LSTM

Contents

1 Introducción a RNN

Las Redes Neuronales Recurrentes son un tipo específico de arquitectura de red neuronal que está diseñado para manejar datos secuenciales, lo que significa que el orden de los datos es muy importante. Este tipo de red es fundamental en tareas donde la información anterior debe ser recordada o procesada junto a la información actual. A lo largo de la historia de la IA, las RNN han sido muy importantes a la hora de resolver problemas relacionados con el análisis de frecuencias como NLP, predicciones de series temporales, transcripción de voz a texto o modelado de vídeo y audio.

En las redes convencionales como las feedforward, la salida solo depende del input actual. Sin embargo, en las RNN, la salida depende tanto del input actual como de los inputs anteriores al introducir el concepto de memoria. Esta capacidad de recordar información previa es lo que hace que las RNN sean tan poderosas en la resolución de problemas secuenciales.

1.1 Problemas de las RNN estándar

Estas redes se enfrentan a un problema fundamental, el **Desvanecimiento del gradiente**. Estas redes se entrenan con un tipo de backpropagation, conocido como **Backpropagation Through Time (BPTT)**. En este algoritmo, el gradiente se propaga hacia atrás en el tiempo y se ajustan los pesos de la red para minimizar el error. El problema es que, a medida que el error se propaga hacia atrás en el tiempo, si las secuencias son muy largas, los gradientes se van desvaneciendo, por lo tanto, el impacto del aprendizaje de los primeros pasos de la secuencia es muy pequeño. Esto hace que las RNN tengan dificultades para aprender de elementos que están muy lejos en la secuencia. Cabe destacar que también puede ocurrir el efecto contrario, la **Explosión del gradiente**, donde los gradientes crecen exponencialmente a medida que se propagan hacia atrás en el tiempo, por lo que producen inestabilidad en el entrenamiento de la red.

Esta es la principal limitación de las RNNs estándar, que son eficaces a corto plazo. Pero cuando las secuencias son muy largas o la información relevante está muy separada en la secuencia, su rendimiento disminuye significativamente.

Este problema ha impulsado que se investigue y se desarrollen nuevas arquitecturas de RNN, como las **LSTM** y las **GRU**, que son capaces de recordar información a largo plazo y evitar el desvanecimiento del gradiente.

2 RNN

Estas redes permiten usar una variedad más amplia de estructuras tanto de entrada como de salida:

- **1 a 1**: Redes convencionales, un solo input y un solo output.
- **1 a N**: Por ejemplo: Generación de texto a partir de una imagen.
- **N a 1**: Por ejemplo: Análisis de sentimientos a partir de una reseña (de N palabras).
- **N a N**: Por ejemplo: Traducción de un idioma a otro. Tiene N palabras de entrada y M palabras de salida ya que la traducción puede tener más o menos palabras que el texto original por lo que técnicamente es `N a M`.

La función matemática que define una RNN es la siguiente:

$$h_t = f_\omega(h_{t-1}, x_t) \tag{1}$$

Donde:

- h_t es el estado oculto en el tiempo t.
- f_ω es la función de activación. Depende de los pesos ω de la red.
- h_{t-1} es el estado oculto en el instante anterior, $t-1$.
- x_t es el input en el tiempo t.

2.1 Vanilla RNN

Vanilla RNN es la forma más básica de RNN. La fórmula de la RNN estándar es la siguiente:

$$h_t = f_\omega(h_{t-1}, x_t) \tag{2}$$

$$h_t = \tanh(w_{hh}h_{t-1} + w_{xh}x_t) \tag{3}$$

Donde:

- w_{hh} es la matriz de pesos que multiplicamos por el estado oculto anterior h_{t-1}.
- w_{xh} es la matriz de pesos que multiplicamos por el input actual x_t.
- tanh es la función de activación tangente hiperbólica. Esto se hace para obtener una no linealidad en el sistema.

Y la salida de la red es:

$$y_t = w_{hy}h_t \tag{4}$$

Donde w_{hy} es la matriz de pesos que multiplicamos por el estado oculto h_t para obtener la salida y_t.

2.2 Flujo de RNN

Aquí se muestra una imagen con cajas que describe el flujo de los datos en una RNN con cajas. En t = 0, se aplica la función f_ω a la entrada x_0 y al estado oculto h_0, que es el estado oculto inicial. Esto genera un nuevo estado oculto h_1. Luego, este nuevo estado oculto se utiliza junto con la entrada x_1 en la función f_ω para generar el siguiente estado oculto h_2. Este proceso se repite para cada entrada en la secuencia. En este caso, usamos siempre la matriz de pesos w para todas las interacciones, lo que significa que la red tiene los mismos pesos para cada paso de tiempo.

Para aplicar el algoritmo de backpropagation, la información se propaga hacia atrás a través de la secuencia. Cada instante temporal tiene su gradiente asociado, así que el gradiente para la matriz de pesos w se calcula como la suma de los gradientes de cada instante temporal.

Para obtener las salidas para cada instante temporal (y_t), se multiplica el estado oculto correspondiente h_t por la matriz de pesos w_{hy}.

La pérdida total del modelo se calcula como la suma de las pérdidas de cada instante temporal. La función de pérdida se puede calcular como la suma de las pérdidas individuales para cada instante temporal, y luego se propaga hacia atrás a través de la secuencia para actualizar los pesos.

Esto se hace para los N a N, si es un N a 1, se hace lo mismo pero solo se calcula la salida para el último instante temporal. En el caso de 1 a N, solo se aplica el input al instante temporal 0 y se calcula la salida para todos los instantes temporales.

Comenta también que un encoder sería un modelo N a 1 y un decoder sería un modelo 1 a N. Cada modelo tiene su matriz w.

2.3 Ejemplo RNN

Estas redes se suelen utilizar para problemas de modelado de lenguaje. Muestra un ejemplo en el que se utiliza una RNN N a N para predecir la siguiente letra de una palabra (hola) y muestra que al introducir la letra "h", la red calcula un estado oculto h_1 y deduce la letra "o". Luego, al introducir la letra "o" como input, y usando el vector de estado oculto h_1, la red calcula el siguiente estado oculto h_2 y deduce la letra "l"...

2.4 Aplicación sobre imágenes

Para su aplicación sobre imágenes, se utiliza una red convolucional sobre la imagen para extraer las características de la imagen y luego se utiliza una RNN para procesar estas características y devolver el texto de salida.

En este caso, la ecuación para calcular el estado oculto sería:

$$h_t = \tanh(w_{hh}h_{t-1} + w_{xh}x_t + w_{ih}i_t) \tag{5}$$

Donde w_{ih} es la matriz de pesos que multiplicamos por el vector de características extraídas de la imagen i_t.

3 Introducción a LSTM

3.1 LSTM

Las LSTMs (Long Short-Term Memory) son un tipo de RNN que fueron introducidas para resolver uno de los principales problemas de las RNN estándar, el desvanecimiento del gradiente. El desvanecimiento del gradiente es un problema que ocurre cuando las redes neuronales tienen problemas para aprender y recordar información a lo largo de secuencias largas. Esto se debe a la disminución progresiva de los gradientes que se calculan durante el entrenamiento, por lo que tienden a olvidar información a largo plazo.

Estas redes resuelven este problema introduciendo un mecanismo de celdas de memoria, que puede retener información a lo largo de la secuencia. Esta capacidad para mantener información durante muchas iteraciones hace que las LSTM sean ideales para tareas que requieran procesamiento de datos temporales, NLP, etc.

Las LSTM son una sofisticación de las RNN tradicionales, que introducen el concepto de la celda de memoria. Esta celda de memoria es una estructura que se encarga de mantener la información relevante durante muchos pasos, evitando que se pierda o degrade la información conforme avanza la secuencia. Esta célula de memoria está controlada por una serie de mecanismos, llamados **puertas** que regulan la información que entra y sale de la celda de memoria y permiten que la red decida de una forma más eficiente qué información mantener y qué información descartar.

3.2 Componentes clave de las LSTM

Los componentes clave de las LSTM que las distinguen de las RNN estándar son:

- **Célula de memoria**: Es el componente central de las LSTM y, a diferencia de las RNN tradicionales, que no tienen memoria, las LSTM están diseñadas para recordar información durante largos pasos de tiempo. La celda de memoria actúa como un contenedor con información que puede actualizarse, olvidarse o utilizarse según sea necesario. El estado de la célula de memoria es lo que permite a las LSTM conservar el contexto relevante durante períodos más largos. El valor de la celda puede cambiar según la red crea conveniente, lo que permite que la LSTM decida qué información recordar y qué información olvidar.

- **Puertas en LSTM**: son mecanismos que controlan de manera precisa cómo fluye la información a través de la célula de memoria.

  - **Puerta de olvido**: Es un componente muy importante en la arquitectura de las LSTM ya que decide cuánta información previa debe descartarse. Es importante para evitar que la célula de memoria almacene información irrelevante. Matemáticamente, la puerta de olvido se define mediante una función sigmoide f , el estado oculto anterior h_{t-1} y el input actual x_t. Produce un valor entre 0 y 1 que determina qué fracción de la memoria anterior se mantendrá. Si es cercano a 0, la célula de memoria se olvida de gran parte de la información anterior, mientras que si es cercano a 1, se mantiene casi toda la información para utilizarla en el futuro.

    $$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{6}$$

    Nota: f se debe a forget

  - **Puerta de entrada**: Es responsable de decidir qué nueva información debe almacenarse en la celda de memoria. Actúa en conjunto con otro valor denominado *nuevo candidato de memoria* o $\tilde{c}_t$, que es calculada mediante una función tangente hiperbólica para generar una posible actualización de la memoria. Usa $\tilde{c}_t$ para determinar qué información se va a añadir a la celda de memoria. Matemáticamente, la puerta de entrada se define mediante una función sigmoide σ, el estado oculto anterior h_{t-1} y el input actual x_t. Produce un valor entre 0 y 1, al igual que la puerta de olvido, que controla la magnitud con la que la nueva información actualiza la memoria. La combinación de la puerta de entrada y con el candidato de memoria permite a las LSTM incorporar información nueva ajustándose dinámicamente a las condiciones cambiantes de la secuencia.

    $$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{7}$$

    Nota: i se debe a input

    $$\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c) \tag{8}$$

– **Puerta de salida**: Es la encargada de determinar qué parte de la información contenida dentro de la célula debe usarse para generar la salida de la LSTM en el instante temporal t. Esto es importante ya que no toda la información almacenada en la celda de memoria es relevante para la salida. Actúa como un filtro que decide qué información debe pasarse al estado oculto h_t y, por lo tanto, a la siguiente iteración.

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{9}$$

Nota: o se debe a output

$$h_t = o_t \cdot \tanh(c_t) \tag{10}$$

Así se obtiene el estado oculto h_t, a partir de la celda de memoria c_t y la puerta de salida o_t.

3.3 LSTM

Imagen de flujo de información en una LSTM:

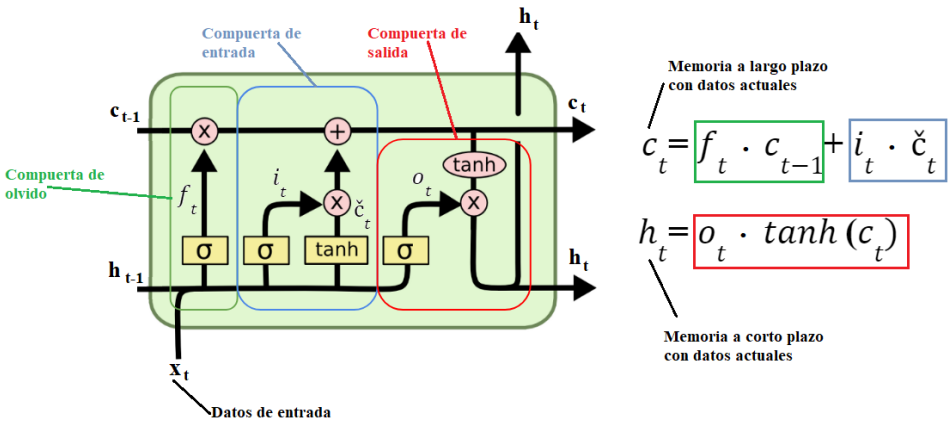

Figure 1: Flujo de información en una LSTM

3.4 Flujo de información en las LSTM

A lo largo de cada paso de tiempo de una secuencia, las LSTM pasan por una serie de actualizaciones durante las cuales las puertas deciden qué información se mantiene, se olvida o se utiliza de una forma más eficiente que las RNNs clásicas.

Esta arquitectura se puede usar tanto en la traducción de una oración, ya que puede utilizar el contexto para traducir palabras que puedan tener diferentes significados según el contexto, como en la predicción de series temporales, ya que puede recordar información relevante a lo largo de la secuencia. También son útiles para modelar dependencias a largo plazo.

4 Comparación de LSTM con RNN estándar y GRU

4.1 Comparación entre RNN, LSTM y GRU

Para solucionar el problema de las RNN estándar de recordar información a largo plazo, se han desarrollado dos arquitecturas de RNN: las LSTM y las GRU (Gated Recurrent Unit).

En una RNN tradicional, cada neurona toma como entrada tanto la entrada actual de la secuencia como el estado oculto de la iteración anterior.

4.2 Ventajas LSTM

Fueron introducidas por Hochreiter y Schmidhuber en 1997. Son una evolución de las RNNs, diseñadas para resolver el problema del desvanecimiento del gradiente. Ventajas:

- **Capacidad para manejar dependencias a largo plazo**: Gracias a la célula de memoria y las puertas, las LSTM pueden recordar información durante períodos prolongados, lo que las hace ideales para tareas que requieren un contexto a largo plazo.
- **Flexibilidad**: Deciden de forma dinámica qué información mantener y qué información olvidar, lo que les permite adaptarse a diferentes tipos de datos y tareas.
- **Mitigación del desvanecimiento del gradiente**: Evitan que, durante el Backpropagation, los gradientes disminuyan progresivamente desvanciéndose.

4.3 GRU

Las Gated Recurrent Units (GRU) son un tipo de RNN que fueron introducidas en 2014 y surgen como una simplificación de las LSTM. Están diseñadas para solucionar los problemas de las RNNs con el desvanecimiento del gradiente y la dificultad para recordar información a largo plazo, pero con una arquitectura más simple que las LSTM. En vez de tener 3 puertas (olvido, entrada y salida), las GRU tienen solo 2 puertas (puerta de actualización y puerta de reinicio). La puerta de actualización combina las funciones de la puerta de olvido y la puerta de entrada de las LSTM, ya que decide cuánta información de la memoria anterior debe retenerse y cuánta nueva información debe añadirse. La puerta de reinicio controla cuánto de la memoria anterior debe tenerse en cuenta para calcular el nuevo estado oculto o salida; tiene una función similar a la puerta de salida de las LSTM.

4.4 Comparación entre LSTM y GRU

- **Complejidad**: Es la principal diferencia entre ambas arquitecturas. Las LSTM tienen 3 puertas y una célula de memoria, mientras que las GRU tienen 2 puertas y no tienen una célula de memoria separada. Esto hace que las GRU sean más simples y, por lo general, más rápidas de entrenar que las LSTM.
- **Rendimiento en secuencias largas**: Ambas arquitecturas son buenas para manejar dependencias a largo plazo, pero algunos estudios sugieren que las LSTM pueden ser ligeramente más efectivas en tareas en las que las secuencias de datos son muy largas o complejas, ya que al tener la célula de memoria permite un mayor control sobre la información que se retiene y se olvida.
- **Coste computacional**: Las GRU suelen ser más rápidas y computacionalmente eficientes que las LSTM, lo que las convierte en una opción preferida en tareas en las que el tiempo de entrenamiento y la eficiencia de cálculo son críticos.
- **Aplicaciones y rendimiento**: En términos generales, las LSTM y las GRU tienden a ofrecer rendimientos similares. En algunos casos, las GRU han demostrado funcionar mejor cuando se dispone de menos datos o cuando las dependencias temporales no son extremadamente complejas.
- **Capacidad de modelado**: Las LSTM, gracias a su arquitectura más compleja, pueden ser más adecuadas para problemas donde se necesita modelar dependencias muy largas y donde la red debe ser extremadamente precisa en la memoria. Por otro lado, las GRU pueden ofrecer resultados similares en problemas con menos requerimientos de memoria a largo plazo.

4.5 Resumen comparativo

Característica	RNN estándar	LSTM	GRU
Memoria a largo plazo	Limitada	Excelente	Muy buena
Problema del gradiente	Desvanecimiento	Mitigado	Mitigado
Complejidad	Baja	Alta	Media
Velocidad de entrenamiento	Rápida (en secuencias cortas)	Lenta (debido a más puertas)	Más rápida que LSTM
Número de puertas	N/A	3 (olvido, entrada, salida)	2 (actualización, reinicio)
Eficiencia computacional	Alta en secuencias cortas	Menos eficiente	Más eficiente que LSTM
Aplicaciones	Secuencias cortas	Secuencias largas y complejas	Secuencias de longitud media/larga

Table 1: Comparación entre RNN estándar, LSTM y GRU

5 ¿Cuándo usar LSTM?

5.1 Casos de uso de las LSTM

- **Dependencias temporales a largo plazo**: uno de los puntos más fuertes de las LSTM es su capacidad de recordar información en largos periodos de tiempo.
  - Traducción automática: ya que para entender el significado de una palabra o frase es importante tener en cuenta el contexto.
  - Generación de textos: ya que para mantener el estilo, la consistencia y la coherencia del texto es importante tener en cuenta lo generado anteriormente.
  - Reconocimiento de voz: ya que es capaz de identificar patrones pronunciados anteriormente y relacionarlos con el contexto actual.
- **Problemas con alta complejidad temporal**: cuando los datos secuenciales tienen relaciones temporales complejas, las LSTM pueden ser muy útiles ya que, gracias a su célula de memoria y sus puertas, pueden seleccionar qué información es relevante y cuál no, ajustándose así a la complejidad de los datos.
  - Predicción de series temporales: como acciones o el precio de la energía que vienen influenciados por factores actuales pero también por el estado del mercado en el pasado.
  - Control robótico o de sistemas:
- **Datos ruidosos o incompletos**:
  - Sensores en tiempo real:
  - Medicina y diagnóstico:
- **Tareas que requieren precisión en la memoria y contexto**:
  - Modelado de lenguaje natural:
  - Análisis de videos:
- **Cuando las RNN estándar no son suficientes**:
  - Predicción de precios:
  - Reconocimiento de patrones en series temporales complejas:

6 Limitaciones y mejores prácticas de LSTM

6.1 Limitaciones de las LSTM

- **Complejidad y coste computacional**: las puertas y la célula de memoria añaden complejidad a la red, además de aumentar el coste computacional y el tiempo de entrenamiento. Así que en casos en los que haya recursos limitados o se necesite un entrenamiento rápido, las LSTM pueden no ser la mejor opción.
- **Problemas con secuencias extremadamente largas**: aunque son bastante mejores que las RNNs en escenarios donde las secuencias son extremadamente largas, pueden seguir teniendo problemas con el desvanecimiento del gradiente a muy largo plazo perdiendo información relevante.
- **Riesgo de sobreajuste**: este riesgo es considerable en conjuntos de datos muy pequeños, por lo que retiene patrones específicos sin generalizar bien a nuevos datos.

6.2 Mejores prácticas de LSTM

- **Normalización de datos**: Una de las primeras prácticas a implementar al entrenar una LSTM. Ayuda a mejorar la convergencia del modelo, haciéndolo más eficiente y evitando que el modelo quede atrapado en mínimos locales durante el proceso de optimización. En el caso de secuencias temporales, esto puede implicar la normalización de las *features* (normalmente de 0 a 1) y, al reducir la escala de las entradas, se facilita que las LSTM aprendan patrones en los datos y mantengan un mejor equilibrio en los gradientes propagados durante el entrenamiento. Sin una normalización adecuada, los gradientes pueden ser muy grandes o muy pequeños, lo que puede dificultar el aprendizaje de la red.
- **Uso de regulación para evitar sobreajuste**:
  - **Dropout**: Consiste en apagar aleatoriamente una fracción de las neuronas en cada paso del entrenamiento. Esto obliga al modelo a ser más robusto, porque no va a depender excesivamente de características muy específicas.
    - **Dropout en las conexiones recurrentes**: apaga neuronas en las conexiones que hay entre capas LSTM, evitando que el modelo memorice patrones temporales innecesarios.
    - **Dropout en las capas de salida**: Se puede aplicar tras las capas *Fully Connected* del final de la red. Aunque es más común aplicarlo en las capas intermedias.
  - **L2 regularization**: esta técnica penaliza los pesos muy altos del modelo haciendo que los parámetros se mantengan bajo control y ayuda a evitar el *overfitting*.
- **Experimentación con capas bidireccionales**: Las capas bidireccionales permiten al modelo considerar tanto el contexto anterior como posterior en una secuencia, lo cual es útil en tareas donde el contexto completo mejora la precisión, como en el procesamiento de lenguaje natural.
- **Uso de modelos apilados** *(stacked LSTM)*: para mejorar la capacidad de las LSTM, se pueden apilar múltiples capas LSTM. Esto permite que el modelo aprenda representaciones más complejas y capture mejor características de datos secuenciales. Cada capa LSTM añadida puede ayudar a procesar los datos de una forma más compleja o abstracta. Esta práctica aumenta la complejidad, por tanto, también incrementa los tiempos de entrenamiento y el riesgo de sobreajuste.
- **Ajuste de hiperparámetros**:
  - **Tamaño del número de unidades LSTM**: es decir, cuántas unidades va a tener la LSTM lo que controla la cantidad de memoria y la capacidad de aprendizaje de la red. Captura patrones más complejos. Aunque aumenta el riesgo de sobreajuste.
  - **Tasa de aprendizaje**: debe ser equilibrada, lo suficientemente pequeña para garantizar que el modelo aprenda de manera gradual y efectiva, pero no tanto como para hacer ineficiente el proceso de entrenamiento.
  - **Batch size**: si los lotes son pequeños, aumenta la capacidad de generalización, mientras que los lotes más grandes aceleran el entrenamiento. Se suele experimentar o hacer validación cruzada.