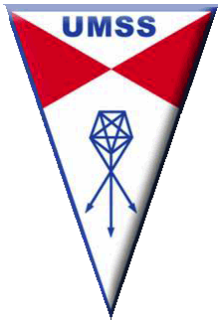


UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
LICENCIATURA EN ING. INFORMÁTICA

REDES NEURONALES RECURRENTES

Estudiantes:

Brun De La Fuente Denis

Pantoja Bustamante Gabriel

Salazar Choque Jorge

Materia: Inteligencia Artificial II

Fecha: 16/05/2023

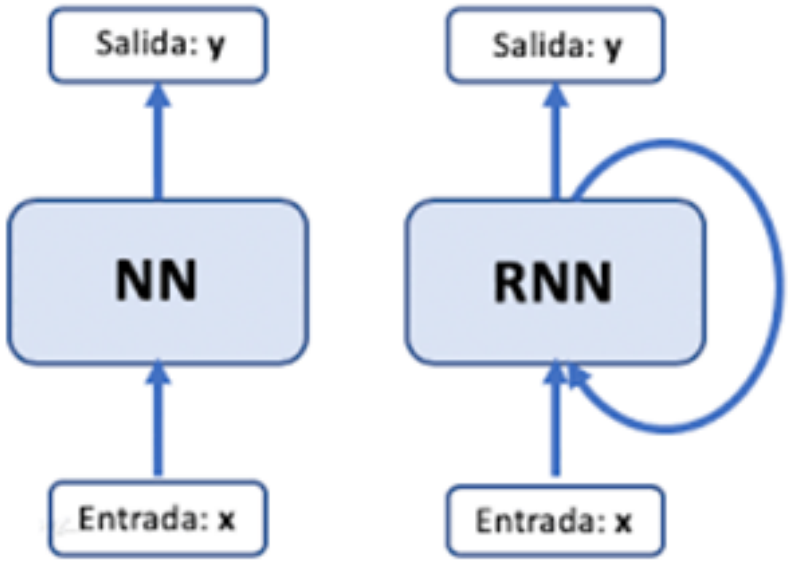
Cochabamba-Bolivia

ÍNDICE

1. ¿Qué son las Redes Neuronales Recurrentes(RNN)?

Las redes neuronales recurrentes (RNN) son un tipo de arquitectura de redes neuronales artificiales que se utiliza para procesar datos secuenciales o de series temporales, donde la salida en cada paso dependerá de la entrada actual y la memoria de la red neuronal.
La característica principal de las RNN es que tienen conexiones recurrentes que permiten que la información se retroalimenta a través del tiempo. En otras palabras, la salida de la red en un paso de tiempo se convierte en una entrada para el siguiente paso de tiempo, lo que permite que la red tenga memoria de pasos anteriores y pueda aprender patrones de secuencias.

1.1 La Neurona Recurrente



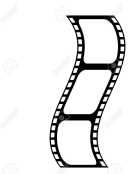
En una neurona convencional, la entrada se procesa y se produce una salida en función de una función de activación. La información fluye solo en una dirección, desde la entrada hasta la salida, sin conexión de retroalimentación.

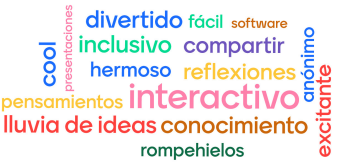
En cambio, en una neurona recurrente, la entrada se procesa de manera similar a una neurona convencional, pero también se almacena en la memoria interna de la neurona, que se utiliza para producir una salida en el siguiente paso de tiempo. Esta memoria interna se actualiza en cada paso de tiempo en función de la entrada actual y la memoria anterior. Debido a esta conexión de retroalimentación, la neurona recurrente puede procesar información secuencial o de series temporales**.**

1.2 ¿Cuándo utilizar las Redes Neuronales Recurrentes?

Cuando se tiene que procesar una serie datos conocido como secuencias o de series temporales como, por ejemplo:

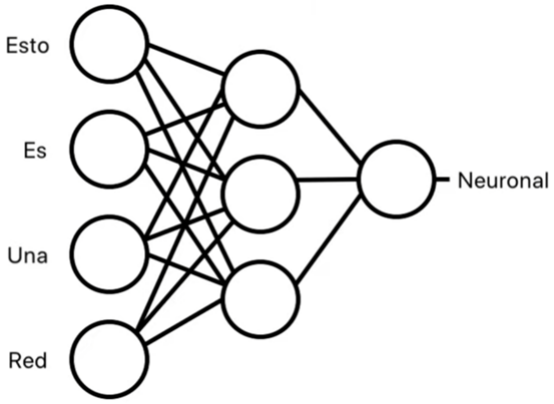
- Videos
- Notas musicales
- Palabras
- El clima





Esta secuencia tiene un orden específico y tienen únicamente significado cuando se analizan en conjunto y no de manera individual.

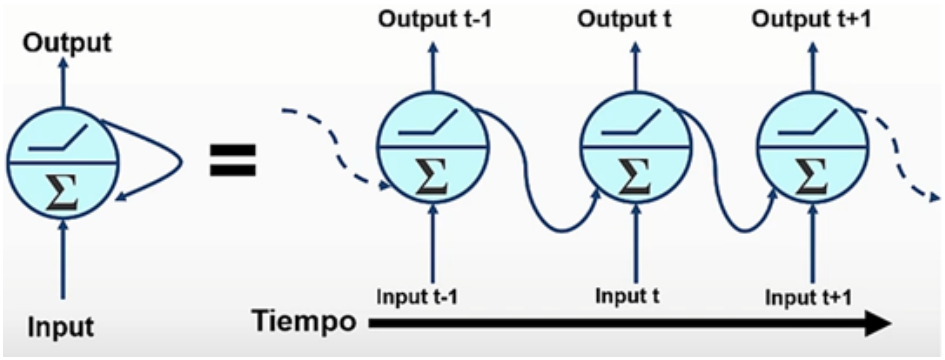
Por ejemplo, si queremos hacer un programa que reciba una oración incompleta y el programa debe predecir la palabra siguiente para completar la oración. Como por ejemplo la oración: **“ESTO ES UNA RED”** y lo que debe predecir es la palabra **“NEURONAL”**.



Si quisiéramos resolver el problema con una red neuronal convencional, el número de entradas que tendremos será el número de palabras que tiene la oración y la salida la palabra que debe predecir, tendríamos dos problemas, el primer problema sería el número fijo de entradas y de salidas por que la red neuronal solo recibe 5 entradas y otra oración puede tener diferente tamaño, entonces las secuencias no tienen un tamaño fijo. No sabemos con antelación el número de palabras que dirá una persona.

El segundo problema es que no se está compartiendo los parámetros en la red neuronal. Las redes neuronales recurrentes resuelven estos problemas por que pueden procesar tanto la entrada como la salida secuencias sin importar el tamaño, además teniendo en cuenta la correlación existente entre los diferentes elementos de la secuencia.

2. ¿Cómo funcionan las Redes Neuronales Recurrentes?

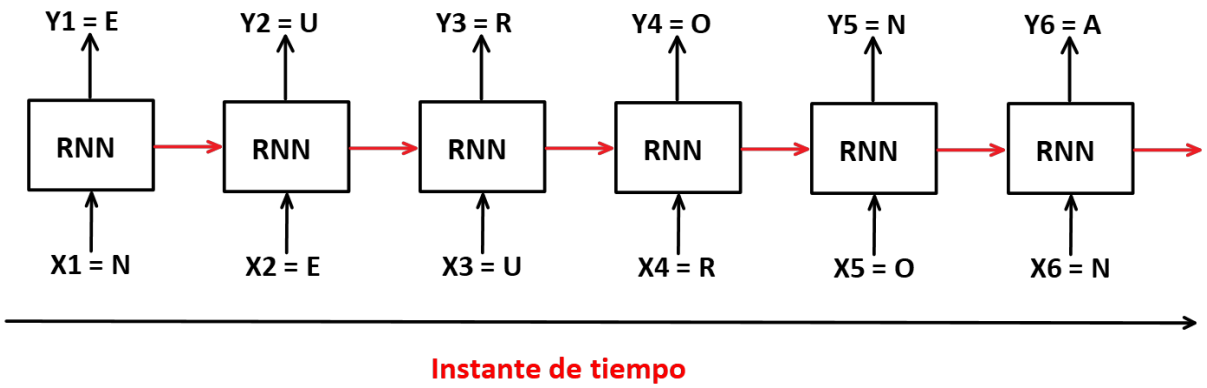


Lo que hace una red neuronal recurrente es que la salida es la realimentación de tal manera que también se convierte en un input en el siguiente estado o momento temporal. Por tanto, si representamos el tiempo en una línea, tenemos en el tiempo **t – 1** la salida de la neurona que va alimentar a la misma neurona en el tiempo **t**, igualmente volverá a alimentar la misma neurona en tiempo **t + 1** y así sucesivamente. Entonces una secuencia tiene un tiempo asociado por ejemplo si fuera la palabra:

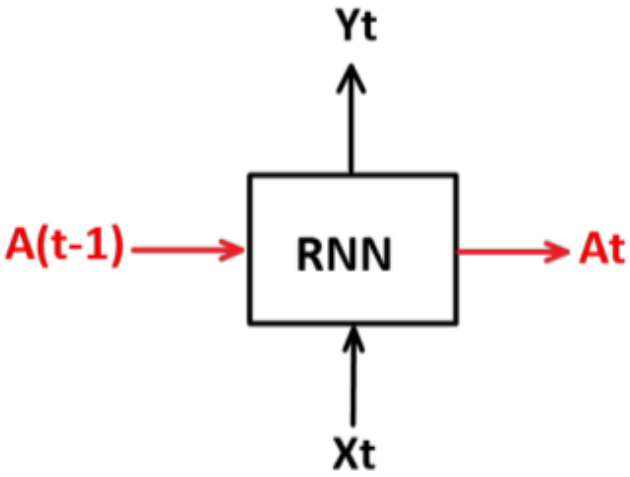
La RNN tendrá los siguientes estados donde:

Xt: entrada a RNN en el instante de tiempo t

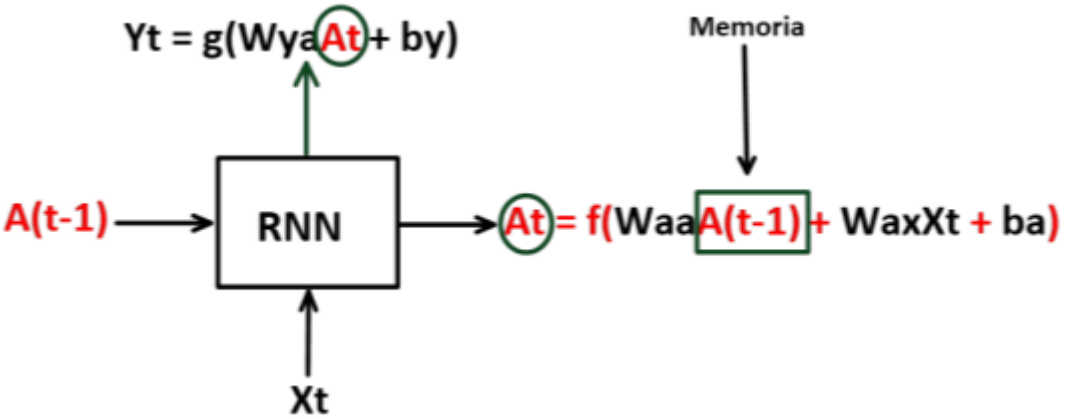
Yt: salida de la RNN en el instante de tiempo t



2.1 ¿Cómo puede predecir el siguiente carácter en la secuencia? ¿Dónde está la memoria de la Neurona?



En la figura en cada instante de tiempo la neurona tiene dos entradas y dos salidas, las entradas son el dato actual **Xt** y la activación anterior **A(t – 1)** mientras que las salidas son la predicción actual **Yt** y la activación actual **At**, esta activación de salida recibe el nombre de estado oculto, esta es la activación que corresponde precisamente a la memoria de la red, pues permite preservar y compartir la información entre un instante de tiempo y otro.



Si observamos las funciones, la salida **Yt** depende de la activación actual **At** pero a su vez esta activación depende no solo de la entrada actual **Xt** si no también del valor previo de la activación **A(t – 1)**, esta es precisamente la memoria de la red y la forma que permite preservar y compartir la información en cada instante de tiempo. Los coeficientes se obtienen con el entrenamiento de la red y estos serán los mismos durante cada instante de tiempo.

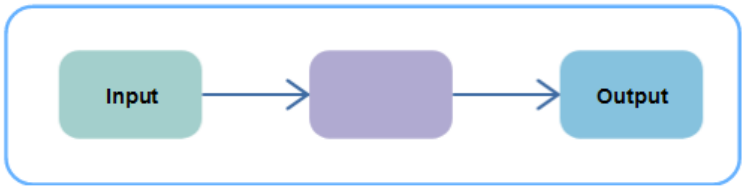
3. Tipos de redes neuronales recurrentes

Hay varios tipos de redes neuronales recurrentes (RNN, por sus siglas en inglés). Los cuatro tipos de Redes Neuronales Recurrentes comúnmente utilizados son:

- One to one
- One to many
- Many to one
- Many to many

3.1 RNN one-to-one.

El tipo más simple de RNN es one-to-one, que permite una sola entrada y una sola salida. Tiene tamaños de entrada y salida fijos y actúa como una red neuronal tradicional. La aplicación Uno-a-Uno se puede encontrar en la clasificación de imágenes.

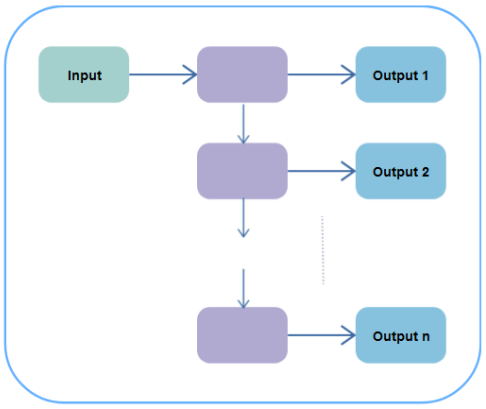


One-to-One

3.2 RNN one-to-many

Una red neuronal recurrente one-to-many es un tipo de red neuronal que toma una sola entrada y produce una secuencia de salidas.

Un ejemplo común de una red neuronal recurrente "one to many" es el proceso de generación de subtítulos para videos o imágenes. En este caso, la red neuronal toma una imagen o un fotograma de video como entrada, y genera una secuencia de palabras o frases que describen el contenido de la imagen o el video.

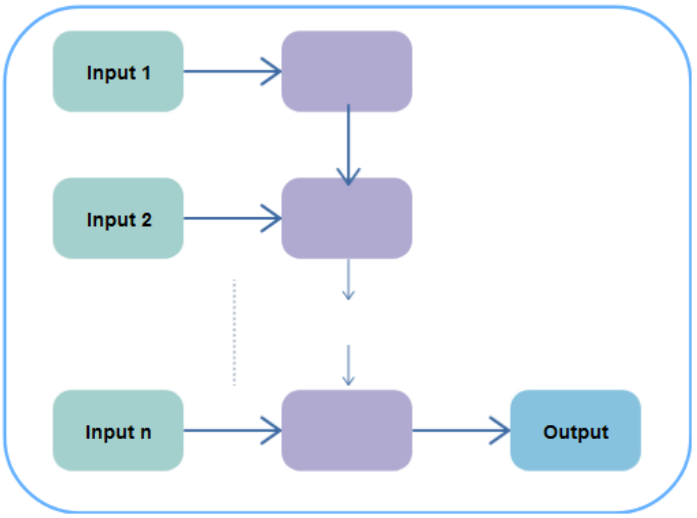


One-to-Many

Este tipo de red neuronal también se puede utilizar en otras aplicaciones, como la generación de música, la descripción de audio o la transcripción de voz a texto. En general, cualquier tarea en la que se necesite generar una secuencia de salidas a partir de una sola entrada puede ser abordada utilizando una red neuronal recurrente one-to-many.

3.3 RNN many-to-one

Una red neuronal recurrente many-to-one es un tipo de red neuronal que toma una secuencia de entradas y produce una única salida.



Many-to-One

Un ejemplo de este tipo de red es el análisis de sentimiento en el procesamiento del lenguaje natural. En este caso, la red neuronal toma una secuencia de palabras como entrada y produce una salida que indica si el texto tiene un sentimiento positivo o negativo.

Otros posibles usos para este tipo de red neuronal pueden ser: la clasificación de documentos, la detección de fraude o la predicción de precios de acciones. En general, cualquier tarea en la que se necesite producir una única salida a partir de una secuencia de entradas puede ser abordada utilizando una red neuronal recurrente many-to-one.

3.4 RNN many to many

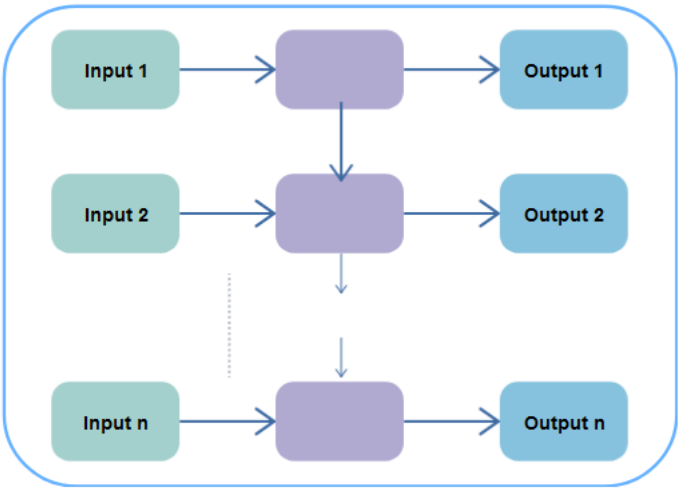
Many-to-many se utiliza para generar una secuencia de datos de salida a partir de una secuencia de unidades de entrada.

En general, este tipo de red se utiliza en tareas que involucran secuencias de datos, como el procesamiento del lenguaje natural, la generación de texto, la detección de objetos en video, entre otros.

Este tipo de RNN se divide además en las siguientes dos subcategorías:

3.4.1 Equal unit size

En este caso, el número de unidades tanto de entrada como de salida es el mismo.

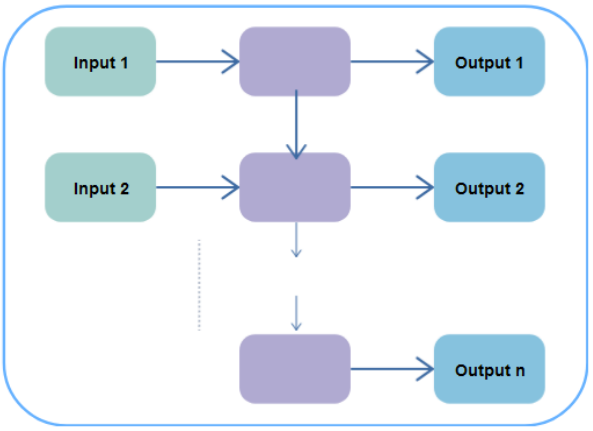


Many-to-Many (Equal)

Este tipo de red neuronal se utiliza, por ejemplo, en la traducción de idiomas, donde la red neuronal toma una secuencia de palabras en un idioma como entrada y produce una secuencia de palabras en otro idioma como salida.

3.4.2 Unequal Unit Size

En este caso, las entradas y salidas tienen un número diferente de unidades.



Many-to-Many (Unequal)

Se puede usar este tipo de red neuronal por ejemplo en el etiquetado de partes del habla, donde la red neuronal toma una secuencia de palabras como entrada y produce una secuencia de etiquetas que indican la parte del habla de cada palabra. En este caso, la etiqueta de cada palabra se relaciona con la palabra en sí misma, pero en un instante de tiempo anterior o posterior.

4. RNN y la retropropagación (backpropagation) a través del tiempo.

Para entender el concepto de retropropagación a través del tiempo o BPTT por sus siglas en inglés, primero necesitamos entender los conceptos de propagación hacia adelante y hacia atrás. Se podría dedicar un artículo completo a discutir estos conceptos, por lo que se proporciona una definición lo más simple posible.

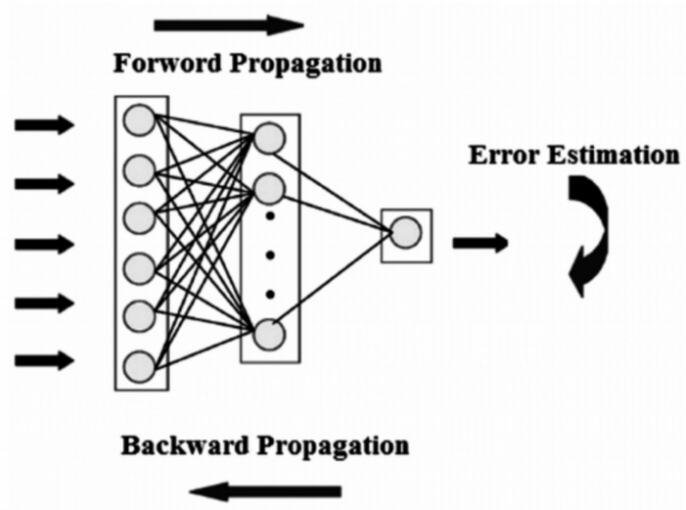
Backpropagation.- La retropropagación es conocida como un algoritmo fundamental en el aprendizaje automático. La retropropagación se utiliza para calcular el gradiente de una función de error con respecto a los pesos de una red neuronal. El algoritmo trabaja en reversa a través de las distintas capas de gradientes para encontrar la derivada parcial de los errores con respecto a los pesos. Luego, la retropropagación utiliza estos pesos para disminuir los márgenes de error durante el entrenamiento.

En las redes neuronales, básicamente haces una propagación hacia adelante para obtener la salida de tu modelo y verificar si esta salida es correcta o incorrecta, para obtener el error. La retropropagación no es más que retroceder a través de tu red neuronal para encontrar las derivadas parciales del error con respecto a los pesos, lo que te permite restar este valor de los pesos.

Esas derivadas son luego utilizadas por el descenso de gradiente, un algoritmo que puede minimizar iterativamente una función dada. Luego, ajusta los pesos hacia arriba o hacia abajo, dependiendo de cuál disminuya el error. Es exactamente así como una red neuronal aprende durante el proceso de entrenamiento.

Entonces, con la retropropagación, básicamente intentas ajustar los pesos de tu modelo mientras lo entrenas.

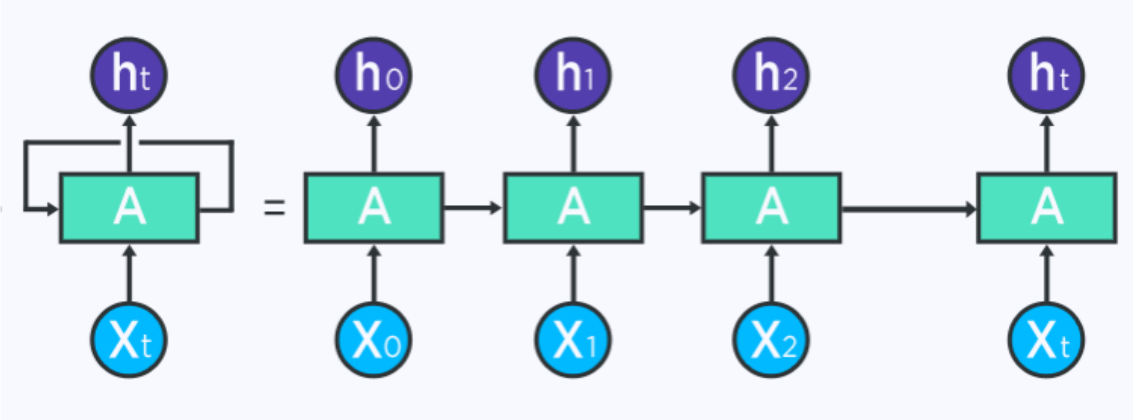
La siguiente imagen ilustra el concepto de propagación hacia adelante y retropropagación en una red neuronal de alimentación directa (feed-forward):



BPTT no es más que una palabra de moda elegante para hacer retropropagación en una red neuronal recurrente desenrollada. El desenrollado es una herramienta de visualización y conceptual que ayuda a comprender lo que está sucediendo dentro de la red. La mayoría de las veces, al implementar una red neuronal recurrente en los frameworks comunes, la retropropagación se maneja automáticamente, pero es necesario entender cómo funciona para solucionar problemas que puedan surgir durante el proceso de desarrollo.

Podemos ver una RNN como una secuencia de redes neuronales que se entrena una tras otra con retropropagación.

La imagen a continuación ilustra una RNN desenrollada. A la izquierda, la RNN se desenrolla después del signo igual. Ten en cuenta que no hay un ciclo después del signo igual, ya que los diferentes pasos de tiempo se visualizan y la información se pasa de un paso de tiempo al siguiente. Esta ilustración también muestra por qué se puede ver una RNN como una secuencia de redes neuronales.



Si se realiza BPTT, la conceptualización del desenrollado es necesaria ya que el error de un paso de tiempo dado depende del paso de tiempo anterior.

Dentro de BPTT, el error se retropropaga desde el último hasta el primer paso de tiempo, mientras se desenrollan todos los pasos de tiempo. Esto permite calcular el error para cada paso de tiempo, lo que permite actualizar los pesos.

Se debe tener en cuenta que BPTT puede ser computacionalmente costoso cuando se tiene un alto número de pasos de tiempo.

5. Problema de las Redes Neuronales Recurrentes

Los modelos de redes neuronales son entrenados por el algoritmo de optimización de descenso de gradiente. Los datos de entrenamiento de entrada ayudan a estos modelos a aprender, y la función de pérdida mide la precisión del rendimiento de la predicción para cada iteración cuando se actualizan los parámetros. A medida que avanza el entrenamiento, el objetivo es reducir la función de pérdida/error de predicción ajustando los parámetros de

forma iterativa. Específicamente, el algoritmo de descenso de gradiente tiene un paso hacia adelante y un paso hacia atrás, lo que le permite hacer esto.

Aunque las Redes Neuronales Recurrentes han demostrado ser efectivas en una amplia gama de aplicaciones, también se enfrentan a varios problemas inherentes que pueden limitar su rendimiento y capacidad para modelar con precisión las secuencias. Estos problemas pueden afectar tanto el proceso de entrenamiento como la capacidad de la red para generalizar a nuevos datos.

5.1 Desvanecimiento del Gradiente

Este problema se presenta cuando estamos entrenando la red neuronal con métodos de aprendizaje basados en gradientes.

Como vimos anteriormente las redes neuronales recurrentes usan gradientes para calcular la función de pérdida del modelo, y esto lo hacen usando el algoritmo de backpropagation.

Estos métodos durante cada iteración del entrenamiento, cada uno de los pesos de la red neuronal recibe una actualización proporcionada por las derivadas parciales de la función de error con respecto al peso actual.

El problema del desvanecimiento del gradiente se presenta ya que en algunos casos el gradiente se irá haciendo cada vez más pequeño, llegando a un punto donde evite el cambio del peso. En el peor de los casos, esto impediría completamente que la red neuronal siga entrenando.

El algoritmo retro propagación a través del tiempo, que es el algoritmo que usa las redes neuronales recurrentes, tiene el inconveniente de que, dado que se introduce una nueva dimensión temporal, se agrava el problema de la desaparición o desvanecimiento del gradiente, particularmente para los primeros instantes de tiempo.

En las redes neuronales recurrentes los gradientes correspondientes a los últimos instantes de tiempo apenas se ven influidos matemáticamente por los de los primeros instantes. La red no memoriza. Sólo es capaz de recordar con eficiencia los instantes de tiempo más recientes y con ello, su capacidad de procesar largas secuencias se ve afectada.

La desaparición o desvanecimiento del gradiente provoca que la memoria de la red neuronal sea tan sólo a corto plazo.

Este problema de la memoria a corto plazo es denominado short-term memory.

Como ya se explicó hasta el momento, las redes neuronales recurrentes basan su funcionamiento en el procesado de secuencias de información, agregando el cómputo de cada elemento de la secuencia, el resultado del análisis del elemento anterior.
Así el resultado final de una red neuronal recurrente será una función que agrega el procesamiento de todos los elementos de la secuencia.

Pero a medida que la red procesa más elementos de la cadena, es decir la secuencia de entrada es muy larga, tiene problemas en recordar información pasada.

Es por esto que las redes neuronales recurrentes más sencillas no son capaces de aprender patrones muy extendidos en el tiempo, sino que sólo son eficaces en rangos cortos, como por ejemplo secuencias de 10 elementos.

5.2 Explosión del Gradiente

En redes profundas o redes neuronales recurrentes, los gradientes de error pueden acumularse durante una actualización y dar lugar a gradientes muy grandes. Éstos, a su vez, dan lugar a grandes actualizaciones de los pesos de la red y, en consecuencia, a una red inestable. En un caso extremo, los valores de los pesos pueden llegar a ser tan grandes como para desbordarse y dar lugar a valores NaN.

La explosión se produce por crecimiento exponencial al multiplicar repetidamente los gradientes a través de las capas de la red que tienen valores superiores a 1,0.

En las redes neuronales recurrentes, los gradientes explosivos pueden dar lugar a una red inestable incapaz de aprender de los datos de entrenamiento y, en el mejor de los casos, a una red incapaz de aprender a partir de secuencias de datos de entrada largas.

El **Gradient Clipping** es una forma de mitigar este problema al limitar el tamaño de los gradientes. La idea básica es establecer un umbral máximo para los valores de los gradientes. Si algún gradiente excede este umbral, se ajusta o se recorta a un valor máximo permitido.

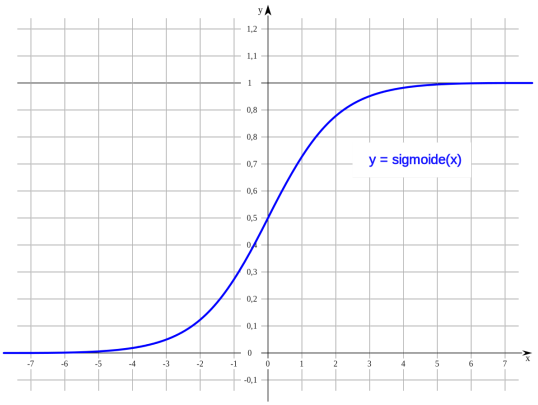
0.9	0.9
3.2	1
150	1
-2.1	-1
0.3	0.3

6. Long Short-Term Memory “y” Gated Recurrent Unit

6.1 Conceptos Previos

6.1.1 Función de activación Sigmoide

Esta función toma cualquier número real como entrada y produce una salida en el rango de 0 a 1.

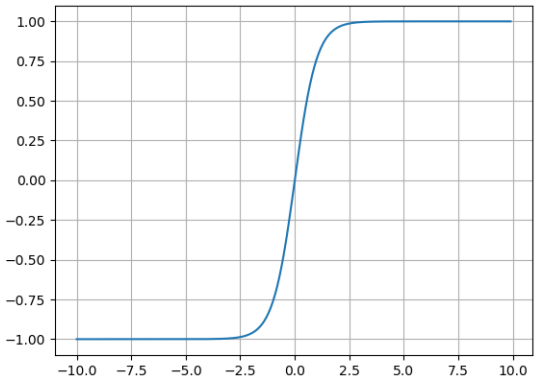


$$f(x) = \frac{1}{1 - e^{-x}}$$

Es útil para simular una compuerta.

6.1.2 Función de activación Tangente Hiperbólica (tanh)

La función tangente hiperbólica tiene una forma similar a la función sigmoide, pero su rango de salida se extiende de -1 a 1.

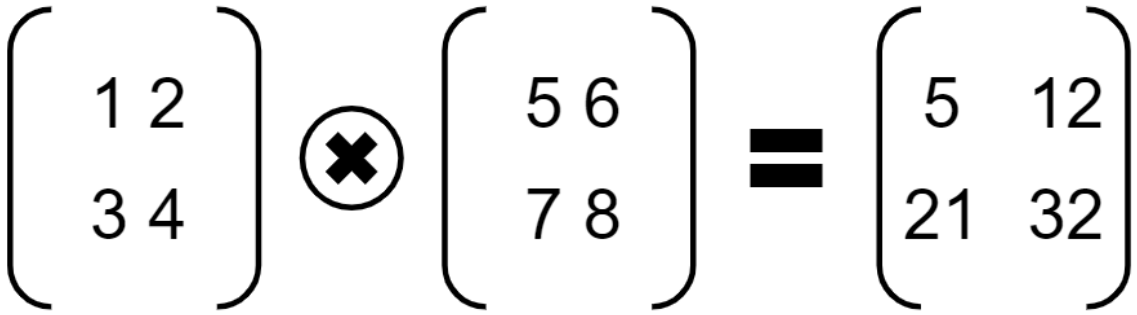


$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Es útil para acotar datos, esto también resuelve el problema de **explosión del gradiente**.

6.1.3 Producto Hadamard

En esta operación, cada elemento de una matriz se multiplica por el correspondiente elemento en la misma posición de la otra matriz, resultando en una nueva matriz del mismo tamaño.



6.2 Long Short-Term Memory (LSTM)

Uno de los problemas principales de las RNN que se ha destacado antes, es que es difícil acceder información de estados muy antiguos. En los casos en los que necesitamos un gran contexto para poder predecir, las RNN no son muy eficientes.

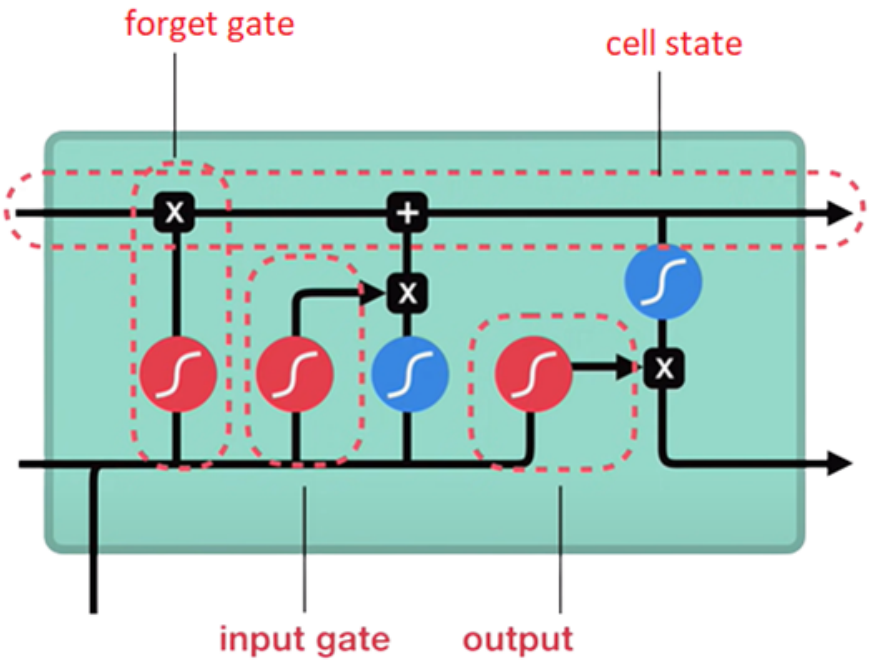
Para poder solucionar el problema del **desvanecimiento del gradiente** se usa la arquitectura **LSTM.**

Long-Short Term Memory son una extensión de las redes neuronales recurrentes, que básicamente amplían su memoria para aprender de experiencias importantes que han pasado hace mucho tiempo. Las LSTM permiten a las RNN recordar sus entradas durante un largo período de tiempo. Esto se debe a que LSTM contiene su información en la memoria, que puede considerarse similar a la memoria de un ordenador , en el sentido que una neurona de una LSTM puede leer, escribir y borrar información de su memoria.

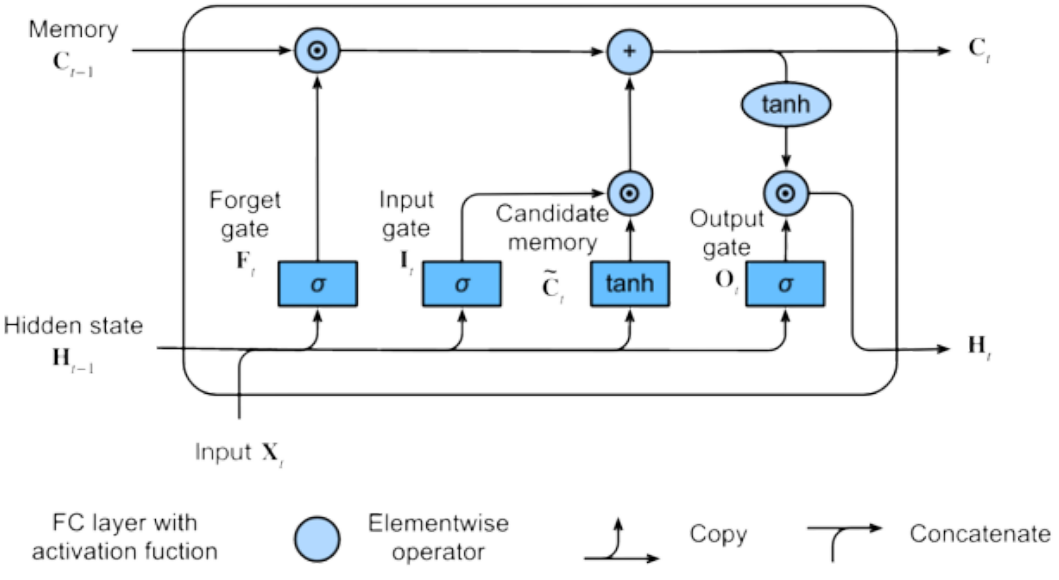
Esta memoria se puede ver como una “celda” bloqueada, donde “bloqueada” significa que la célula decide si almacenar o eliminar información dentro (abriendo la puerta o no para almacenar), en función de la importancia que asigna a la información que está recibiendo. La asignación de importancia se decide a través de los pesos, que también se aprenden mediante el algoritmo. Esto lo podemos ver como que aprende con el tiempo qué información es importante y cuál no.

En una neurona LSTM hay tres puertas a estas “celdas” de información: puerta de entrada (**input gate**), puerta de olvidar (**forget gate**) y puerta de salida (**output gate**). Estas puertas

determinan si se permite o no una nueva entrada, se elimina la información porque no es importante o se deja que afecte a la salida en el paso de tiempo actual.
Las puertas en una LSTM son análogas a una forma sigmoide, lo que significa que van de 0 a 1.



Forget Gate: Controla qué información anterior se olvida en una celda de memoria.
Input Get: Controla qué nueva información se ingresa a una celda de memoria.
Output Get: Controla qué información de la celda de memoria se utiliza como salida.
Cell State: Es la memoria a largo plazo en una red LSTM.



Para calcular el valor de la compuerta Forget, se toma el vector de entrada X se lo multiplica por la matriz de pesos W, a este producto se suma el producto entre el vector de valores ocultos de la capa anterior y su respectiva matriz de pesos, por último se suma el vector de vallas. A todo esto se le aplica la función de activación sigmoidal.

Los valores de las compuertas Input y Output se calculan de la misma manera.

Ahora calcularemos un valor candidato para ir a la memoria $\tilde{\mathbf{C}}_t$, este se calcula de la misma forma que las compuertas, pero si le aplica la función de activación tangente hiperbólica.

Luego de calcular el valor del candidato, tenemos que decidir si lo guardamos en memoria o lo desechamos, para este cálculo se usa el producto de hadamard.

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

Si **F** vale cero tiramos el valor anterior y guardamos el valor candidato, si **I** vale cero desechamos el valor candidato y nos quedamos con lo que ya teníamos en memoria.

Ahora debemos calcular el nuevo valor para el estado oculto (**H**), usando lo que tenemos en memoria.

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t)$$

Si **O** vale cero, olvidamos todo lo que sabíamos hasta el momento, y si vale uno, leemos lo que hay en memoria.

En conclusión podemos decir que **Forget Gate** limpia la memoria, **Input Gate** escribe en memoria y **Output Gate** leé la memoria.

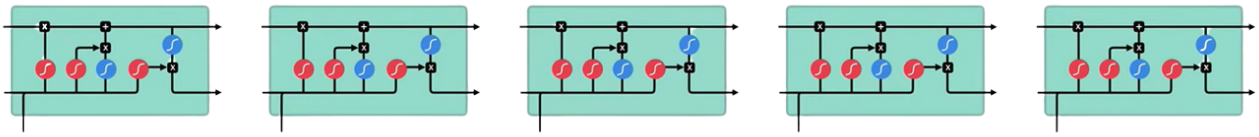
Fórmulas:

$$\begin{aligned}
\mathbf{I}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),\\
\mathbf{F}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f)\\
\mathbf{O}_t &= \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o)
\end{aligned}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c)$$

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t)$$



Red LSTM

6.3 Gated Recurrent Unit (GRU)

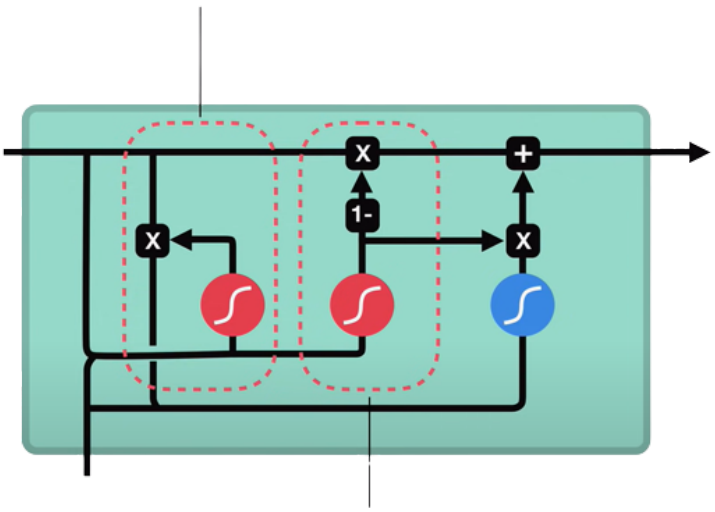
Las GRUs son neuronas recurrentes con una estructura algo diferente que las LSTM. Son más nuevas y eliminan algunas de las operaciones de las LSTM.

Las GRUs no emplean el estado de célula (cell state), solo usan el hidden state para la transferencia de información. Además, tienen 2 puertas o gates diferenciadas: la puerta de actualización o **update gate** y la puerta de reseteo o **reset gate.**

Estas puertas proporcionan a las GRUs la capacidad de adaptarse dinámicamente a diferentes secuencias y decidir qué información es relevante y cómo se debe combinar con la información existente en el estado oculto. Esta flexibilidad en el control de la información permite a las GRUs capturar dependencias a largo plazo y modelar eficazmente secuencias complejas.

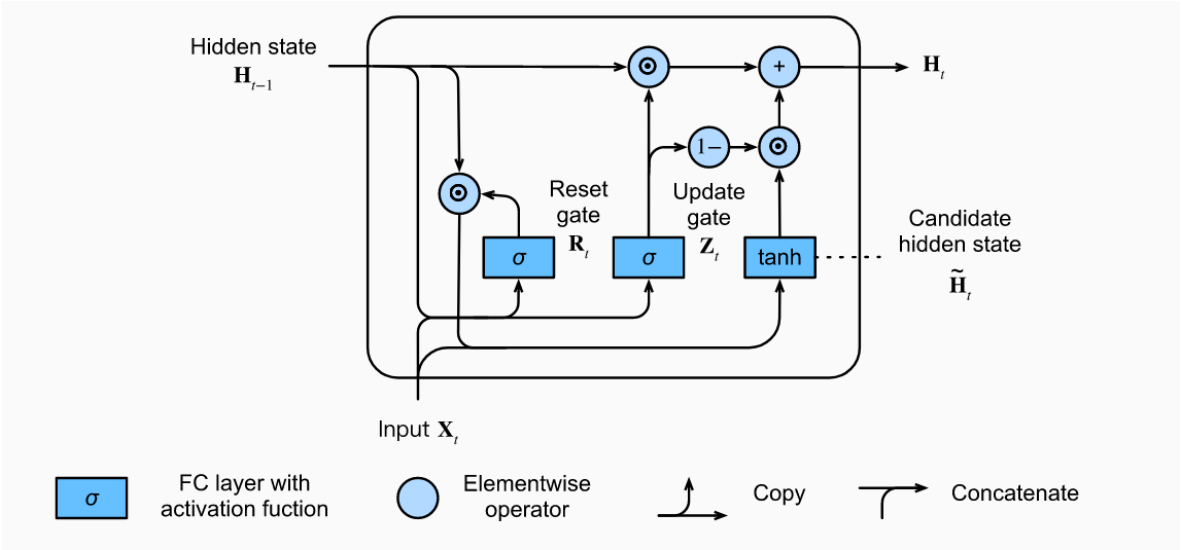
Arquitectura

Reset Gate



Update Gate

Reset Gate: Controla qué información pasada se debe olvidar en una red GRU, permitiendo que la red "reseteé" su estado interno.
Update Gate: Controla cuánta información pasada debe fusionarse con la nueva información.



El cálculo del valor de salida de las compuertas **Reset Gate** y **Update Gate** es igual que en las redes **LSTM.**
Se toma el vector de entrada **X** multiplicada por la matriz de pesos **W,** se suma el estado oculto anterior multiplicado por su matriz de pesos y se suma el vector de vallas.
Finalmente a todo esto se aplica la función de activación sigmoide.
Si el valor de Reset Gate es cero, significa que ignoramos el estado oculto anterior.
Si el valor de Update Gate es 1 queremos mantener el estado oculto anterior.

Con estos valores calcularemos el nuevo estado oculto candidato $\tilde{\mathbf{H}}_t$.

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$$

Si **R** (valor de Reset Gate) es cero, no tomamos en cuenta el valor anterior del estado oculto.

Ahora debemos decidir si desechamos este valor candidato o lo guardamos en memoria.

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$

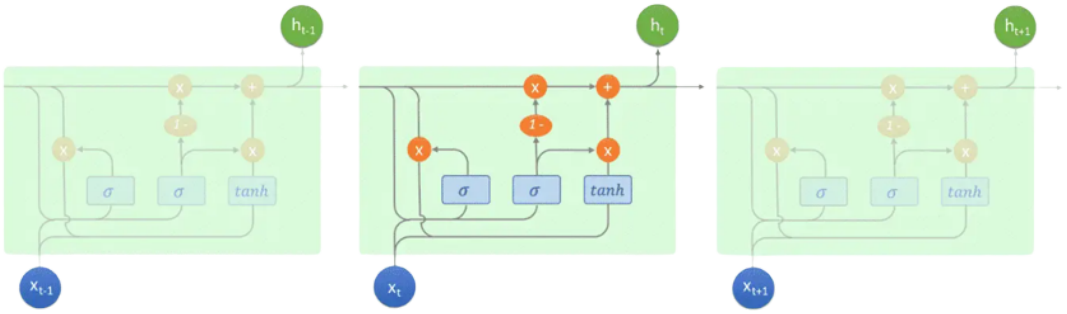
Z es 0, tiramos el estado anterior y guardamos el valor candidato.
Z es 1, tiramos el valor candidato y nos quedamos con el estado anterior.

Fórmulas:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z),$$

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$$

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$



Red GRU