

Tema 4: Arquitecturas y aplicaciones de las redes neuronales profundas



**Aprendizaje
Profundo**

Grado en Ingeniería y Ciencia de datos (Universidad de Oviedo)

Pablo González, Pablo Pérez
{gonzalezgablo, pabloperez}@uniovi.es
Centro de Inteligencia Artificial, Gijón

En la actualidad existen numerosas arquitecturas DNN creadas en función del problema que pretenden resolver.

Veremos en detalle las siguientes:

- Redes recurrentes
- Transformers
- Redes convolucionales
- Redes generativas adversarias
- Autoencoders
- Otras

Redes neuronales recurrentes (RNN)

Introducción

¿Qué son?

Redes neuronales profundas que incorporan capas con conexiones recurrentes.

Las conexiones recurrentes sirven para sacar partido de lo aprendido en instantes anteriores.



¿Para qué sirven?

Son útiles en el tratamiento de información secuencial (texto, audio, video,...).

Ejemplos:

- Reconocimiento del habla
- Análisis de sentimientos a partir de textos
- Traducción automática
- Reconocimiento de entidades
- Generación de música
- Análisis de secuencias de ADN
- etc.

Ejemplo: Predecir la palabra siguiente a partir del comienzo de una frase

Entrada: Secuencia de palabras de longitud variable.

Salida: Palabra.

Otros datos:

- Cada palabra se codifica utilizando one-hot.
- Una frase es una secuencia con número variable de vectores one-hot.
- Supongamos:
 - Vocabulario de 10000 palabras.
 - Longitud máxima de frase de 50 palabras.

Feed Forward vs RNN

Mismo problema, arquitecturas diferentes:



Matrices de pesos:

- **U**: Conexiones entre capa de entrada y recurrente.
- **V**: Conexiones de la capa recurrente con la siguiente capa.
- **W**: Conexiones de la capa recurrente con ella misma.
- **b_h**, **b_y**: Bias de la capa recurrente y de la de salida.

Funciones de activación:

- g_1 : Suele ser Tangente Hiperbólica o ReLU.
- g_2 : Sigmoide (generalmente).

Notación

Dimensiones:

- \mathbf{U} : 3×5
- \mathbf{V} : 2×3
- \mathbf{W} : 3×3
- \mathbf{b}_h : 3×1
- \mathbf{b}_y : 2×1

Ecuaciones:

- $\mathbf{h}^{(t)} = g_1(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_h)$
- $\hat{\mathbf{y}}^{(t)} = g_2(\mathbf{V}\mathbf{h}^{(t)} + \mathbf{b}_y)$



Memoria en las RNN

Partiendo de:

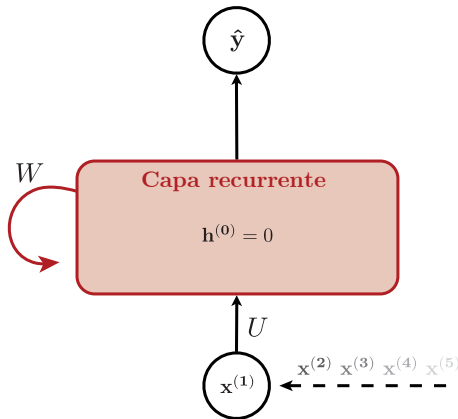
- $\mathbf{h}^{(t)} = g_1(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_h)$

Para simplificar, eliminamos activación y bias:

- $\mathbf{h}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$

Notación:

- $\mathbf{x}^{(i)}$: Vector del i-ésimo elemento de la secuencia de entrada.
- $\mathbf{h}^{(i)}$: Salida de la capa recurrente debida a la entrada $\mathbf{x}^{(i)}$, es decir, en la etapa i-ésima.



Memoria en las RNN

Partiendo de:

- $\mathbf{h}^{(t)} = g_1(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_h)$

Para simplificar, eliminamos activación y bias:

- $\mathbf{h}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$

Notación:

- $\mathbf{x}^{(i)}$: Vector del i-ésimo elemento de la secuencia de entrada.
- $\mathbf{h}^{(i)}$: Salida de la capa recurrente debida a la entrada $\mathbf{x}^{(i)}$, es decir, en la etapa i-ésima.



Memoria en las RNN

Partiendo de:

- $\mathbf{h}^{(t)} = g_1(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_h)$

Para simplificar, eliminamos activación y bias:

- $\mathbf{h}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$

Notación:

- $\mathbf{x}^{(i)}$: Vector del i-ésimo elemento de la secuencia de entrada.
- $\mathbf{h}^{(i)}$: Salida de la capa recurrente debida a la entrada $\mathbf{x}^{(i)}$, es decir, en la etapa i-ésima.



Memoria en las RNN

Partiendo de:

- $\mathbf{h}^{(t)} = g_1(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_h)$

Para simplificar, eliminamos activación y bias:

- $\mathbf{h}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$

Notación:

- $\mathbf{x}^{(i)}$: Vector del i-ésimo elemento de la secuencia de entrada.
- $\mathbf{h}^{(i)}$: Salida de la capa recurrente debida a la entrada $\mathbf{x}^{(i)}$, es decir, en la etapa i-ésima.



Memoria en las RNN

$$\mathbf{h}^{(1)} = g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right)$$

$$\mathbf{h}^{(2)} = g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(2)} + \mathbf{b}_h \right)$$

$$\mathbf{h}^{(3)} = g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(2)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(3)} + \mathbf{b}_h \right)$$

⋮

$$\mathbf{h}^{(t)} = g_1 \left(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_h \right) \Leftarrow \text{Salida de la capa recurrente.}$$

$$\mathbf{y}^{(1)} = g_2 \left(\mathbf{V}\mathbf{h}^{(1)} + \mathbf{b}_y \right) = g_2 \left(\mathbf{V} \cdot g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) + \mathbf{b}_y \right)$$

$$\mathbf{y}^{(2)} = g_2 \left(\mathbf{V} \cdot g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(2)} + \mathbf{b}_h \right) + \mathbf{b}_y \right)$$

$$\mathbf{y}^{(3)} = g_2 \left(\mathbf{V} \cdot g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(2)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(3)} + \mathbf{b}_h \right) + \mathbf{b}_y \right)$$

⋮

$$\mathbf{y}^{(t)} = g_2 \left(\mathbf{V}\mathbf{h}^{(t)} + \mathbf{b}_y \right) \Leftarrow \text{Salida de la red.}$$



Memoria en las RNN

$$\mathbf{h}^{(1)} = g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right)$$

$$\mathbf{h}^{(2)} = g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(2)} + \mathbf{b}_h \right)$$

\vdots

$$\mathbf{h}^{(t)} = g_1 \left(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}_h \right)$$

$$\mathbf{M} = \left(\begin{array}{ccc|ccc} w_{1,1} & \cdots & w_{1,n_h} & u_{1,1} & \cdots & u_{1,n_i} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ w_{n_h,1} & \cdots & w_{n_h,n_h} & u_{n_h,1} & \cdots & u_{n_h,n_i} \end{array} \right)$$

$$\mathbf{h}^{(t)} = g_1 \left(\mathbf{M} \left[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)} \right] + \mathbf{b}_h \right) \Leftarrow \text{Salida de la capa recurrente.}$$

$$\mathbf{y}^{(1)} = g_2 \left(\mathbf{V}\mathbf{h}^{(1)} + \mathbf{b}_y \right) = g_2 \left(\mathbf{V} \cdot g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) + \mathbf{b}_y \right)$$

$$\mathbf{y}^{(2)} = g_2 \left(\mathbf{V} \cdot g_1 \left(\mathbf{W} \left(g_1 \left(\mathbf{W}\mathbf{h}^{(0)} + \mathbf{U}\mathbf{x}^{(1)} + \mathbf{b}_h \right) \right) + \mathbf{U}\mathbf{x}^{(2)} + \mathbf{b}_h \right) + \mathbf{b}_y \right)$$

\vdots

$$\mathbf{y}^{(t)} = g_2 \left(\mathbf{V}\mathbf{h}^{(t)} + \mathbf{b}_y \right) \Leftarrow \text{Salida de la red.}$$

Despliegue en una RNN

Las mismas matrices \mathbf{U} , \mathbf{V} y \mathbf{W} se utilizan en distintas etapas de la secuencia de entrada:



Entrenamiento de las RNN

Backpropagation through time (BPTT):

- 1 Se presenta una secuencia de etapas con su correspondiente entrada y salida deseada a la red.
- 2 Se desarrolla la red y se aplica la propagación hacia delante (etapa a etapa), luego se calculan y acumulan los errores de cada etapa.
- 3 Se actualizan los pesos en el sentido contrario al desarrollo de la red (como si se enrollase la red de nuevo).



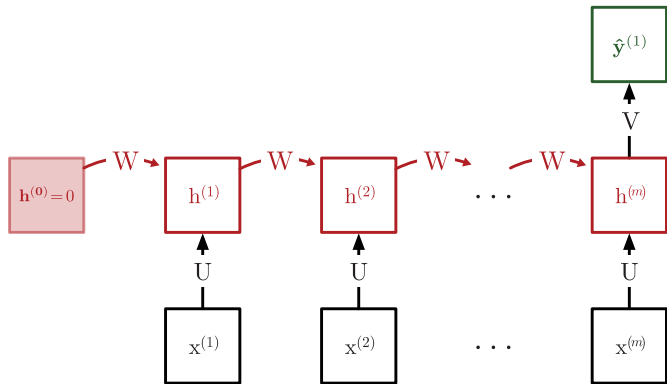
Arquitecturas de ejemplo: 1 a N

Generación automática de música: A partir de una nota de partida se genera una pieza musical, es decir, una secuencia más o menos larga de notas.



Arquitecturas de ejemplo: M a 1

Análisis de sentimientos: A partir de un texto, predecir si la opinión que refleja es favorable (positiva) o desfavorable (negativa) con respecto a un ítem que se está evaluando.



Arquitecturas de ejemplo: M a N

Traducción automática: A partir de un texto en un idioma, generar la versión traducida a otro idioma.



Exploding/Vanishing Gradients

En redes profundas, debido al gran número de productos encadenados, puede producirse los ya descritos:

- **Exploding gradients:** Crecimiento de gradientes tal que las oscilaciones durante el descenso de gradiente provocan que la red no converja.
- **Vanishing gradients:** Los gradientes se hacen tan pequeños que el descenso apenas es apreciable y la red no aprende.

¿Puede ser un problema en una red recurrente con una sola capa? → **SI**

Vanishing Gradients

- En redes **feed forward** este efecto provoca variaciones mínimas en las primeras capas.
- En las **redes recurrentes** implica que la influencia de los elementos de la secuencia de entrada se concentrará en el final de la secuencia.
 - “La *bicicleta*, siendo la alternativa de transporte urbano más ecológica, *es* poco usada”
 - “Las *bicicletas*, siendo la alternativa de transporte urbano más ecológica, *son* poco usadas”

Sería bueno que la red ‘recordase’ que estamos hablando en **singular** o en **plural**.

Redes neuronales recurrentes (RNN)

Gate Recurrent Unit (GRU)

Gate Recurrent Unit (GRU)

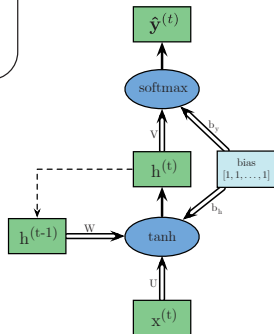
Definición

Redes recurrentes con 'gates' (puertas) que regulan el flujo de información que pasa de una etapa a la siguiente.

- **Regulación:** Transmitir desde cada neurona una salida ponderada entre su activación y la entrada recurrente.
 - El factor de ponderación se aprende y suele ser en muchos casos próximo a 0 o a 1.
- **Efecto memoria:** Puede ocurrir que entre la etapa j y la etapa k fluya la información de la etapa anterior, es decir, $\mathbf{h}_i^{(j)} \simeq \mathbf{h}_i^{(j+1)} \simeq \dots \simeq \mathbf{h}_i^{(k-2)} \simeq \mathbf{h}_i^{(k-1)}$. Esto provoca que la salida en la etapa k dependa en gran medida de la etapa j , que puede ser muy anterior.

Intuitivamente, una red GRU puede permitir que las neuronas se especialicen en determinadas propiedades de la secuencia que están tratando para mejorar la predicción.

RNN vs GRU

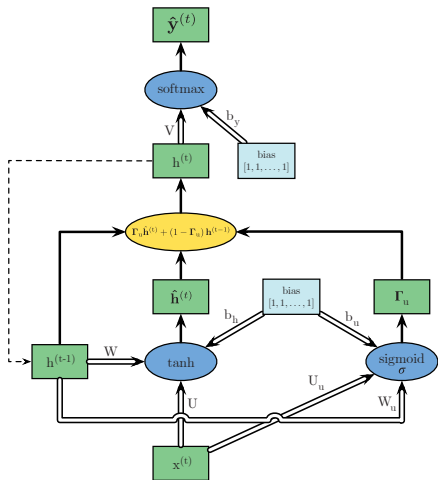


RNN simple

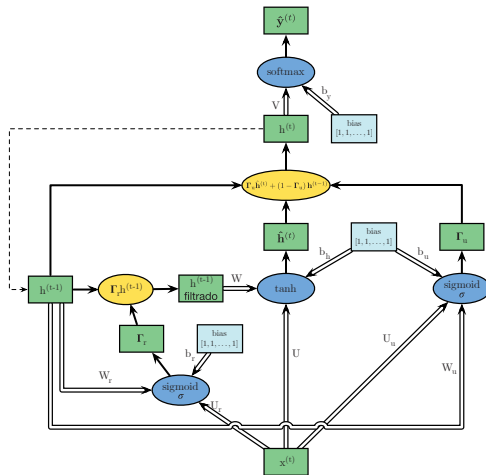


GRU

GRU con relevancia



GRU



GRU con puerta de relevancia

Notación



GRU con puerta de relevancia

$$\Gamma_u = \sigma(\mathbf{M}_u[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_u)$$

$$\Gamma_r = \sigma(\mathbf{M}_r[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_r)$$

$$\hat{\mathbf{h}}^{(t)} = \tanh(\mathbf{M}[\Gamma_r \odot \mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_h)$$

$$\mathbf{h}^{(t)} = \Gamma_r \odot \hat{\mathbf{h}}^{(t)} + (1 - \Gamma_u) \odot \mathbf{h}^{(t-1)}$$

donde:

$$\mathbf{M} = [\mathbf{W}, \mathbf{U}]$$

$$\mathbf{M}_r = [\mathbf{W}_r, \mathbf{U}_r]$$

$$\mathbf{M}_u = [\mathbf{W}_u, \mathbf{U}_u]$$

Notación



GRU con puerta de relevancia



GRU con puerta de relevancia desplegado

Redes neuronales recurrentes (RNN)

Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM)

Descripción

Se presentaron antes que las GRU (Hochreiter y Schmidhuber, 1997). Más complejas y flexibles que las redes GRU ya que cuentan con varias 'gates'.

En las redes LSTM hay dos vectores de información que se pasan de una etapa a otra:

- $\hat{\mathbf{h}}^{(t)}$: Salida de la capa LSTM.
- $\hat{\mathbf{c}}^{(t)}$: *Estado* de la capa LSTM.



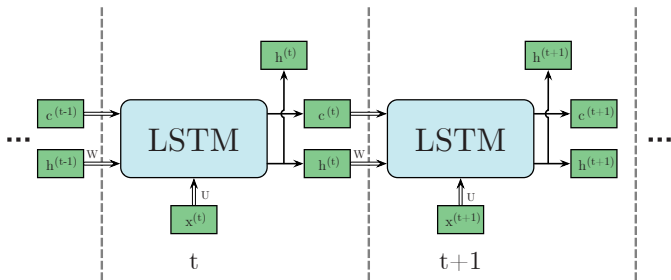
Long Short Term Memory (LSTM)

Descripción

Se presentaron antes que las GRU (Hochreiter y Schmidhuber, 1997). Más complejas y flexibles que las redes GRU ya que cuentan con varias 'gates'.

En las redes LSTM hay dos vectores de información que se pasan de una etapa a otra:

- $\hat{\mathbf{h}}^{(t)}$: Salida de la capa LSTM.
- $\hat{\mathbf{c}}^{(t)}$: *Estado* de la capa LSTM.



Notación



$$\Gamma_u = \sigma(\mathbf{M}_u[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_u)$$

$$\Gamma_r = \sigma(\mathbf{M}_r[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_r)$$

$$\Gamma_o = \sigma(\mathbf{M}_o[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_o)$$

$$\hat{c}^{(t)} = \tanh(\mathbf{M}_c[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_c)$$

$$\mathbf{c}^{(t)} = \Gamma_u \odot \hat{c}^{(t)} + \Gamma_f \odot \mathbf{c}^{(t-1)}$$

$$\mathbf{h}^{(t)} = \Gamma_o \odot \tanh(\mathbf{c}^{(t)})$$

donde:

$$\mathbf{M}_u = [\mathbf{W}_u, \mathbf{U}_u] \quad \mathbf{M}_f = [\mathbf{W}_f, \mathbf{U}_f]$$

$$\mathbf{M}_o = [\mathbf{W}_o, \mathbf{U}_o] \quad \mathbf{M}_c = [\mathbf{W}_c, \mathbf{U}_c]$$



$$\Gamma_u = \sigma(\mathbf{M}_u[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_u)$$

$$\Gamma_r = \sigma(\mathbf{M}_r[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_r)$$

$$\Gamma_o = \sigma(\mathbf{M}_o[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_o)$$

$$\hat{\mathbf{c}}^{(t)} = \tanh(\mathbf{M}_c[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_c)$$

$$\mathbf{c}^{(t)} = \Gamma_u \odot \hat{\mathbf{c}}^{(t)} + \Gamma_f \odot \mathbf{c}^{(t-1)}$$

$$\mathbf{h}^{(t)} = \Gamma_o \odot \tanh(\mathbf{c}^{(t)})$$

donde:

$$\mathbf{M}_u = [\mathbf{W}_u, \mathbf{U}_u] \quad \mathbf{M}_f = [\mathbf{W}_f, \mathbf{U}_f]$$

$$\mathbf{M}_o = [\mathbf{W}_o, \mathbf{U}_o] \quad \mathbf{M}_c = [\mathbf{W}_c, \mathbf{U}_c]$$

LSTM con peepholes:

- El cálculo de los valores de las gates también depende del estado de la capa recurrente, \mathbf{c} .
- Esta variante permite aprender con mayor precisión a partir de secuencias con diferencias temporales muy sutiles.

LSTM:

$$\Gamma_u = \sigma(\mathbf{M}_u[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_u)$$

$$\Gamma_r = \sigma(\mathbf{M}_r[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_r)$$

$$\Gamma_o = \sigma(\mathbf{M}_o[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_o)$$

LSTM con peepholes:

$$\Gamma_u = \sigma(\mathbf{M}_u[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_u)$$

$$\Gamma_r = \sigma(\mathbf{M}_r[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_r)$$

$$\Gamma_o = \sigma(\mathbf{M}_o[\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}] + \mathbf{b}_o)$$

Redes neuronales recurrentes (RNN)

Bidireccionales

Descripción

Las redes bidireccionales (Schuster y Paliwal, 1997) tienen en cuenta el contexto **posterior** a un elemento en una secuencia.

El contexto puede ser relevante:

- “Deberías ver *Roma*, es una ciudad con gran cantidad de monumentos”
- “Deberías ver *Roma*, es una película de Alfonso Cuarón que te va a encantar”

En este caso es necesario utilizar elementos posteriores a la palabra Roma para determinar de que estamos hablando.

RNN Bidireccional (BRNN)

Constituidas por:

- **Capa forward:** Celdas RNN que procesan la secuencia de inicio a fin.
- **Capa backward:** Celdas que la procesan en orden inverso.



RNN Bidireccional (BRNN)

Capas forward y backward en detalle:



Transformers

Transformers

Attention mechanisms

Transformers

Transformer

Redes convolucionales

Redes convolucionales

ResNet

Redes convolucionales

DenseNet

Redes generativas

Redes generativas

GAN

Redes generativas

WaveGAN, TextGAN

Autoencoders

Autoencoders

Denoising Autoencoders

Autoencoders

Variational Autoencoders

Otras arquitecturas

Otras arquitecturas

Redes siamesas

Otras arquitecturas

Sistemas de recomendación

Otras arquitecturas

Arquitecturas diseñadas para el procesamiento de conjuntos

Arquitecturas diseñadas para conjuntos

Existen arquitecturas de red especialmente diseñadas para tratar y modelar **conjuntos**. La clave en este tipo de problemas es que el orden de los elementos del conjunto no debería de influenciar la salida de la red. Algunas aplicaciones podrían ser:

- Detección de anomalías en un conjunto.
- Predicción de estadísticos de un conjunto.
- Cuantificación



Definición del problema:

Dado un conjunto de entrenamiento $D = (\{x_i\}_{i=1}^n, y)$, es decir, un conjunto de muestras etiquetadas, tratamos de aprender un modelo que sea capaz de predecir una nueva muestra de test $T = \{x_i\}_{i=1}^m$.

Para resolver este problema necesitamos arquitecturas que sean capaces de procesar conjuntos de datos y representarlos de manera **invariante al orden**. Es decir, la evaluación de la red para una muestra T debería de cumplir:

$$f(\{x_i\}_{i=1}^m) = f(\phi(\{x_i\}_{i=1}^m)),$$

siendo ϕ una permutación cualquier de los ejemplos de la muestra.

Arquitecturas diseñadas para conjuntos: DeepSets

DeepSets es una arquitectura de red que propone una capa de representación basada en **funciones de agregación simples**:



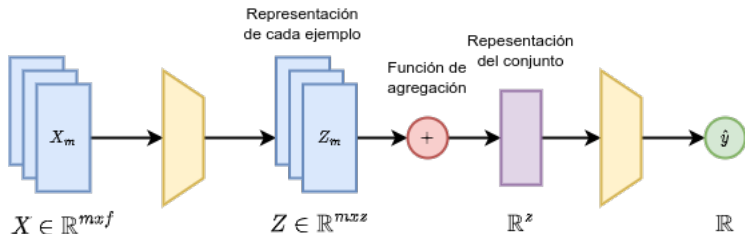
donde m sería el tamaño del conjunto de entrada y f la dimensión de cada ejemplo.

Funciones de agregación

Las funciones de agregación a utilizar pueden ser: suma, máximo, mínimo, media, mediana, etc.

Arquitecturas diseñadas para conjuntos: DeepSets

Las funciones de agregación utilizadas por DeepSets tienen como objetivo obtener una **representación única de la muestra en un espacio latente**.



A partir de esta representación la segunda parte de la red puede aprender a asociar estos vectores con su etiqueta final.

Arquitecturas diseñadas para conjuntos: DeepSets

Algunos puntos importantes sobre DeepSets son:

- La arquitectura está pensada funcionar como un **aproximador universal** de cualquier función que trabaje con conjuntos.
- La función de agregación es una manera **demasiado simple** de representar los ejemplos del conjunto, perdiéndose mucha información en el proceso.
- La función de agregación no captura **iteraciones entre los elementos del conjunto**.
- Algunas funciones de agregación como la suma pueden no sirven para **conjuntos de tamaño variable**.

Arquitecturas diseñadas para conjuntos: SetTransformers

SetTransformers es otra arquitectura diseñada para trabajar con conjuntos que explota el concepto de **atención** de los **transformers**.

- **Modelan relaciones** entre los diferentes ejemplos del conjunto.
- Aprovechan la atención para capturar **relaciones entre elementos**.
- Flexibilidad para manejar **conjuntos de tamaños variables**.
- Mejoran el rendimiento de DeepSets.

Arquitecturas diseñadas para conjuntos: SetTransformers

La idea básica de los SetTransformers es calcular la atención entre los elementos del conjunto. El principal problema de esta arquitectura es que su complejidad es $O(n \times n)$, siendo n el número de ejemplos del conjunto.

Para resolver este problema los autores de SetTransformers calculan la atención de cada ejemplo a una serie de puntos llamados **inducing points**, reduciendo por tanto la complejidad a $O(n \times i)$, siendo i el número de inducing points.

Nota importante

Ten en cuenta que **los transformers por defecto son invariantes al orden**. Es por ello que en problemas de NLP se les añade la posición de cada palabra en la frase para poder capturar esa información que es importante a la hora de procesar texto.

Arquitecturas diseñadas para conjuntos: HistNetQ

HistNetQ es una arquitectura de red pensada específicamente para **cuantificación**. En esta arquitectura se utiliza como capa de representación de la muestra **histogramas**. La idea es que un histograma es una herramienta de representación adecuada para este problema.



Cuantificación

Recuerda que la cuantificación consiste en estimar la **prevalencia de las clases** en un conjunto de test.

Arquitecturas diseñadas para conjuntos: HistNetQ

La arquitectura es similar a la de **DeepSets** pero utilizando histogramas para representar las muestras. La red calcula un histograma por cada una de las características de la representación de ejemplos.

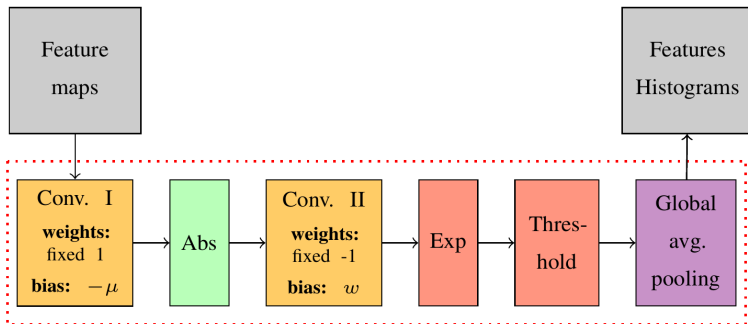


Como usar histogramas en una red

El principal problema de usar histogramas es que **el cálculo de un histograma no es una operación diferenciable**.

Arquitecturas diseñadas para conjuntos: HistNetQ

La solución es utilizar **histogramas diferenciables**.



Histogramas diferenciables

Los histogramas diferenciables utilizan operaciones comunes (como las convoluciones), para calcular una aproximación a un histograma.