y prevenir el sobreajuste.
- Ajuste de la arquitectura del modelo: Si un modelo está subajustando los datos, podría ser útil añadir más capas o neuronas (en el caso de las redes neuronales) para aumentar la complejidad del modelo. Si un modelo está sobreajustando los datos, podrías probar con un modelo más sencillo.
- Validación cruzada: Esta es una técnica que se utiliza para obtener una medida robusta del rendimiento del modelo en datos no vistos. Puede ser útil para seleccionar el modelo que mejor se ajusta a los datos.
- Conjunto de validación y pruebas: Es importante tener un conjunto separado de datos para validar y probar el modelo, de forma que se pueda obtener una

estimación no sesgada de cómo se desempeñará el modelo en datos no vistos.

- Early stopping: En el aprendizaje de las redes neuronales, una forma común de prevenir el sobreajuste es detener el entrenamiento cuando el rendimiento del modelo en el conjunto de validación comienza a empeorar.
- Dropout: Es una técnica utilizada en el entrenamiento de las redes neuronales donde algunas neuronas se "desactivan" o se ignoran durante el entrenamiento. Esto ayuda a prevenir el sobreajuste al forzar a la red a aprender representaciones más robustas.
- Data Augmentation: Es una estrategia que se utiliza para aumentar la cantidad de datos de entrenamiento mediante la creación de versiones modificadas de los datos existentes. Por ejemplo, en el procesamiento de imágenes, se podrían crear nuevas imágenes mediante la rotación, el escalamiento o el recorte de las imágenes existentes. Esta técnica puede ayudar a prevenir el sobreajuste.

1.3.5.- Técnicas de regularización

La regularización es una técnica utilizada en el aprendizaje automático para prevenir el sobreajuste y mejorar la generalización del modelo en datos no vistos. La idea básica es añadir una penalización a la función de pérdida del modelo que refuerza la preferencia por modelos más simples. A continuación, te presento algunas de las técnicas de regularización más comunes:

Regularización L1 (Lasso): La regularización L1 agrega un término a la función de pérdida que es proporcional a la suma de los valores absolutos de los coeficientes del modelo. Esto tiene el efecto de hacer que algunos de los coeficientes del modelo se reduzcan a cero, lo que puede ser útil para la selección de características.
Regularización L2 (Ridge): La regularización L2 agrega un término a la función de pérdida que es proporcional a la suma de los cuadrados de los coeficientes del modelo. Esto tiene el efecto de reducir la magnitud de los coeficientes del modelo pero no los reduce a cero.
Regularización Elastic Net: Esta es una combinación de la regularización L1 y L2. Agrega tanto un término L1 como un término L2 a la función de pérdida. Este método puede ser útil cuando hay características correlacionadas.
Dropout: Esta es una técnica de regularización utilizada en las redes neuronales. Durante el entrenamiento, algunas de las neuronas de la red se "desactivan" aleatoriamente en cada paso, lo que ayuda a prevenir el sobreajuste. El Dropout puede ser visto como una forma de regularización en

la que se aplica una penalización a la complejidad de la red al reducir el número de neuronas que se utilizan.
Early Stopping: Esta es otra técnica utilizada en las redes neuronales para prevenir el sobreajuste. Durante el entrenamiento, se monitorea el rendimiento del modelo en un conjunto de validación. Cuando el rendimiento en el conjunto de validación comienza a empeorar (indicando que el modelo puede estar comenzando a sobreajustar los datos de entrenamiento), se detiene el entrenamiento.
Regularización de pesos de la red (Weight Decay): Esta técnica agrega un término a la función de pérdida que es proporcional a la suma de los cuadrados de los pesos de la red. Esta técnica es similar a la regularización L2 pero se aplica a las redes neuronales.

Estas técnicas de regularización pueden ayudar a mejorar el rendimiento del modelo en datos no vistos al prevenir el sobreajuste. La elección de la técnica de regularización y el valor del hiperparámetro de regularización (que determina cuánto se penaliza la complejidad del modelo) puede depender de tu problema específico y puede requerir un poco de experimentación.

MODULO II – FUNDAMENTOS DE REDES NEURONALES E INTRODUCCIÓN AL DEEP LEARNING

2.1. Fundamentos de redes neuronales

2.1.1.- Neuronas artificiales

Las neuronas artificiales, también conocidas como nodos o unidades, son componentes fundamentales de las redes neuronales en deep learning. Están inspiradas en las neuronas biológicas, las células nerviosas que forman el sistema nervioso, incluyendo el cerebro.

En términos simples, una neurona artificial recibe una o más entradas, realiza una operación de cálculo y produce una salida. Aquí te dejo los componentes clave de una neurona artificial:

Entradas: Cada neurona artificial en una red recibe una serie de entradas. Estas pueden ser datos de entrada directos (como los píxeles de una imagen

en la primera capa de una red neuronal convolucional) o salidas de otras neuronas en las capas anteriores.

Pesos y sesgos: Cada entrada está asociada con un peso, que determina la importancia de la entrada. Si el peso es grande, significa que la entrada tiene una gran influencia en la salida de la neurona. El sesgo es un término adicional que permite ajustar la salida de la neurona independientemente de sus entradas.

Función de combinación: Las entradas y los pesos se combinan para formar una suma ponderada. A esta suma ponderada se le añade un término de sesgo.

Función de activación: La salida de la función de combinación se pasa a través de una función de activación, que determina la salida final de la neurona. La función de activación puede ser lineal (como en las neuronas de la capa de salida para ciertos problemas de regresión) o no lineal (como la función ReLU, sigmoide o tanh que se utiliza en las capas ocultas). La función de activación introduce no linealidades que permiten a la red neuronal aprender patrones complejos.

Cuando las neuronas artificiales se organizan en capas y se conectan entre sí, forman una red neuronal artificial. En el deep learning, estas redes pueden tener muchas capas, de ahí el término "deep" (profundo). El proceso de aprendizaje implica ajustar los pesos y los sesgos de las neuronas en la red para minimizar la diferencia entre las salidas de la red y los valores verdaderos, dada una función de pérdida y un conjunto de datos de entrenamiento.

2.1.2.- Perceptrón simple

El perceptrón simple, también conocido simplemente como "perceptrón", es una de las estructuras más simples de una red neuronal artificial. Es esencialmente una neurona artificial de un solo nivel y fue el primer modelo de neurona artificial propuesto. Fue desarrollado por Frank Rosenblatt en 1957.

El perceptrón toma un conjunto de entradas, cada una de las cuales se multiplica por un peso, se suman los resultados y luego se aplica una función de activación para producir la salida. La estructura básica es la siguiente:

Entradas: El perceptrón recibe un conjunto de entradas (x1, x2, ..., xn). Estas entradas pueden representar diferentes características de un objeto. Por ejemplo, si estamos tratando de clasificar correos electrónicos como spam o no spam, las entradas podrían ser la frecuencia de ciertas palabras o

caracteres.
Pesos: Cada entrada está asociada con un peso (w1, w2, ..., wn). Los pesos son los parámetros que el modelo aprenderá durante el entrenamiento. Indican la importancia de cada característica para la decisión final.
Combinación Lineal: Las entradas y los pesos se combinan en una suma ponderada: suma(wi * xi). A esta suma se le añade un término de sesgo (b), que se puede considerar como un peso que no depende de las entradas.
Función de Activación: La suma ponderada se pasa a través de una función de activación para producir la salida del perceptrón. En el caso del perceptrón, la función de activación es a menudo la función escalón unitario, que produce una salida de 1 si la suma ponderada es mayor que un cierto umbral, y 0 en caso contrario.

Es importante notar que el perceptrón simple es un modelo lineal y sólo puede aprender fronteras de decisión lineales. No puede manejar problemas que son no lineales, es decir, problemas en los que la frontera de decisión entre clases no es una línea recta (en 2D) o un plano (en 3D). Sin embargo, al combinar múltiples perceptrones en una red multicapa (lo que llamamos una red neuronal multicapa), podemos modelar relaciones más complejas y aprender fronteras de decisión no lineales.

2.1.3.- Funciones de activación

Las funciones de activación en deep learning son operaciones matemáticas que transforman la suma ponderada de las entradas de una neurona en la salida que será enviada al siguiente nivel de la red. Estas funciones son esenciales para que las redes neuronales puedan aprender de los datos no lineales.

Existen varias funciones de activación que se utilizan comúnmente en deep learning, incluyendo:

Función sigmoide: Esta función toma cualquier valor de entrada y lo comprime en un rango entre 0 y 1. Es especialmente útil en la capa de salida de una red neuronal para problemas de clasificación binaria.
Función de unidad lineal rectificada (ReLU): Esta función toma cualquier valor de entrada y lo convierte en 0 si es negativo. Si el valor de entrada es positivo, lo deja igual. Se ha encontrado que ReLU y sus variantes (como Leaky ReLU y Parametric ReLU) funcionan bien en la mayoría de las aplicaciones de redes
neuronales, ya que ayudan a mitigar el problema del desvanecimiento

del gradiente.

Función tangente hiperbólica (tanh): Esta función toma cualquier valor de entrada y lo comprime en un rango entre -1 y 1. Es similar a la función sigmoide, pero centrada en 0, lo que puede facilitar el aprendizaje en algunas situaciones.

Función Softmax: Esta función se utiliza comúnmente en la capa de salida de una red neuronal para problemas de clasificación multiclase. Convierte los valores de entrada en probabilidades que suman 1, lo que permite interpretar las salidas como probabilidades para cada clase.

La elección de la función de activación depende de varios factores, como el problema que se está tratando de resolver, el tipo de red neuronal que se está utilizando, y las características de los datos de entrada. Cada función tiene sus propias características y puede influir en el rendimiento del modelo de deep learning.

2.1.4.- Conceptos generales de redes neuronales

Las redes neuronales artificiales (ANN) son modelos computacionales inspirados en el sistema nervioso humano. Se utilizan en el machine learning y el deep learning para realizar tareas complejas de clasificación, predicción, reconocimiento de patrones y más. A continuación, te presento algunos conceptos clave relacionados con las redes neuronales:

Neurona Artificial: Una neurona artificial o unidad es el componente básico de una red neuronal. Cada neurona recibe una serie de entradas, las pondera en función de unos pesos (que representan la importancia de las respectivas entradas para la salida), y luego pasa la suma ponderada a través de una función de activación para determinar la salida.

Capas: Las neuronas en una red neuronal se organizan en capas. Una red neuronal típica consiste en una capa de entrada, una o más capas ocultas y una capa de salida. Las capas de entrada reciben los datos de entrada, las capas ocultas procesan los datos y la capa de salida proporciona el resultado final.

Pesos y sesgos: Los pesos y sesgos son los parámetros que una red neuronal aprende durante el entrenamiento. Los pesos determinan la importancia de cada entrada, y el sesgo permite ajustar la salida de la neurona junto con los pesos.

Función de activación: La función de activación determina la salida de una neurona basada en sus entradas ponderadas. Existen diferentes tipos de

funciones de activación como la función sigmoide, la función de unidad lineal rectificada (ReLU), la tangente hiperbólica (tanh), entre otras.
Entrenamiento: Durante el entrenamiento, los pesos y sesgos de la red se ajustan para minimizar el error entre la salida de la red y el valor real. Esto se hace utilizando algoritmos de optimización, como el descenso del gradiente.
Retropropagación: Es el método utilizado para actualizar los pesos y sesgos de la red. La idea básica de la retropropagación es calcular el gradiente de la función de pérdida (una medida del error) con respecto a los pesos de la red y utilizar este gradiente para actualizar los pesos para minimizar la pérdida.
Overfitting y Underfitting: Overfitting se produce cuando la red aprende los datos de entrenamiento demasiado bien, al punto que tiene un rendimiento deficiente en los datos de prueba o nuevos datos. Underfitting es lo contrario; la red no aprende lo suficiente de los datos de entrenamiento, lo que también lleva a un mal rendimiento en los datos de prueba.
Regularización: Son técnicas que se usan para prevenir el overfitting, como la regularización L1 y L2, dropout, y early stopping.

Estos son solo algunos de los conceptos básicos relacionados con las redes neuronales. El campo del deep learning, donde se utilizan redes neuronales con muchas capas, ha introducido una serie de nuevas arquitecturas y conceptos adicionales.

2.1.5.- Feedforward y backpropagation

Feedforward

El feedforward es el proceso por el cual una red neuronal artificial genera una salida a partir de una entrada. En este proceso, la información se propaga hacia adelante desde la capa de entrada, a través de las capas ocultas, hasta la capa de salida.

Cada neurona en una capa recibe las salidas de todas las neuronas de la capa anterior, multiplica cada entrada por su peso correspondiente, suma todos estos productos y luego aplica una función de activación a este total. Este resultado se pasa a las neuronas de la siguiente capa y así sucesivamente.

El feedforward es una operación de "una vía", en el sentido de que la información se mueve en una sola dirección - de la entrada a la salida - y no hay ciclos o bucles en la red.

Backpropagation

El Backpropagation, o propagación hacia atrás, es un algoritmo utilizado para entrenar redes neuronales, es decir, para optimizar los pesos y sesgos en una red neuronal. Se utiliza en combinación con un algoritmo de optimización como el descenso de gradiente.

El objetivo del entrenamiento de una red neuronal es minimizar la diferencia entre la salida actual de la red y la salida deseada. Esta diferencia se cuantifica en una función de coste o de pérdida.

El proceso de backpropagation consta de dos pasos:

> En la fase de propagación hacia adelante (feedforward), la red genera una salida para una entrada dada. Se calcula la pérdida, que es una medida de la discrepancia entre la salida generada y la salida deseada.
> Luego, en la fase de propagación hacia atrás, se calcula el gradiente de la función de pérdida con respecto a los pesos y sesgos de la red, es decir, se calcula cuánto cambiaría la pérdida si se cambiaran los pesos y los sesgos. Esto se hace aplicando la regla de la cadena del cálculo diferencial, propagando los errores desde la capa de salida hasta la capa de entrada.

Finalmente, se utiliza un algoritmo de optimización, como el descenso de gradiente, para ajustar los pesos y sesgos en la dirección que reduce la pérdida.

En resumen, el feedforward es el proceso por el cual una red neuronal genera una salida, y el backpropagation es el proceso por el cual la red aprende de los errores que cometió y ajusta sus pesos y sesgos para mejorar sus predicciones futuras.

2.1.6.- Diseño de redes neuronales

Diseñar una red neuronal implica tomar una serie de decisiones sobre su arquitectura y configuración. Aquí te detallo algunos de los pasos y consideraciones clave:

Elección del tipo de red: Debes decidir qué tipo de red neuronal es más apropiado para tu problema. Las redes neuronales de alimentación directa, como las redes neuronales de perceptrón multicapa (MLP), son una elección común para los problemas de clasificación y regresión. Las redes convolucionales (CNN) son ideales para problemas que implican imágenes, mientras que las redes recurrentes (RNN) son adecuadas para problemas que implican datos secuenciales o de series temporales.

Número de capas: Debes decidir cuántas capas de neuronas (nodos) se deben usar en la red. Las redes con más capas pueden aprender relaciones más complejas, pero también pueden ser más propensas al overfitting y requieren más tiempo de entrenamiento. En general, puedes comenzar con una o dos capas ocultas y aumentar el número si es necesario.

Número de neuronas por capa: También debes decidir cuántas neuronas colocar en cada capa. En general, puedes comenzar con un número de neuronas en la capa oculta que sea el promedio del número de neuronas en la capa de entrada y el número de neuronas en la capa de salida.

Función de activación: Necesitas seleccionar la función de activación para las neuronas en la red. Las opciones comunes incluyen la función sigmoide, tanh, ReLU y softmax. La elección de la función de activación puede depender del problema que estás resolviendo. Por ejemplo, la función softmax es una buena opción para la capa de salida de un problema de clasificación multiclase.

Inicialización de pesos: Los pesos de la red deben inicializarse antes de comenzar el entrenamiento. Una opción común es inicializar los pesos con pequeños valores aleatorios.

Optimización y tasa de aprendizaje: Debes seleccionar un algoritmo de optimización (como el descenso de gradiente, el descenso de gradiente estocástico, Adam, etc.) y establecer una tasa de aprendizaje. La tasa de aprendizaje determina cuán rápido la red aprende de los datos.

Función de coste: Debes seleccionar una función de coste apropiada para tu problema. Las opciones comunes son el error cuadrático medio para problemas de regresión, y la entropía cruzada para problemas de clasificación.

Regularización: Si la red sufre de overfitting, puedes considerar técnicas de

regularización como el abandono (dropout), la regularización L1/L2 o la parada temprana.

Diseñar una red neuronal requiere experimentación y ajuste de los hiperparámetros. Es un proceso iterativo de probar diferentes configuraciones, evaluar el rendimiento del modelo, ajustar los hiperparámetros y volver a probar. El objetivo es encontrar el equilibrio adecuado que permita a la red aprender de los datos sin overfitting y que
generalice bien a nuevos datos.

2.1.7.- Arquitecturas

La arquitectura de una red neuronal se refiere a la disposición de las neuronas y las capas en la red, así como a la forma en que están conectadas entre sí. Las diferentes arquitecturas se utilizan para diferentes tareas de aprendizaje automático, y cada una tiene sus propias características y ventajas. Aquí te presento algunas arquitecturas comunes:

Perceptron multicapa (MLP): Esta es la forma más simple de una red neuronal artificial. Un MLP consta de al menos tres capas de nodos: una capa de entrada, una capa oculta y una capa de salida. Cada nodo en una capa está conectado a todos los nodos en la capa siguiente, y la información se mueve en una dirección, desde la entrada hasta la salida.

Redes neuronales convolucionales (CNN): Estas redes son especialmente eficaces para tratar con datos que tienen una estructura de cuadrícula espacial, como las imágenes. Las CNN utilizan capas convolucionales que aplican un conjunto de filtros a la entrada, capas de pooling que reducen la dimensionalidad, y capas completamente conectadas que generan la salida. Redes neuronales recurrentes (RNN): Estas redes son útiles para tratar con datos secuenciales o de series temporales, como el texto o las series temporales. Las RNN tienen conexiones de retroalimentación que les permiten procesar secuencias de entradas, manteniendo la información de las entradas anteriores en la secuencia.

Long Short-Term Memory (LSTM): Esta es una variante de las RNN diseñada para mitigar el problema del desvanecimiento del gradiente que se produce en las RNN al tratar con secuencias largas. Las LSTM utilizan "puertas" que controlan la información que se mantiene y la que se descarta en cada paso de la secuencia.

Redes Generativas Adversarias (GANs): Las GANs constan de dos partes, un generador que produce datos y un discriminador que intenta distinguir entre los datos reales y los generados. Las GANs se entrenan a través de un juego competitivo en el que el generador intenta engañar al discriminador y
el discriminador intenta no dejarse engañar.

Transformadores: Esta es una arquitectura más reciente que se utiliza principalmente para el procesamiento del lenguaje natural. Los transformadores utilizan la atención de múltiples cabezas para prestar atención a diferentes partes de la entrada simultáneamente, lo que les permite manejar dependencias a largo plazo en los datos.

Estas son solo algunas de las arquitecturas de redes neuronales más comunes. Cada una de ellas tiene variaciones y extensiones, y se están desarrollando
continuamente nuevas arquitecturas para abordar diferentes desafíos en el aprendizaje automático y el deep learning.

2.2. Implementación de redes neuronales profundas en Python

2.2.1.- Implementación en Tensorflow

TensorFlow es una de las bibliotecas de aprendizaje automático más utilizadas para construir, entrenar y desplegar redes neuronales profundas. Aquí hay un ejemplo de cómo implementar un modelo de red neuronal profunda utilizando TensorFlow y su API de alto nivel, Keras:

Primero, necesitas instalar Tensorflow. Puedes hacerlo usando

pip: `pip install tensorflow`

A continuación, importa las bibliotecas necesarias:

```
import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense
```

Vamos a crear un modelo simple de red neuronal profunda para clasificación binaria. Supongamos que nuestros datos de entrada tienen 8 características (por ejemplo, podrían ser un conjunto de datos de pacientes con diabetes con varias medidas médicas), y estamos tratando de predecir una etiqueta binaria (por ejemplo, si el paciente tiene o no diabetes).

```
# Crear el modelo Sequential

model = Sequential()



# Añadir la primera capa oculta

model.add(Dense(16, input_dim=8, activation='relu'))
# Añadir la segunda capa oculta

model.add(Dense(8, activation='relu'))



# Añadir la capa de salida

model.add(Dense(1, activation='sigmoid'))
```

Una vez que hemos construido la arquitectura de la red, necesitamos compilar el modelo. Aquí es donde especificamos la función de pérdida que queremos minimizar, el optimizador que queremos utilizar para hacerlo, y cualquier métrica que queramos seguir durante el entrenamiento:

```
# Compilar el modelo

model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Ahora, asumamos que `X_train` e `y_train` son tus datos de entrenamiento y las etiquetas correspondientes. Entonces, puedes entrenar la red neuronal con estos datos:

```
# Entrenar el modelo

model.fit(X_train, y_train, epochs=150, batch_size=10)
```

Por último, puedes evaluar el rendimiento del modelo en tus datos de prueba (`X_test` e `y_test`):

```
# Evaluar el modelo

_, accuracy = model.evaluate(X_test, y_test)

print('Accuracy: %.2f' % (accuracy*100))
```

Este es un ejemplo simple de cómo puedes construir, entrenar y evaluar una red neuronal profunda con TensorFlow y Keras. Recuerda que hay muchos otros detalles
a tener en cuenta al trabajar con redes neuronales, como el preprocesamiento de datos, la regularización, el ajuste de hiperparámetros y el manejo del sobreajuste.

2.2.2.- Implementación en Keras

Keras es una biblioteca de Python de alto nivel para el aprendizaje automático profundo. Keras se puede utilizar de forma independiente o como interfaz para otras bibliotecas de aprendizaje profundo, como TensorFlow. En la versión 2.0 y superiores, Keras está completamente integrado en TensorFlow, por lo que los códigos suelen ser muy similares.

MODULO III – REDES NEURONALES CONVOLUCIONALES Y VISION COMPUTACIONAL

3.1.- Fundamentos de redes convolucionales

3.1.1.- Operación de convolución

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal diseñada para procesar datos con una estructura de cuadrícula, como una imagen, que se interpreta en términos de píxeles en altura y anchura, y que a menudo tiene múltiples canales de color (por ejemplo, rojo, verde y azul).

La operación clave en una CNN es la convolución, un tipo especial de operación lineal. La convolución es esencialmente el proceso de aplicar un filtro ('kernel') a los datos de entrada para obtener alguna salida que representa ciertas características

de los datos de entrada.

Aquí te dejo un desglose detallado de cómo funciona la convolución en las CNN:

- Filtros o kernels: Un filtro es una pequeña matriz de pesos. Este filtro se aplica a partes de la imagen de entrada de tamaño correspondiente. Por ejemplo, si el filtro es de 3x3, entonces se aplica a partes de 3x3 de la imagen de entrada.
- Aplicación del filtro: Aplicar el filtro consiste en deslizar el filtro por toda la imagen y calcular el producto punto entre los pesos del filtro y los píxeles correspondientes de la imagen en cada posición. Este proceso genera un nuevo mapa de características (también conocido como mapa de activaciones) que representa las áreas de la imagen que activaron el filtro.
- Mapa de características: Los valores en el mapa de características indican cuánto la imagen de entrada se asemeja al filtro en cada posición. Por ejemplo, si estás usando un filtro que detecta bordes verticales, las áreas del mapa de características con valores altos corresponden a áreas de la imagen con bordes verticales fuertes.
- Varias características: En una CNN, se utilizan múltiples filtros, cada uno diseñado para detectar una característica diferente en la imagen de entrada. Esto permite que la CNN capture una variedad de aspectos de la imagen, como bordes, texturas, colores, etc.
- Capas de convolución profundas: En una CNN, las capas convolucionales se apilan una encima de la otra. Cada capa aprende a detectar características cada vez más complejas. Las primeras capas pueden aprender a detectar bordes, las siguientes capas pueden aprender a detectar partes de objetos (como una rueda de un coche), y las capas finales pueden aprender a detectar objetos completos (como un coche).

Así es como las CNN utilizan la operación de convolución para extraer automáticamente las características importantes de las imágenes, lo que las hace muy efectivas para tareas como la clasificación de imágenes.

3.1.2.- Capa de agrupamiento

En las redes neuronales convolucionales (CNNs), después de varias capas convolucionales que extraen características visuales de una imagen, a menudo se utiliza una capa de agrupamiento o 'pooling' (como se conoce en inglés). Esta capa tiene la función de reducir la dimensionalidad espacial (ancho x alto) de la entrada, lo que ayuda a disminuir la cantidad de parámetros y cálculos en la red, y por ende controla el sobreajuste.

Existen varios tipos de operaciones de agrupamiento, pero las más comunes son el max pooling y el average pooling.

> Max pooling: Esta es la forma más común de agrupamiento. En la operación de max pooling, se define una ventana de un tamaño específico (por ejemplo, 2x2 píxeles) que se desliza a través de la entrada. En cada paso, el valor máximo dentro de esa ventana se selecciona y se coloca en la nueva matriz de salida. Esto tiene el efecto de preservar las características más fuertes (las que tienen los valores más altos) mientras se reduce la dimensión. Average pooling: Funciona de manera similar al max pooling, pero en lugar de tomar el valor máximo dentro de la ventana, toma el promedio de los valores.
> Esto puede suavizar la imagen pero puede no preservar las características más fuertes como lo hace el max pooling.

En resumen, las capas de agrupamiento actúan como una forma de reducción de la dimensionalidad no lineal, ayudando a hacer que la representación sea aproximadamente invariante a pequeñas traslaciones, y a controlar el sobreajuste al limitar la complejidad del modelo.

3.1.3.- Capa de conexión completa

En una red neuronal convolucional (CNN), las capas de conexión completa, también conocidas como capas densas o fully connected en inglés, son aquellas en las que cada neurona en la capa está conectada a todas las neuronas en la capa anterior.

Las capas de conexión completa se suelen encontrar cerca del final de una arquitectura de red neuronal convolucional. Después de varias capas convolucionales y de agrupamiento que sirven para extraer características de nivel inferior y medio de los datos de entrada, los datos resultantes se aplanan en un vector unidimensional y se introducen en una o más capas de conexión completa.

Estas capas funcionan esencialmente como un clasificador. Las características extraídas por las capas convolucionales y de agrupamiento se combinan de manera compleja en las capas de conexión completa para hacer la clasificación final. Cada neurona en una capa de conexión completa calcula una función lineal de las salidas de todas las neuronas en la capa anterior, y luego aplica una función de activación no lineal a esa suma ponderada.

Por ejemplo, en una tarea de clasificación de imágenes, las primeras capas de una CNN podrían aprender a detectar bordes, texturas y colores, y las capas de conexión

completa al final de la red aprenderían a combinar estas características para reconocer y clasificar objetos completos como "gato" o "perro".

Por lo tanto, una capa de conexión completa en una red neuronal convolucional tiene la importante tarea de tomar las características de alto nivel aprendidas por las capas anteriores y utilizarlas para hacer la predicción final requerida por la tarea, ya sea una clasificación, una regresión u otra salida.

3.1.4.- Diseño de redes neuronales convolucionales

Diseñar una red neuronal convolucional (CNN) es un proceso de varios pasos que implica la elección y configuración de diferentes tipos de capas y parámetros. Aunque hay muchas variaciones posibles, aquí se muestra un proceso general y un ejemplo de cómo podrías diseñar una CNN básica para una tarea de clasificación de imágenes.

1. Capa de entrada: Primero, necesitas una capa de entrada que tome tus datos. Para una tarea de clasificación de imágenes, esto sería una matriz de píxeles de la imagen.

2. Capas convolucionales: Después de la capa de entrada, las capas convolucionales comienzan a extraer características de las imágenes. Estas capas utilizan un conjunto de filtros que se aplican a partes de la imagen para detectar patrones como bordes, colores y texturas. Típicamente, las primeras capas convolucionales detectarán características de nivel inferior (como bordes), mientras que las capas posteriores detectarán características de nivel superior (como formas de partes de objetos).

3. Capas de agrupamiento: Después de cada una o varias capas convolucionales, generalmente se agrega una capa de agrupamiento para reducir la dimensionalidad espacial de la imagen, lo que ayuda a evitar el sobreajuste y a reducir la cantidad de cálculos necesarios en las capas posteriores.

4. Capas de normalización (opcional): Algunas CNNs también incluyen capas de normalización, como la normalización por lotes, que pueden ayudar a acelerar el entrenamiento y mejorar el rendimiento del modelo.

5. Capas de conexión completa: Después de varias capas convolucionales y de agrupamiento, los datos se aplanan en un vector y se introducen en una o más capas de conexión completa. Estas capas funcionan como un clasificador que toma las características extraídas de las capas anteriores y las combina para hacer la clasificación final.

6. Capa de salida: Finalmente, necesitarás una capa de salida que produzca las predicciones de tu red. Para una tarea de clasificación de imágenes, esta sería una capa de conexión completa con una neurona por clase, utilizando una función de activación softmax para producir una distribución de probabilidad sobre las clases.

Aquí hay un ejemplo de cómo podría verse este diseño en código, utilizando la biblioteca Keras en Python:

```
from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense



# Inicializa el modelo

model = Sequential()



# Agrega la primera capa convolucional

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))



# Agrega la primera capa de agrupamiento

model.add(MaxPooling2D(pool_size=(2, 2)))



# Agrega la segunda capa convolucional

model.add(Conv2D(64, (3, 3), activation='relu'))



# Agrega la segunda capa de agrupamiento

model.add(MaxPooling2D(pool_size=(2, 2)))



# Aplana los datos para las capas de conexión completa

model.add(Flatten())
```

```
# Agrega una capa de conexión completa

model.add(Dense(128, activation='relu'))



# Agrega la capa de salida

model.add(Dense(num_classes, activation='softmax'))
```

En este ejemplo, estamos asumiendo que nuestras imágenes de entrada son de 64x64 píxeles con 3 canales de color (RGB), y que tenemos un número variable de clases (`num_classes`).

Recuerda que esto es solo un ejemplo básico. Hay muchas variaciones posibles en el diseño de las CNNs, dependiendo de las necesidades específicas de tu tarea. Por ejemplo, podrías agregar más capas, cambiar el tamaño de los filtros, utilizar diferentes tipos de agrupamiento o funciones de activación, agregar capas de dropout para la regularización, etc.

3.1.5.- Arquitecturas

Las redes neuronales convolucionales (CNNs) han evolucionado significativamente a lo largo del tiempo. Hay varias arquitecturas de CNN notables que han sido fundamentales en el avance de los algoritmos de aprendizaje profundo, cada una introduciendo conceptos y técnicas nuevas y valiosas. Aquí te dejo algunos ejemplos de arquitecturas de CNN notables:

LeNet-5 (1998): Desarrollada por Yann LeCun, LeNet-5 es reconocida como la primera CNN exitosa. Se utilizó principalmente para tareas de reconocimiento de dígitos manuscritos.
AlexNet (2012): Desarrollada por Alex Krizhevsky, ganó la competencia ImageNet ILSVRC-2012 por un margen significativo y demostró que las CNNs son capaces de producir resultados de vanguardia en la clasificación de imágenes. Utilizó técnicas como el ReLU como función de activación, dropout para la regularización, y entrenamiento en GPU para acelerar el aprendizaje.
VGGNet (2014): VGGNet, desarrollada por el Visual Geometry Group de la Universidad de Oxford, destacó por su simplicidad, utilizando solo capas convolucionales de 3x3 y capas de pooling de 2x2. A pesar de su simplicidad, alcanzó resultados de vanguardia en la competencia ImageNet ILSVRC-2014.
GoogLeNet / Inception (2014): Ganadora del concurso ILSVRC-2014, introdujo

el módulo de "Inception", que permite a la red elegir el tamaño del filtro que desea utilizar en cada capa, mejorando la eficiencia computacional. ResNet (2015): ResNet, desarrollada por Kaiming He y otros en Microsoft Research, introdujo el concepto de "conexiones residuales" para permitir el entrenamiento de redes mucho más profundas. La red "Residual Network" ganó la competencia ILSVRC-2015 y ha sido ampliamente adoptada en la comunidad de aprendizaje profundo.

DenseNet (2017): En lugar de sumar las salidas de las conexiones de salto como en ResNet, DenseNet las concatena. Esto resulta en redes que son más compactas y eficientes.

EfficientNet (2019): EfficientNet, desarrollada por Google AI Research, utiliza una nueva técnica de escalado compuesto para escalar de manera uniforme la profundidad, la anchura y la resolución de la red, logrando un rendimiento superior con menos parámetros y menor coste computacional.

Estas son solo algunas de las arquitecturas de CNN más notables. Cada una ha aportado algo nuevo y valioso a la disciplina, y han servido de base para muchas de las redes neuronales convolucionales que se utilizan hoy en día.

3.2.- Implementación de redes convolucionales en Python

Aquí tienes un ejemplo de cómo podrías implementar una red neuronal convolucional (CNN) básica para la clasificación de imágenes en Python usando Keras, que es una biblioteca de aprendizaje profundo de alto nivel construida sobre TensorFlow.

```
from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense


# Inicializa el modelo

model = Sequential()


# Agrega la primera capa convolucional
```

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))


# Agrega la primera capa de agrupamiento

model.add(MaxPooling2D(pool_size=(2, 2)))


# Agrega la segunda capa convolucional

model.add(Conv2D(64, (3, 3), activation='relu'))
# Agrega la segunda capa de agrupamiento

model.add(MaxPooling2D(pool_size=(2, 2)))


# Aplana los datos para las capas de conexión completa

model.add(Flatten())


# Agrega una capa de conexión completa

model.add(Dense(128, activation='relu'))


# Agrega la capa de salida

model.add(Dense(num_classes, activation='softmax'))
```

En este código:

- **Estamos utilizando el modelo secuencial de Keras, que permite agregar capas una por una en el orden en que deseamos que los datos fluyan.**
- **Cada `Conv2D` es una capa convolucional. El primer argumento es el número de filtros que la capa tendrá. `(3, 3)` es el tamaño de esos filtros. `activation='relu'` especifica que estamos utilizando la función de activación ReLU.**
- **`MaxPooling2D(pool_size=(2, 2))` agrega una capa de agrupamiento que reduce la dimensionalidad de la imagen.**

- `Flatten()` **aplana la imagen en un vector para que pueda ser procesada por las capas densas (conexión completa).**
- **Las capas** `Dense` **son las capas de conexión completa. El primer argumento es el número de neuronas en la capa. La capa final tiene** `num_classes` **neuronas (una por cada clase) y utiliza la función de activación softmax para producir una distribución de probabilidad sobre las clases.**

Una vez que hayas definido tu modelo, puedes compilarlo y luego entrenarlo con tus datos. Aquí tienes un ejemplo de cómo podrías hacerlo:

```
# Compila el modelo
model.compile(loss='categorical_crossentropy',

              optimizer='adam',

              metrics=['accuracy'])



# Entrena el modelo

model.fit(x_train, y_train,

          validation_data=(x_test, y_test),

          epochs=10,

          batch_size=32)
```

En este código, `categorical_crossentropy` **es la función de pérdida que se utiliza para la clasificación multiclase,** `adam` **es el optimizador, y** `accuracy` **es la métrica que queremos seguir durante el entrenamiento.**

El método `fit` **se utiliza para entrenar el modelo.** `x_train` **y** `y_train` **son los datos y las etiquetas de entrenamiento, respectivamente, y** `x_test` **y** `y_test` **son los datos y las etiquetas de validación.** `epochs=10` **especifica que queremos entrenar el modelo durante 10 épocas, y** `batch_size=32` **especifica el número de ejemplos que se**
utilizan en cada actualización de los pesos.

3.3.- Aplicaciones prácticas de redes convolucionales en Python

3.3.1.- Preprocesamiento y gestión de los datos

El preprocesamiento y la gestión de datos son componentes esenciales al trabajar con redes neuronales convolucionales (CNNs). Los datos de entrada para una CNN generalmente son imágenes, pero también pueden ser otros tipos de datos multimensionales. Aquí hay algunas técnicas comunes de preprocesamiento y gestión de datos:

Normalización de los datos: La normalización generalmente implica escalar los valores de los píxeles de las imágenes a un rango de 0 a 1. Esto se hace típicamente dividiendo cada valor de píxel por 255, que es el valor máximo que puede tener un píxel (para las imágenes en escala de grises). Esto ayuda a la red a aprender y converger más rápido.
Redimensionamiento de las imágenes: Las CNNs requieren que todas las imágenes tengan las mismas dimensiones. Por lo tanto, es común redimensionar todas las imágenes a un tamaño estándar antes de usarlas como entrada para la red.
Augmentación de datos: Para aumentar la cantidad de datos de entrenamiento y hacer que el modelo sea más robusto a las variaciones, se puede utilizar la augmentación de datos. Esto puede incluir varias transformaciones de las imágenes, como rotaciones, traslaciones, zoom, volteos, recortes, cambios de brillo, etc.
One-hot encoding: Si estás haciendo clasificación multiclase, necesitas convertir las etiquetas de clase en una representación de one-hot encoding. Esto significa que si tienes, por ejemplo, 3 clases, en lugar de tener una sola etiqueta de clase (por ejemplo, 0, 1, 2), tendrías un vector de longitud 3 para cada etiqueta, donde el índice correspondiente a la clase es 1 y todos los demás son 0 (por ejemplo, la clase 0 sería [1, 0, 0], la clase 1 sería [0, 1, 0], y la clase 2 sería [0, 0, 1]).
División de los datos: Es importante dividir tus datos en conjuntos de entrenamiento, validación y prueba. El conjunto de entrenamiento es para entrenar el modelo, el conjunto de validación es para ajustar los hiperparámetros y hacer selección de modelo, y el conjunto de prueba es para evaluar el rendimiento final del modelo.

A continuación, un ejemplo de cómo se podría implementar parte de esto en Python con la biblioteca Keras:

```
from keras.preprocessing.image import ImageDataGenerator
```

```
# Crea un generador de datos de imágenes con augmentación de datos
datagen = ImageDataGenerator(
    rescale=1./255, # Normaliza los datos
        rotation_range=20, # Rota las imágenes aleatoriamente hasta 20 grados
    width_shift_range=0.2, # Desplaza las imágenes aleatoriamente horizontalmente
    height_shift_range=0.2,# Desplaza las imágenes aleatoriamente verticalmente
    horizontal_flip=True, # Voltea las imágenes aleatoriamente horizontalmente
    validation_split=0.2 # Divide los datos en entrenamiento y validación
)

# Generador para los datos de entrenamiento
train_generator = datagen.flow_from_directory(
    'data/train',
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

# Generador para los datos de validación
validation_generator = datagen.flow_from_directory(
    'data/train',
    target_size=(64, 64),
    batch_size=32,
```

```
    class_mode='categorical',
    subset='validation'
)
```

Este código crea un generador de imágenes que realiza una serie de transformaciones de augmentación de datos en las imágenes (normalización, rotación, desplazamiento, volteo). Luego, se crean dos generadores, uno para los datos de entrenamiento y otro para los datos de validación, a partir de las imágenes en el directorio 'data/train'. Las imágenes se redimensionan a 64x64 y las etiquetas se codifican en one-hot.

3.3.2.- Implementación en Tensorflow

Aquí tienes un ejemplo de cómo podrías implementar una red neuronal convolucional (CNN) básica para la clasificación de imágenes en Python usando TensorFlow y su API de alto nivel Keras:

```
import tensorflow as tf
from tensorflow.keras import layers

# Inicializa el modelo
model = tf.keras.Sequential()

# Agrega la primera capa convolucional
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))

# Agrega la primera capa de agrupamiento
model.add(layers.MaxPooling2D((2, 2)))
```

```
# Agrega la segunda capa convolucional

model.add(layers.Conv2D(64, (3, 3), activation='relu'))



# Agrega la segunda capa de agrupamiento

model.add(layers.MaxPooling2D((2, 2)))



# Aplana los datos para las capas de conexión completa

model.add(layers.Flatten())



# Agrega una capa de conexión completa

model.add(layers.Dense(128, activation='relu'))
# Agrega la capa de salida

model.add(layers.Dense(10, activation='softmax'))



# Compila el modelo

model.compile(optimizer='adam',

              loss=tf.keras.losses.SparseCategoricalCrossentropy(),

              metrics=['accuracy'])



# Entrena el modelo

model.fit(train_images, train_labels, epochs=10,

          validation_data=(test_images, test_labels))
```

En este código:

- `Conv2D` es una capa convolucional. El primer argumento es el número de filtros que la capa tendrá. `(3, 3)` es el tamaño de esos filtros. `activation='relu'` especifica que estamos utilizando la función de activación ReLU.
- `MaxPooling2D` es una capa de agrupamiento que reduce la dimensionalidad de la imagen.
- `Flatten` aplana la imagen en un vector para que pueda ser procesada por las capas densas (completamente conectadas).
- `Dense` es una capa completamente conectada. El primer argumento es el número de neuronas en la capa. La capa final tiene 10 neuronas (una para cada clase de dígitos 0-9) y utiliza la función de activación softmax para producir una distribución de probabilidad sobre las clases.
- `optimizer='adam'` especifica que estamos utilizando el optimizador Adam, que es un algoritmo de optimización popular para el aprendizaje profundo. ● `loss=tf.keras.losses.SparseCategoricalCrossentropy()` especifica que estamos utilizando la entropía cruzada categórica como función de pérdida, que es apropiada para problemas de clasificación multiclase.
- `metrics=['accuracy']` significa que queremos monitorear la precisión durante el entrenamiento.
- `model.fit` entrena el modelo para un número dado de épocas (iteraciones en un conjunto de datos).

Nota: Este código asume que ya tienes tus datos de entrenamiento y prueba en las variables `train_images`, `train_labels`, `test_images` y `test_labels`. También asume que tus imágenes ya han sido preprocesadas (es decir, redimensionadas, normalizadas, etc.) y que tus etiquetas han sido codificadas adecuadamente para un problema de clasificación multiclase.

3.3.3.- Implementación en Keras

3.3.4. Implementación en PyTorch

3.3.5.- Entrenamiento y evaluación del desempeño

El entrenamiento y la evaluación del rendimiento de una red neuronal convolucional (CNN) es un proceso multifacético que involucra varias etapas:

1. Entrenamiento: Una vez que se ha definido y compilado una CNN, se puede entrenar utilizando datos de entrenamiento. El proceso de entrenamiento implica la alimentación de los datos de entrada a la red y el ajuste de los pesos y sesgos de la

red para minimizar la función de pérdida. Aquí está un ejemplo de cómo puedes entrenar una red en TensorFlow:

```
# Asume que `model` es una CNN que ya ha sido definida y compilada

# Asume que `train_images` y `train_labels` son tus datos de entrenamiento y las correspondientes etiquetas

model.fit(train_images, train_labels, epochs=10)
```

El argumento `epochs` indica cuántas veces la red debe iterar sobre todo el conjunto de datos de entrenamiento. En cada época, la red actualiza sus pesos y sesgos en un intento de minimizar la función de pérdida.

2. Validación: Durante el entrenamiento, es común también proporcionar un conjunto de datos de validación a la red. Estos datos no se utilizan para ajustar los pesos y sesgos de la red, sino que sirven para dar una indicación de cómo la red está desempeñándose en datos que no ha visto antes. Puedes hacer esto añadiendo un argumento `validation_data` a la función `fit`:

```
# Asume que `val_images` y `val_labels` son tus datos de validación y las correspondientes etiquetas

model.fit(train_images, train_labels, epochs=10, validation_data=(val_images, val_labels))
```

Durante cada época, después de que la red haya sido ajustada utilizando los datos de entrenamiento, se evaluará su rendimiento en los datos de validación.

3. Evaluación: Después de que la red ha sido entrenada, puedes evaluar su rendimiento en un conjunto de datos de prueba separado:

```
# Asume que `test_images` y `test_labels` son tus datos de prueba y las correspondientes etiquetas

test_loss, test_acc = model.evaluate(test_images, test_labels)
```

La función `evaluate` devuelve la pérdida de prueba y la precisión de prueba (o cualquier otra métrica que hayas especificado durante la compilación del modelo). Estos valores te dan una indicación de cómo se espera que se desempeñe la red en

datos nuevos y no vistos.

4. Ajuste de Hiperparámetros: Basado en el rendimiento en los conjuntos de validación y prueba, puedes optar por ajustar los hiperparámetros de la red (por ejemplo, la tasa de aprendizaje, el número de capas, el número de neuronas por capa, etc.) para intentar mejorar su rendimiento.

5. Predicción: Finalmente, una vez que estás satisfecho con el rendimiento de la red, puedes usarla para hacer predicciones en datos nuevos:

```
# Asume que `new_images` son nuevas imágenes que quieres clasificar

predictions = model.predict(new_images)
```

La función `predict` devuelve las predicciones de la red para las imágenes proporcionadas.

Ten en cuenta que cada uno de estos pasos puede requerir una considerable experimentación y ajuste para obtener buenos resultados. Además, siempre es
importante garantizar que tus datos estén bien preparados y que la tarea que estás tratando de resolver sea adecuada para el tipo de red que estás utilizando.

3.3.6.- Aplicaciones de Visión Computacional

Las redes neuronales convolucionales (CNN) han revolucionado el campo de la visión por computadora, ofreciendo un rendimiento sin precedentes en una variedad de tareas. Algunas de las aplicaciones más destacadas incluyen:

1. Clasificación de imágenes: La tarea de clasificación de imágenes consiste en asignar a una imagen dada una etiqueta de una serie de categorías predefinidas. CNN se utiliza comúnmente para esta tarea debido a su capacidad para aprender automáticamente las características relevantes de las imágenes.

2. Detección de objetos: En la detección de objetos, el objetivo no es solo clasificar una imagen, sino también localizar dónde en la imagen se encuentran los objetos. Esto se logra a menudo utilizando una CNN para clasificar diferentes partes de la imagen.

3. Segmentación semántica: La segmentación semántica es la tarea de clasificar cada píxel en una imagen, es decir, asignar a cada píxel una etiqueta de una serie de categorías predefinidas. Este tipo de tarea se puede lograr con una arquitectura especial de CNN llamada Fully Convolutional Network (FCN).

4. Reconocimiento facial: CNN es comúnmente utilizado en aplicaciones de reconocimiento facial, donde el objetivo es identificar o verificar la identidad de una persona a partir de una imagen de su rostro.

5. Análisis de imágenes médicas: Las CNN también se utilizan en el análisis de imágenes médicas, como la detección de tumores en imágenes de resonancia magnética, la clasificación de células de cáncer en imágenes microscópicas o la detección de enfermedades oculares en imágenes de la retina.

6. Reconocimiento de texto y escenas: Las CNN pueden utilizarse para reconocer texto en imágenes, lo que puede ser útil para una variedad de aplicaciones, como el reconocimiento de placas de matrícula o la transcripción de texto en imágenes de documentos.

7. Análisis de video: Las CNN, a menudo en combinación con otros tipos de redes neuronales como las redes neuronales recurrentes (RNN), pueden utilizarse para analizar secuencias de video, por ejemplo, para la detección y seguimiento de objetos, el reconocimiento de acciones o la predicción de eventos futuros.

Estas son solo algunas de las muchas aplicaciones de CNN en visión por computadora. La flexibilidad y el rendimiento de las CNN las hacen adecuadas para una amplia gama de tareas de visión por computadora.

3.3.7.- Clasificación de imágenes

La clasificación de imágenes es una de las aplicaciones más comunes de las redes neuronales convolucionales (CNN). Las CNN son particularmente adecuadas para la clasificación de imágenes debido a su capacidad para operar directamente en los datos de imagen en bruto y extraer características jerárquicas, que pueden ser utilizadas para la clasificación.

Aquí se muestra un ejemplo de cómo podrías usar una CNN para clasificar imágenes en TensorFlow y Keras:

```
import tensorflow as tf
```

```
from tensorflow.keras import layers, models

# Definir la arquitectura de la CNN
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Añadir capas densas para clasificación
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid')) # Asume una tarea de clasificación binaria

# Compilar el modelo
model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])

# Entrenar el modelo con los datos de entrenamiento
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(val_images, val_labels))
```

Este código define una CNN con tres capas convolucionales, seguidas de capas de agrupamiento máximo. Después de las capas convolucionales, la imagen se aplana y se pasa a través de una capa densa antes de llegar a la capa de salida. La capa de salida utiliza una función de activación sigmoidea, lo que indica que estamos realizando una tarea de clasificación binaria.

Este código también entrena la red utilizando un conjunto de datos de entrenamiento (asumiendo que `train_images` y `train_labels` contienen tus datos de entrenamiento y las etiquetas correspondientes) y valida el modelo en un conjunto de datos de validación (`val_images` y `val_labels`).

Cabe señalar que el preprocesamiento de las imágenes y las etiquetas (por ejemplo, normalización, codificación de etiquetas, etc.) no está incluido en este ejemplo y tendría que realizarse antes del entrenamiento.

MODULO IV – REDES NEURONALES RECURRENTES Y PROCESAMIENTO DEL LENGUAJE NATURAL

Fundamentos de redes recurrentes

- Recurrencia
- Intuición práctica
- Gradiente desvaneciente (vanishing gradient)
- Memoria de corto-largo plazo
- LSTM y otras arquitecturas

Implementación de redes recurrentes en Python

Aplicaciones prácticas de redes recurrentes en Python

- Preprocesamiento y gestión de los datos
- Implementación en Keras
- Entrenamiento y evaluación del desempeño
- Aplicaciones de Procesamiento de Lenguaje Natural
- Clasificación de sentimientos

MODULO V – MODELOS GENERATIVOS Y AUTOCODIFICADORES

Autocodificadores

- Intuición general de los autocodificadores
- Arquitectura de los autocodificadores
- Deep autoencoders
- Denoising autoencoders
- Sparse autoencoders
- Variational autoencoders
- Otros autoencoders
- Implementación de autocodificadores en Python

Modelos generativos

- Intuición general de las redes adversarias generativas
- Arquitectura de las redes adversarias generativas
- Proceso de entrenamiento
- Implementación de modelos generativos en Python

Aplicaciones prácticas de autocodificadores y modelos generativos en Python

• Taller de aplicaciones prácticas

MODULO VI – PROYECTO FIN DE CURSO

DEEP LEARNING

El deep learning (aprendizaje profundo) es una rama de la inteligencia artificial y el aprendizaje automático que se enfoca en el desarrollo y entrenamiento de modelos de redes neuronales artificiales altamente complejas. Su objetivo principal es permitir que las máquinas aprendan automáticamente patrones y representaciones complejas de datos sin intervención humana directa. El término "profundo" se refiere a la utilización de múltiples capas de neuronas artificiales para crear un modelo jerárquico y de mayor complejidad.

El deep learning se inspira en la estructura y funcionamiento del cerebro humano y se basa en redes neuronales artificiales, que son sistemas de procesamiento de información compuestos por capas de nodos interconectados. Cada nodo, también conocido como neurona artificial, realiza operaciones matemáticas simples y toma decisiones en función de las entradas recibidas. Estas redes se diseñan para aprender y mejorar con la experiencia, ajustando automáticamente sus parámetros internos en función de los datos proporcionados durante el proceso de entrenamiento.
Las etapas fundamentales del Deep Learning son:

Preprocesamiento de datos: Antes de entrenar un modelo de Deep Learning, es esencial realizar un procesamiento previo (preprocesamiento) de los datos. Esto implica limpiar y preparar los datos de manera que estén listos para ser utilizados en el entrenamiento. El preprocesamiento puede incluir pasos como normalización, escalamiento, codificación de variables categóricas, manejo de valores faltantes y división del conjunto de datos en datos de entrenamiento, validación y prueba.
Definición de la arquitectura del modelo: En esta etapa, se define la estructura del modelo de Deep Learning. Esto incluye el tipo y número de capas que compondrán la red neuronal, la cantidad de neuronas en cada capa y cómo estarán conectadas entre sí. La elección de la arquitectura del modelo puede variar según el problema que se esté abordando, y es una parte crucial del proceso, ya que afectará directamente el rendimiento del modelo.
Entrenamiento del modelo: Una vez que la arquitectura del modelo está

definida, el siguiente paso es entrenar el modelo utilizando datos de entrenamiento. Durante el entrenamiento, el modelo ajusta sus parámetros (pesos y sesgos) para minimizar la función de pérdida y mejorar su rendimiento en la tarea específica. Esto se logra mediante el proceso de alimentación hacia adelante y retroalimentación (forward propagation y backpropagation) que se mencionó anteriormente. El entrenamiento se repite durante varias épocas hasta que el modelo converja a una solución óptima o hasta que se alcance un criterio de parada.

Validación del modelo: Una vez que el modelo está entrenado, es importante evaluar su rendimiento en un conjunto de datos diferente al utilizado en el entrenamiento, llamado conjunto de validación. Esto ayuda a asegurarse de que el modelo no se haya sobreajustado (overfitting) a los datos de entrenamiento y que pueda generalizar bien a nuevos datos. La validación también puede ayudar a ajustar los hiperparámetros del modelo para obtener un mejor rendimiento.

Prueba del modelo: Finalmente, después de haber entrenado y validado el modelo, se evalúa su rendimiento en un conjunto de datos completamente independiente, llamado conjunto de prueba. Este conjunto de datos debe estar separado de los datos de entrenamiento y validación y servirá como una evaluación final del rendimiento del modelo en datos no vistos previamente. Es importante asegurarse de que el conjunto de prueba sea representativo de los datos que el modelo encontrará en situaciones reales.

El proceso de entrenamiento del modelo implica tres etapas fundamentales:

Alimentación hacia adelante (forward propagation): Los datos de entrenamiento son introducidos en la red, y las salidas calculadas se comparan con las salidas reales conocidas. Las diferencias entre las salidas predichas y las reales se expresan a través de una función de pérdida, que mide qué tan bien se está desempeñando la red en el problema dado.

Retroalimentación (backpropagation): A través de algoritmos de optimización, como el descenso de gradiente, se ajustan los pesos y las conexiones entre las neuronas para minimizar la función de pérdida. Esta es la fase de aprendizaje donde la red "aprende" al adaptar sus conexiones y parámetros para mejorar su rendimiento en tareas específicas.

Actualización de pesos y repetición: El proceso de ajuste de los pesos se repite muchas veces (a través de múltiples épocas) hasta que la red alcanza un nivel de rendimiento satisfactorio en el conjunto de entrenamiento.

1.- Parametros
Los parámetros son valores que el modelo de aprendizaje profundo aprende directamente durante el proceso de entrenamiento. Estos son los pesos y los sesgos de las neuronas en las diferentes capas de la red neuronal. El objetivo del entrenamiento es ajustar estos parámetros para que el modelo pueda hacer predicciones precisas en los datos de entrenamiento y generalizar bien a datos no vistos.

Por ejemplo en una red neuronal profunda, los parámetros incluyen los pesos y sesgos de cada neurona en cada capa. Si tienes una red neuronal con 3 capas ocultas, habrá un conjunto de parámetros para la primera capa oculta, otro conjunto para la segunda capa oculta y así sucesivamente. Estos parámetros se actualizan a medida que el modelo se entrena mediante algoritmos de optimización como el descenso gradiente

2.- Hiperparametros
Son configuraciones que se establecen antes del proceso de entrenamiento y afectan la arquitectura y el comportamiento del modelo de aprendizaje profundo. Estos no son aprendidos durante el entrenamiento sino que se determinan antes de comenzar el proceso
de aprendizaje. La elección adecuada de hiperparametros puede influir en la capacidad del modelo para aprender y generalizar de manera efectiva

Ejemplos de hiperparametros
.- Tasa de aprendizaje.
Controla el tamaño de los pesos que toma el algoritmo de optimización durante el proceso de ajuste de los parámetros
.-Número de capas ocultas en la red neuronal
Determina la profundidad de la red neuronal
.- Número de neuronas por capa
Controla el ancho de la red
.- Tamaño del lote (batch size)
Especifica la cantidad de ejemplos que se utilizan en cada paso del algoritmo de optimización
.- Número de épocas de entrenamiento

Define la cantidad de veces que el modelo recorre todo el conjunto de entrenamiento

Es importante seleccionar adecuadamente los hiperparametros ya que una mala elección puede llevar a problemas como el sobreajuste o bajo rendimiento del modelo

En resumen, los parámetros son aprendidos directamente por el modelo durante el entrenamiento y representan los pesos y sesgos de las neuronas, mientras que los hiperparametros son configuraciones establecidas antes del entrenamiento y afectan a la arquitectura y comportamiento del modelo