COLOMBIA POTENCIA DE LA VIDA

TIC

Actividad 3

Redes Neuronales recurrentes

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira

Una red neuronal recurrente (RNN)

Es un tipo de red neuronal artificial diseñada para procesar secuencias de datos, donde la salida de cada paso de tiempo se utiliza como entrada para el siguiente paso. A diferencia de las redes neuronales convolucionales (CNN) que se utilizan principalmente para datos de tipo imagen, las RNN son adecuadas para datos secuenciales como texto, audio y series temporales.

La arquitectura de una RNN incluye unidades recurrentes que mantienen un estado interno que se actualiza con cada nueva entrada. Estas unidades tienen conexiones recurrentes que les permiten tener memoria o recordar la información de entradas anteriores. Esto les permite capturar dependencias a largo plazo en secuencias de datos, lo que las hace útiles para tareas que implican comprensión del contexto, como traducción de idiomas, análisis de sentimientos, generación de texto, entre otros.

Las RNN son especialmente útiles cuando la longitud de las secuencias de entrada o salida puede variar, ya que pueden manejar secuencias de longitud variable. Sin embargo, las RNN tradicionales pueden sufrir del problema de desvanecimiento o explosión del gradiente, lo que dificulta el entrenamiento de dependencias a largo plazo. Para abordar este problema, se han desarrollado variantes de RNN como las redes LSTM (Long Short-Term Memory) y las redes GRU (Gated Recurrent Units), que han demostrado ser más efectivas para capturar dependencias a largo plazo en secuencias de datos.

Una red neuronal recurrente LSTM (Long Short-Term Memory)

Es un tipo especializado de red neuronal recurrente diseñada para abordar el problema del desvanecimiento del gradiente en las RNN estándar y para capturar dependencias a largo plazo en secuencias de datos. Las LSTM fueron propuestas por primera vez por Hochreiter y Schmidhuber en 1997 y han demostrado ser muy efectivas en una variedad de tareas de procesamiento de secuencias, como el procesamiento del lenguaje natural, la traducción automática, el reconocimiento de voz, entre otros.

A diferencia de las RNN tradicionales, que tienen problemas para retener información a largo plazo debido al desvanecimiento del gradiente, las LSTM incorporan una estructura de memoria especializada que les permite almacenar y acceder a información a lo largo del tiempo. Esto se logra a través de una unidad de memoria llamada "celda de memoria", que puede aprender qué información retener y qué información descartar en cada paso de tiempo.

Las LSTM tienen tres puertas principales que controlan el flujo de información dentro de la celda de memoria:

Puerta de olvido (Forget gate):
Decide qué información almacenada en la celda de memoria debe ser olvidada o descartada.

Puerta de entrada (Input gate):
Decide qué nueva información debe ser almacenada en la celda de memoria.

Puerta de salida (Output gate):
Decide qué información de la celda de memoria se debe utilizar para generar la salida de la red en ese paso de tiempo.

Una red neuronal recurrente LSTM (Long Short-Term Memory)

Estas puertas se activan mediante funciones de activación sigmoideas y operaciones de producto punto, lo que permite que la red aprenda a controlar el flujo de información de manera adaptativa según el contexto de la secuencia.

Las redes neuronales recurrentes LSTM son una poderosa arquitectura de red neuronal diseñada para manejar eficazmente el procesamiento de secuencias a largo plazo, lo que las hace especialmente útiles en tareas de procesamiento de lenguaje natural y otras aplicaciones que implican datos secuenciales.

Ejercicio de Clasificación de texto aplicando redes LSTM

Para esta actividad se realiza el mismo proceso inicial de la actividad anterior

- Descarga del conjunto de datos
- Preprocesamiento de las secuencias
- Tokenización y generación de secuencias
- Construcción y entrenamiento del modelo

Adicionalmente se Construyen y entrenan otro modelos con el objetivo de comparar los resultados del modelo que usa capas LSTM y el modelo que no las usa.

Construye un modelo de red neuronal recurrente (RNN) bidireccional con capas LSTM utilizando la biblioteca Keras.

1. Construcción del modelo

Se utiliza keras.Sequential() para inicializar un modelo secuencial. Luego se agregan capas al modelo utilizando la función add():

- keras.layers.Embedding: Esta capa convierte los enteros positivos (índices de palabras) en vectores densos de tamaño embedding_dim. Cada secuencia de entrada tiene una longitud de max_length.
- keras.layers.LSTM(64, return_sequences=True): Esta es una capa LSTM con 64 unidades de memoria (neuronas). El parámetro return_sequences=True hace que la capa devuelva la secuencia completa de salidas en lugar de solo la última salida.
- keras.layers.LSTM(32)): Otra capa LSTM, pero esta vez con 32 unidades de memoria. No especifica return_sequences=True, lo que significa que solo devuelve la última salida de la secuencia.
- keras.layers.Dense(6, activation='relu'): Capa densa con 6 neuronas y función de activación ReLU.
- keras.layers.Dense(1, activation='sigmoid'): Capa de salida con una neurona y función de activación sigmoide, que es adecuada para problemas de clasificación binaria.

Ver imagen

```
# Build the model
model = keras.Sequential([
    keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    keras.layers.LSTM(64, return_sequences=True),
    keras.layers.LSTM(32),
    keras.layers.Dense(6, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

2. Compilación del modelo

Se configura el proceso de entrenamiento del modelo utilizando el método compile(). Se especifica la función de pérdida (binary_crossentropy), el optimizador (adam) y las métricas (accuracy).

3. Resumen del modelo

Se imprime un resumen del modelo utilizando el método summary(), que muestra la arquitectura de la red neuronal y el número de parámetros entrenables.

```
# Setup the training parameters
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

# Print the model summary
model.summary()
```

4. Entrenamiento del modelo

El modelo se entrena utilizando el método fit(). Se ajusta a los datos de entrenamiento (padded) durante un número especificado de épocas (num_epochs). Los datos de validación (test_padded) se utilizan para evaluar el rendimiento del modelo durante el entrenamiento.

5. Visualización del historial de entrenamiento

Se genera un gráfico que muestra cómo evolucionan la pérdida y la precisión del modelo en el conjunto de entrenamiento y el conjunto de validación a lo largo de las épocas. Esto se hace utilizando la biblioteca Pandas para crear un DataFrame a partir del historial de entrenamiento y luego llamando al método plot() para visualizar los datos.

```
num_epochs = 10

# Train the model
history = model.fit(padded,
                    train_labels,
                    epochs=num_epochs,
                    validation_data=(test_padded, test_labels))

# Graficar el historial de entrenamiento:
pd.DataFrame(history.history).plot(grid=True)
```

Este código construye, compila, entrena y evalúa un modelo de red neuronal recurrente bidireccional con capas LSTM para clasificación binaria.

Preguntas:

1. ¿Qué tipo de modelo de red neuronal se construye en el código?
2. ¿Cuál es la función de activación utilizada en la capa de salida y por qué se elige esta función para este problema?
3. ¿Cuál es la diferencia entre una capa LSTM bidireccional y una capa LSTM unidireccional?
4. ¿Qué significan los parámetros return_sequences=True en la primera capa LSTM bidireccional?
5. ¿Qué función de pérdida se utiliza en el modelo y por qué es adecuada para problemas de clasificación binaria?
6. ¿Qué métrica se utiliza para evaluar el rendimiento del modelo durante el entrenamiento?
7. ¿Por qué se utilizan dos capas LSTM bidireccionales en el modelo?
8. ¿Qué hace la capa de embedding en el modelo y por qué es necesaria?

Ejercicios:

1. Modifica el número de unidades de memoria en las capas LSTM y observa cómo afecta el rendimiento del modelo.
2. Cambia la función de activación de la capa densa a 'tanh' y compara los resultados.
3. Entrena el modelo durante más épocas y observa si mejora el rendimiento en el conjunto de validación.
4. Experimenta con diferentes valores para los parámetros max_length y vocab_size y observa cómo afectan el tamaño y la eficiencia del modelo.
5. Agrega una capa de regularización, como Dropout, al modelo y observa si mejora la generalización del modelo.
6. Cambia la arquitectura del modelo agregando más capas ocultas y observa cómo afecta la capacidad de aprendizaje del modelo.

Estas preguntas y ejercicios deberían ayudar a los estudiantes a comprender mejor el código y a experimentar con diferentes aspectos del modelo para mejorar su comprensión y habilidades en el campo del aprendizaje profundo.

COLOMBIA POTENCIA DE LA VIDA

TIC

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira