Inteligencia Artificial

CC-421

César Lara Avila

Universidad Nacional de Ingeniería

(actualización: 2021-02-10)

Bienvenidos

NLP y redes Neuronales

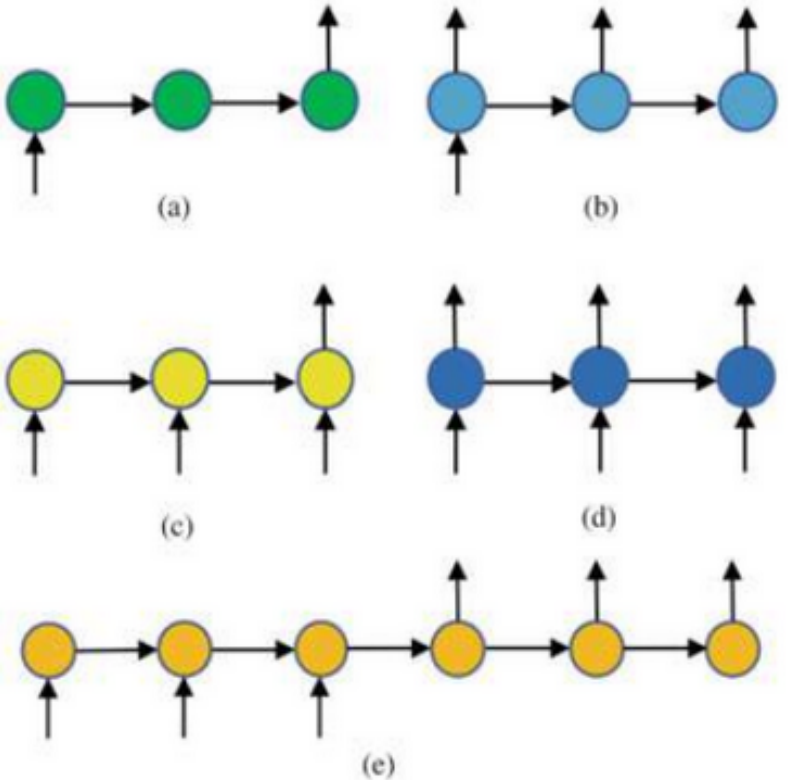
- Extensiones de la redes neuronales recurrentes
- Modelo secuencia a secuencia (seq2seq)
- Mecanismos de atención
- Transformer
- Aplicaciones de las RNN en NLP

Extensiones de la redes neuronales recurrentes

Las redes neuronales recurrentes se pueden utilizar para resolver muchos problemas basados en secuencia. Se pueden dar los siguientes casos en este contexto:

- **RNN de secuencias uno a uno**, es equivalente a una red neuronal con pesos compartidos.
- **RNN de secuencia de uno a muchos**. Genera una serie de salidas, dada una entrada. Un ejemplo de aplicación de esta arquitectura es el titulado de imágenes (image caption), donde la capa de entrada acepta una sola imagen y le asigna a varias palabras en el título.
- **RNN de secuencia de muchos a uno**. En esta arquitectura una serie de entradas produce una única salida. Un ejemplo de aplicación es la clasificación de sentimientos, tarea donde la capa de entrada acepta múltiples tokens de palabras de una oración y las asigna a un solo sentimiento de la oración como positivo o negativo.

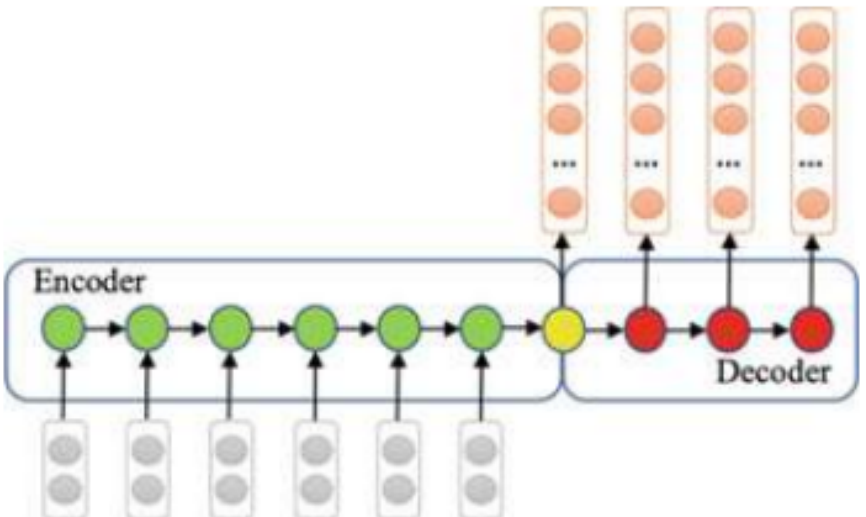
- **RNN de secuencia de muchos a muchos**. En esta arquitectura, varias unidades de entrada de RNN se asignan a múltiples unidades ocultas y de salida. En este escenario, tenemos una alineación de uno a uno entre el número de pasos de tiempo de entrada y salida. Esta estructura es común en el modelado de lenguaje.



Modelo secuencia a secuencia (seq2seq)

El modelo seq2seq intenta aprender una red neuronal que predice una secuencia de salida a partir de una secuencia de entrada.

Las secuencias son un poco diferentes de los vectores tradicionales, porque una secuencia implica un orden de eventos y el tiempo es una forma intuitiva de ordenar eventos.



Dichos modelos constan de dos componentes: un codificador que calcula una representación para una oración de entrada y un decodificador que genera una palabra objetivo a la vez.

Es decir en el proceso de codificación, la secuencia de entrada $X = (\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_T)$ se alimenta en el codificador, que es básicamente un modelo RNN.

Luego, a diferencia de un RNN simple donde se toma en cuenta la salida de cada unidad de estado, solo se mantiene la unidad oculta de la última capa $\mathbf{h}_t$.

Este vector se llama vector de contexto $\mathbf{c}$ y pretende contener una representación de la oración de entrada:

$$\begin{aligned} \mathbf{h}_t &= LSTM(\mathbf{h}_{t-1}, \mathbf{x}_t) \\ \mathbf{c} &= \tanh(\mathbf{h}_T) \end{aligned}$$

donde $\mathbf{h}_t$ es un estado oculto en el tiempo t, y $\mathbf{c}$ es el vector de contexto de las capas ocultas del codificador.

Por otro lado, al pasar el vector de contexto $\mathbf{c}$ y todas las palabras predichas previamente $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}$ al decodificador, el proceso de decodificación predice la siguiente palabra $\mathbf{y}_t$.

El decodificador define una probabilidad sobre la salida $\mathbf{y}$ descomponiendo la probabilidad conjunta de la siguiente manera:

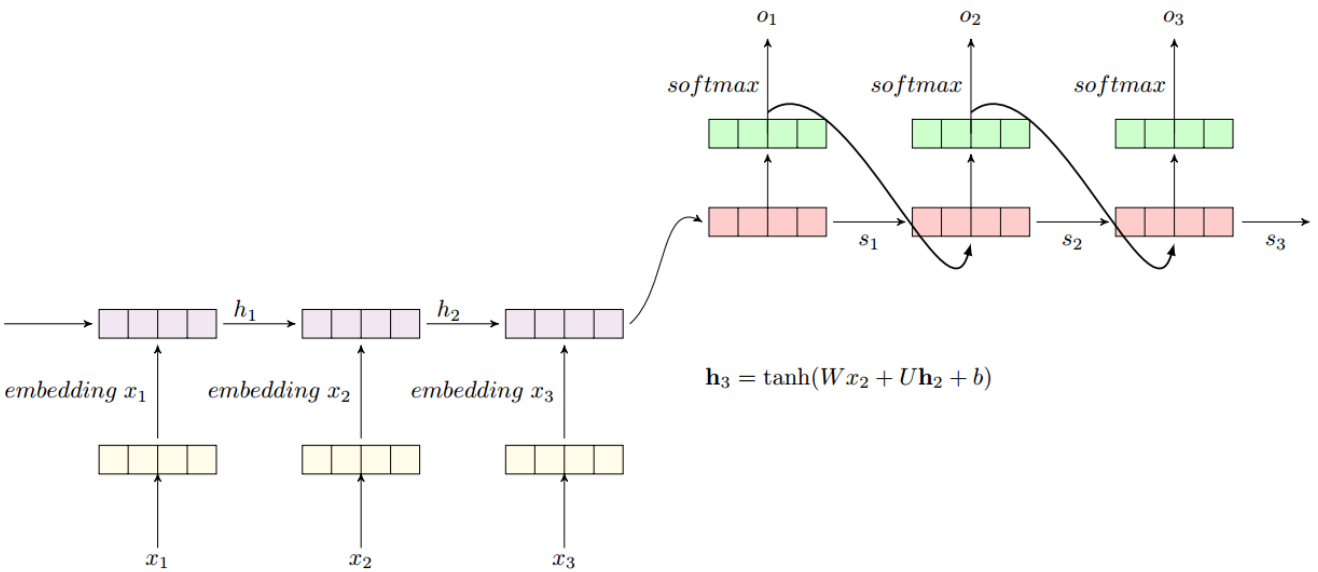
$$p(\mathbf{y}) = \prod_{t=1}^{T} p(\mathbf{y}_t | \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}\}, \mathbf{c})$$

$$p(\mathbf{y}_t | \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1}\}, c) = \tanh(\mathbf{y}_{t-1}, \mathbf{c})$$

donde $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T)$ y $\mathbf{h}_t$ es la unidad oculta.

La siguiente figura muestra el proceso de codificación y decodificación para una secuencia de entrada $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$.

La función Softmax se utiliza para normalizar la distribución de probabilidad para la salida y para calcular la pérdida de error.



La mayoría de los modelos de secuencia a secuencia para la corrección gramatical de errores (GEC) tienen dos limitaciones:

- Los modelos seq2seq están entrenados con solo pares de oraciones con corrección de errores limitada y limitados por el tamaño de los datos de entrenamiento eso implica que los modelos con millones de parámetros pueden no estar bien generalizados

- Un modelo seq2seq puede fallar al corregir completamente una oración con múltiples errores a través de la inferencia normal seq2seq, porque algunos errores en una oración pueden hacer que el contexto sea extraño, lo que confunde a los modelos para corregir otros errores.

Para resolver estos inconvenientes se plantea un nuevo mecanismo de inferencia y un aprendizaje llamado **fluency boost** utilizando el concepto de fluidez.

Se define la fluidez como la probabilidad de que la oración sea escrita por un hablante nativo.

Este modelo no solo se entrena un modelo secuencia a secuencia con pares de oraciones corregidas con errores originales, sino que también genera oraciones menos fluidas para establecer nuevos pares de oraciones con corrección de errores al emparejarlas con sus oraciones correctas durante el entrenamiento, siempre que la fluidez de las oraciones sea inferior a la de sus oraciones correctas.

Una dificultad más al enfoque secuencia a secuencia es que el estado oculto tiende a reflejar la información más reciente, perdiendo memoria del contenido anterior.

Esto fuerza una limitación para **secuencias largas**, donde toda la información sobre esa secuencia debe resumirse en una sola codificación, lo que conlleva a utilizar un **mecanismo de atención**.

Mecanismos de atención

La atención ha sido una de las técnicas más populares para abordar el problema de dependencias largas y el uso eficiente de la memoria para el cálculo.

El mecanismo de atención interviene como una capa intermedia entre el codificador y el decodificador, con el objetivo de capturar la información de la secuencia de palabras que son relevantes para el contenido de la oración.

Si s_i es el estado oculto de atención aumentada en el tiempo i, se necesitan tres entradas:

- El estado oculto anterior del decodificador $\mathbf{s}_{i-1}$
- La predicción del paso de tiempo anterior $\mathbf{y}_{i-1}$
- Un vector de contexto $\mathbf{c}_i$ que pondera los estados ocultos apropiados para el paso de tiempo dado.

$$s_i = f(\mathbf{s}_{i-1}, \mathbf{s}_i, \mathbf{c}_i)$$

El vector de contexto c_i es definido como:

$$\sum_{j=1}^{T_x} \alpha_{ij} \mathbf{h}_j$$

donde los **pesos de atención**, tienen la forma:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$
$$e_{ij} = a(\mathbf{s}_{i-1}, \mathbf{h}_j)$$

donde $a(\mathbf{s}, \mathbf{h})$ es un vector de alineación que califica qué tan bien coinciden las entradas en la posición j y la salida en la posición i.

Un inconveniente de usar todos los pasos de tiempo anteriores es que requiere una gran cantidad de cálculo para secuencias largas.

Un beneficio adicional de la atención es que proporciona una puntuación para cada paso de tiempo, identificando qué entradas fueron más útiles para la predicción.

Transformers

El éxito de la atención en las tareas de secuencia a secuencia genera la pregunta de si se puede aplicar directamente a la entrada, reduciendo o incluso eliminando la necesidad de conexiones recurrentes en la red.

El transformer aplica esta atención directamente a la entrada superando los modelos recurrentes y convolucionales en la traducción automática. En lugar de depender de los RNN para acumular una memoria de estados anteriores como en los modelos de secuencia a secuencia, el transformador utiliza la atención de **múltiples cabeceras** directamente en los embeddings de entrada.

Esto alivia las dependencias secuenciales de la red permitiendo que gran parte del cálculo se realice en paralelo.

La atención se aplica directamente a la secuencia de entrada, así como a la secuencia de salida según se predice.

Las partes de codificador y decodificador se combinan, utilizando otro mecanismo de atención antes de predecir una distribución de probabilidad sobre el diccionario de salida.

La atención de múltiples cabeceras, se define mediante tres matrices de entrada: $\mathbf{Q}$ el conjunto de consultas, las claves $\mathbf{K}$ y valores $\mathbf{V}$.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

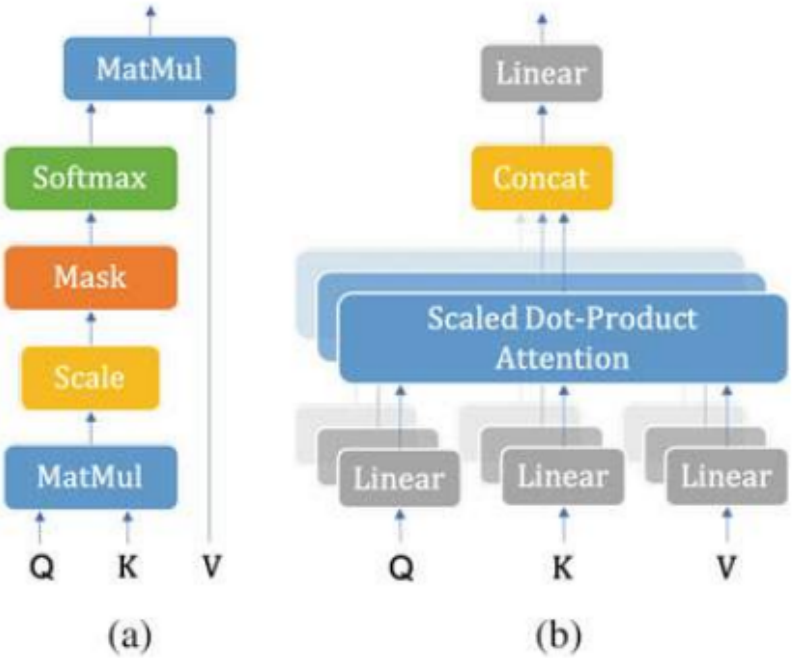
La atención de múltiples cabeceras se define entonces como:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots \text{head}_h)\mathbf{W}^O$$

donde:

$$\text{head}_i(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{V}_i^Q)$$

Los parámetros de todas las matrices $\mathbf{W}$ son matrices de proyección.



El codificador y el decodificador aplican múltiples capas de atención de múltiples cabezales con conexiones residuales y capas adicionales completamente conectadas.

Aplicaciones de las RNN en NLP

- Clasificación de texto
- Categorización gramatical (POS) y reconocimiento de entidad nombrada (NER)
- Análisis de dependencia
- Modelado y resumen de temas
- Respuesta a preguntas

Fin del curso!

Estas notas están basados en el texto de Deep Learning for NLP and Speech Recognition de Kamath, Liu y Whitaker.