Energy Consumption Forecast

An approach with Recurrent Neural Networks

Junghanss, Juan Cruz

Métodos Cuantitativos para Cs. Sociales y Negocios
Prof. Sergio Pernice, PhD, MBA.
Universidad del CEMA

Final Presentation, Junio 2021

Índice

Índice

Abstract

El presente trabajo ofrece una reinterpretación en español de las principales obras académicas sobre los pronósticos de series de tiempo con Machine Learning junto con dos aplicaciones prácticas y sus respectivos códigos. El objetivo es distinguir los alcances de los métodos estadísticos tradicionales de los de Deep Learning y cómo se integran en proyectos reales.

Abstract

Lazzeri (2020) cubre un gran espectro del know-how de las soluciones end-to-end profesionales que se ofrecen en la industria de la ciencia de datos. Como ejemplo principal aborda un pronóstico del consumo de energía con Redes Neuronales Recurrentes, que inspirará la primer aplicación práctica de este trabajo. Zhang et. al. (2020) desarrolla de principio a fin todas las herramientas de Deep Learning, su análisis cuántitativo y aplicación en código. Ng (2018) concentra brevemente las mejores prácticas recomendadas para administrar un proyecto de Machine Learning. Por último, Hyndman et. al (2018) profundiza en los principales métodos estadísticos clásicos para el pronóstico de series de tiempo, uno de los cuales (ARIMA) es empleado para la segunda aplicación práctica: un pronóstico del consumo de energía agregado por tipo de usuario en Argentina, entre 2012 y 2020.

Introducción

- La motivación del trabajo reside en el potencial existente en la rama del Machine Learning que abarca el uso de los datos de series de tiempo para poder generar predicciones.
- Si bien el forecasting de series de tiempo fue siempre una de las principales tareas de la econometría, en la actualidad aunque lo sigue siendo, también es una de las responsabilidades de los científicos de datos y expertos de Machine Learning.
- Hoy el objetivo de la presentación es ver una (muy) pequeña muestra de lo que podemos explotar con modelos de Deep Learning sin importar el tipo de industria o negocio.

Aspectos del Machine Learning

Aunque sería redundante explicar cada área del aprendizaje automático, recordemos cuáles son:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Por lo general, el pronóstico de series de tiempo se basa en "Supervised Learning".

Datos de Series de Tiempo

Debemos comprender que los datos de times series poseen los siguientes componentes:

- **Tendencia** (movimiento de largo plazo)
- **Estacionalidad** (movimiento de corto plazo)
- **Ciclos** (movimiento de corto plazo)
- **Componente estocástico** (ruido)

Dados los componentes, es posible que una serie de tiempo sea "**estacionaria**": su media y varianza son constantes a lo largo del tiempo. Esto mejora la precisión de los resultados de los algoritmos.

Aspectos del Forecasting de Series de Tiempo

¿Qué aspectos debemos tener definidos como científicos de datos antes de construir el modelo?

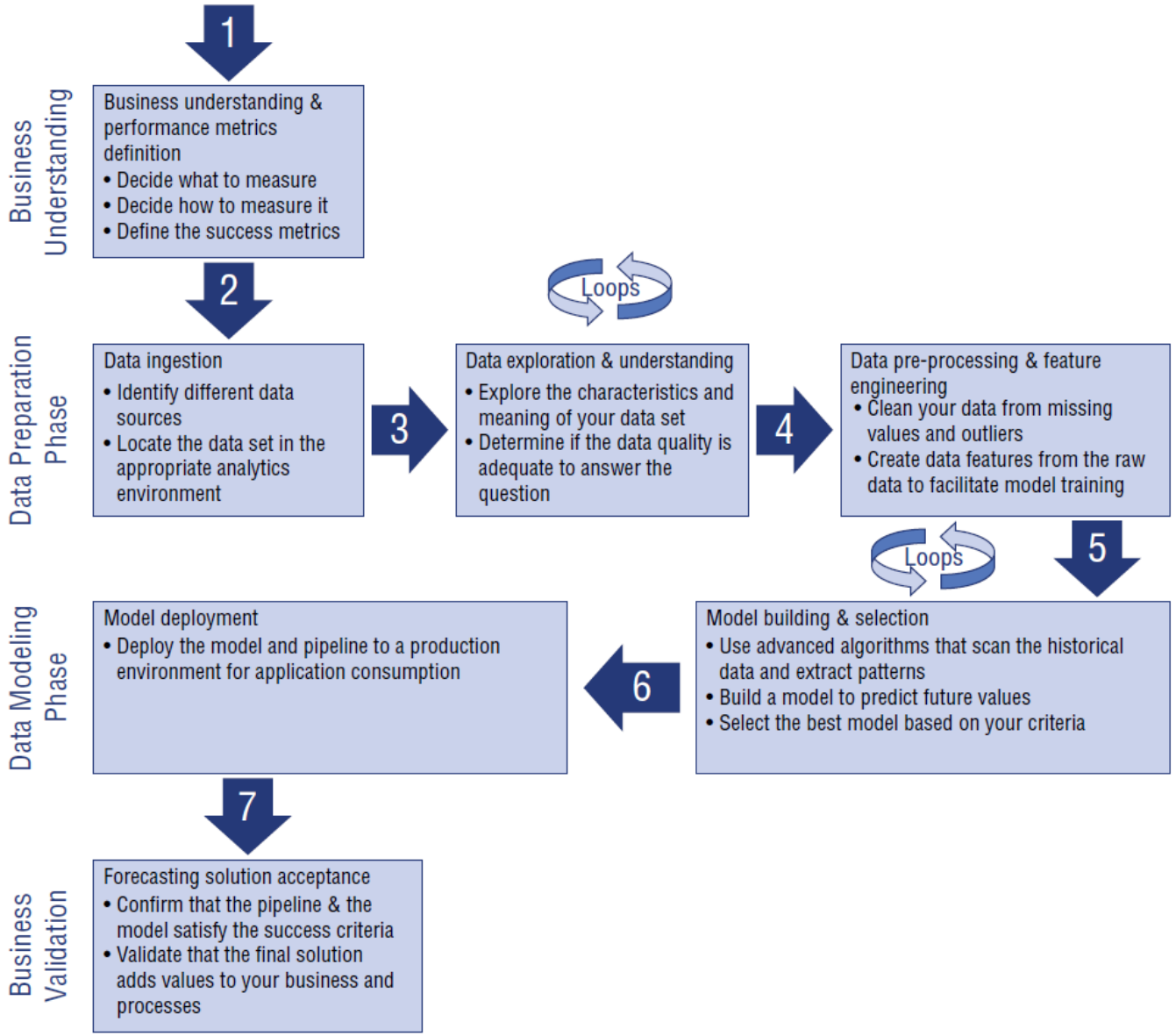
- Inputs y outputs del modelo.
- Granularidad de los datos: nivel de detalle.
- Horizonte del pronóstico: horas, días, meses, años (largo plazo).
- Endogeneidad y exogeneidad de los parámetros a usar.
- Estructura de los datos y parámetros subyacentes.
- Naturaleza univariable o multivariable del modelo.
- Estructura "multistep": cuántos períodos se pronosticarán $(t+1, t+2, ..., t+n)$
- Contiguidad de los valores de series de tiempo: existencia de missing & corrupt values

Analysis or Forecasting

Hay que distinguir las diferencias entre el **análisis** de series de tiempo y el **pronóstico** de series de tiempo.

Índice

Diseño de Soluciones Profesionales End-to-End



Esquema General para Time Series Forecasting

Para resolver problemas de negocios reales en una determinada industria, es esencial tener un esquema bien estructurado para tener como guía. El esquema de una solución profesional, de principio a fin, es:

1. Entendimiento del negocio y definición de las métricas de performance
2. Ingestión de datos
3. Exploración de datos y entendimiento
4. Pre-procesamiento de datos e Ingeniería de Parámetros
5. Construcción del modelo y selección
6. Montaje del modelo
7. Aceptación final de la solución

Técnicas de Modelización de Predicciones de Demanda

La selección de la técnica depende de muchos factores, pero de forma preliminar distingamos las principales y más populares técnicas de la estadística tradicional para diferenciarlas luego de las técnicas de machine learning.

- Moving Average (MA)
- Exponential Smoothing
- Autoregressive Integrated Moving Average (ARIMA)
- Seasonal ARIMA
- General Multiple Regression

Casos de Uso

Los principales casos de uso y aplicaciones son:

- Bolsas y mercados bursátiles
- Planificación financiera y análisis presupuestario
- Planificación de recursos
- Pronósticos de demandas
- Logística
- Manufactura o producción

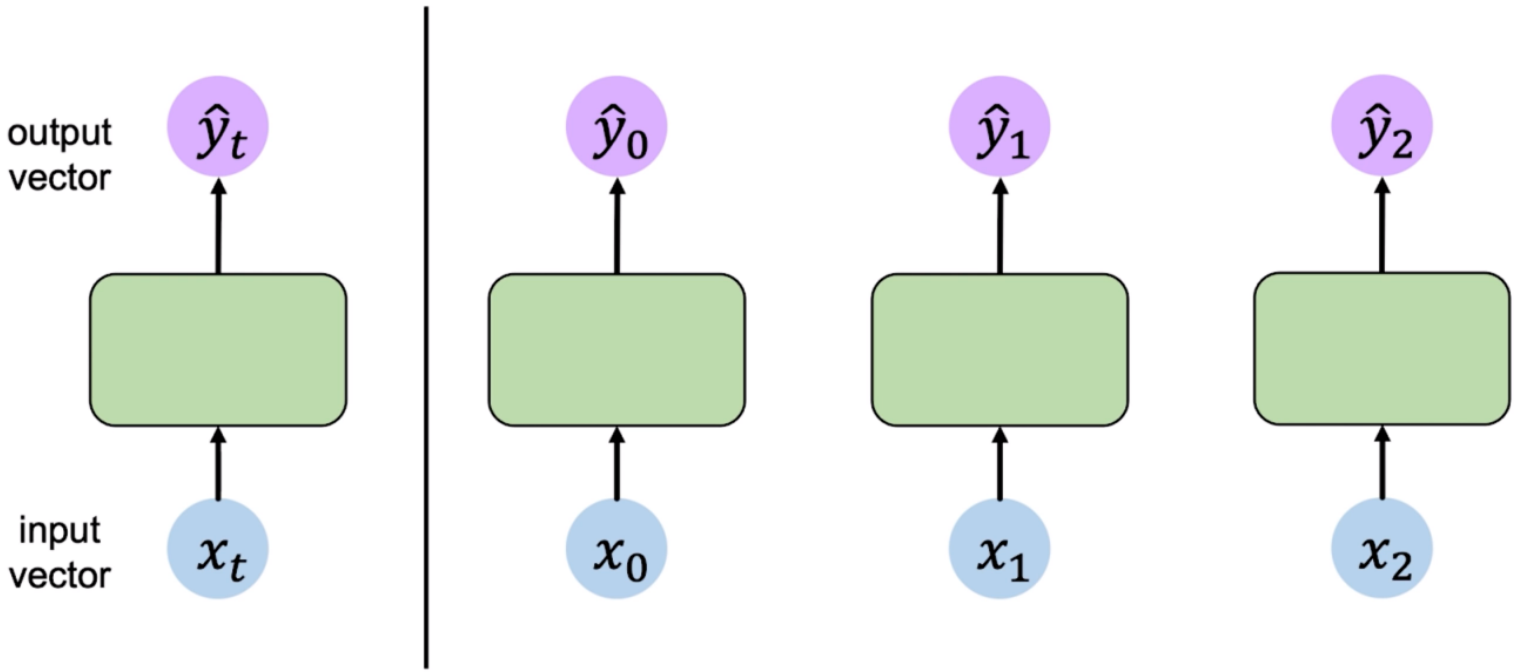
Índice

Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (RNN) son un modelo de secuencias, que a diferencia de una Neural Network clásica, puede tomar como input una **secuencia** de datos que dependen del orden o tiempo en que se distribuyen (audio, texto, video, series de tiempo).

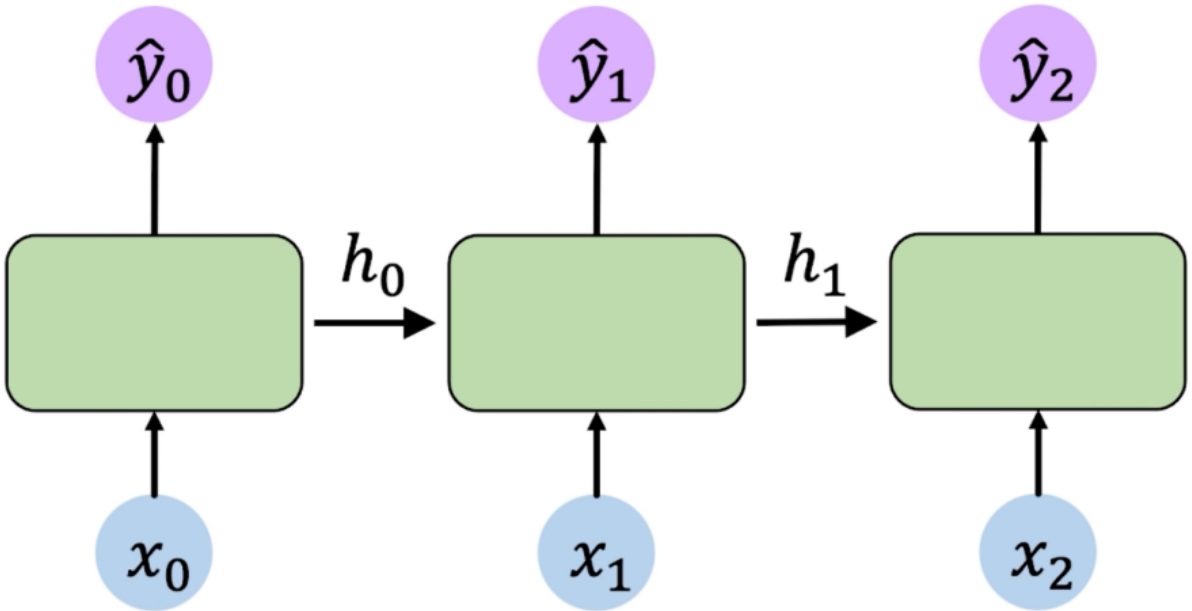
Arquitectura RNN

Pensemos la secuencia de x_t datos donde $t \in (0, 1, ..., n)$ como n datos (períodos) procesados individualmente:



Arquitectura RNN

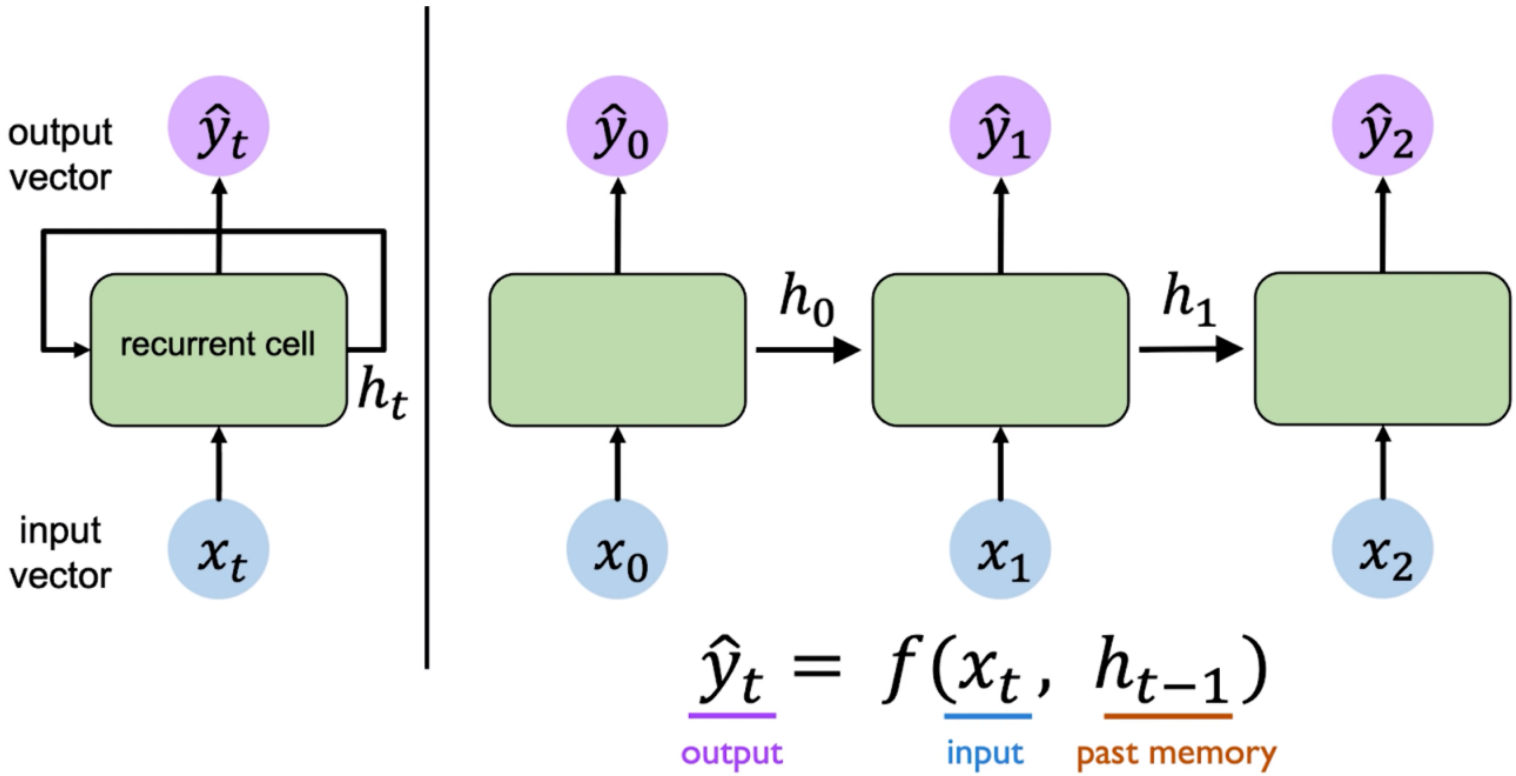
Pero... una secuencia como tal depende de su orden lógico, por lo que los outputs de cada celda deberían incorporar los datos del pasado también.



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Arquitectura RNN

Surge el concepto de "**recurrencia**", *una sola capa* RNN que procesa un input secuencial de *n* períodos se actualiza constamente con los valores del pasado y junto con el input del período actual computa un nuevo output.



Nótese que la representación compacta de la izquierda es lo mismo que la de la derecha.

Vectorización de una RNN

La memoria (llamada "estado oculto") en t se determina por el input actual en t junto con la memoria del período anterior $t-1$:

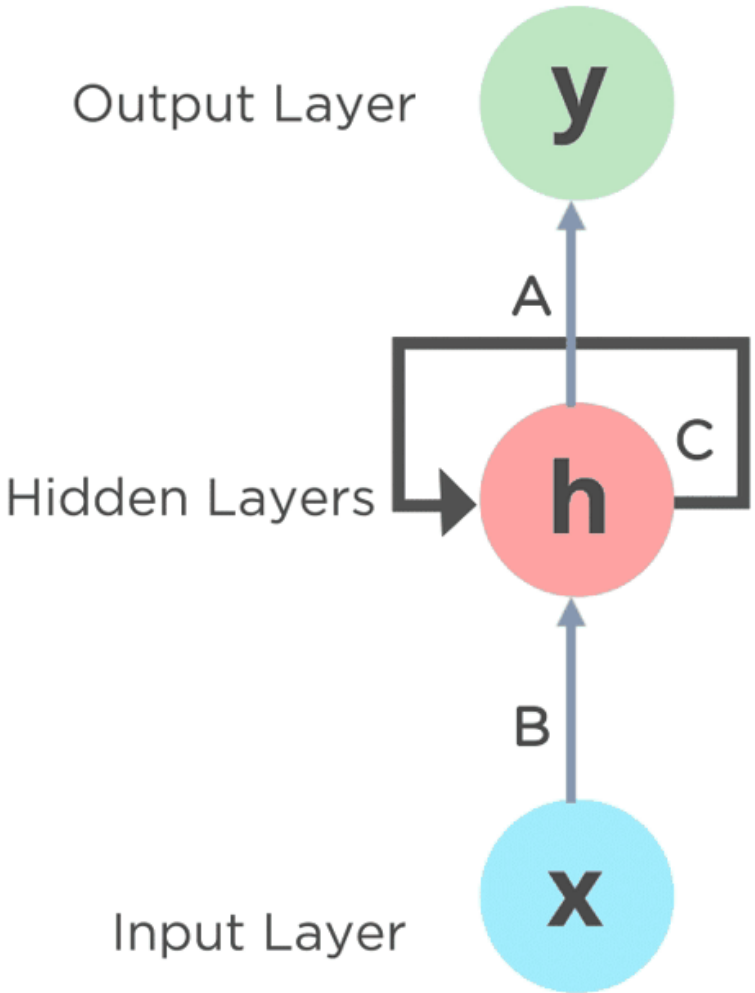
$$\mathbf{H}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h) \tag{1}$$

Y el output de la capa de salida en t se computa de la siguiente manera:

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q \tag{2}$$

Los parámetros de la RNN incluyen los ponderadores $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$, $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$, y el sesgo de la capa oculta $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$, junto con los ponderadores $\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ y el otro sesgo $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ de la output layer. En períodos diferentes, se usan siempre estos *mismos parámetros*, por eso el costo de parametrización de una RNN no crece con el número de períodos en el tiempo.

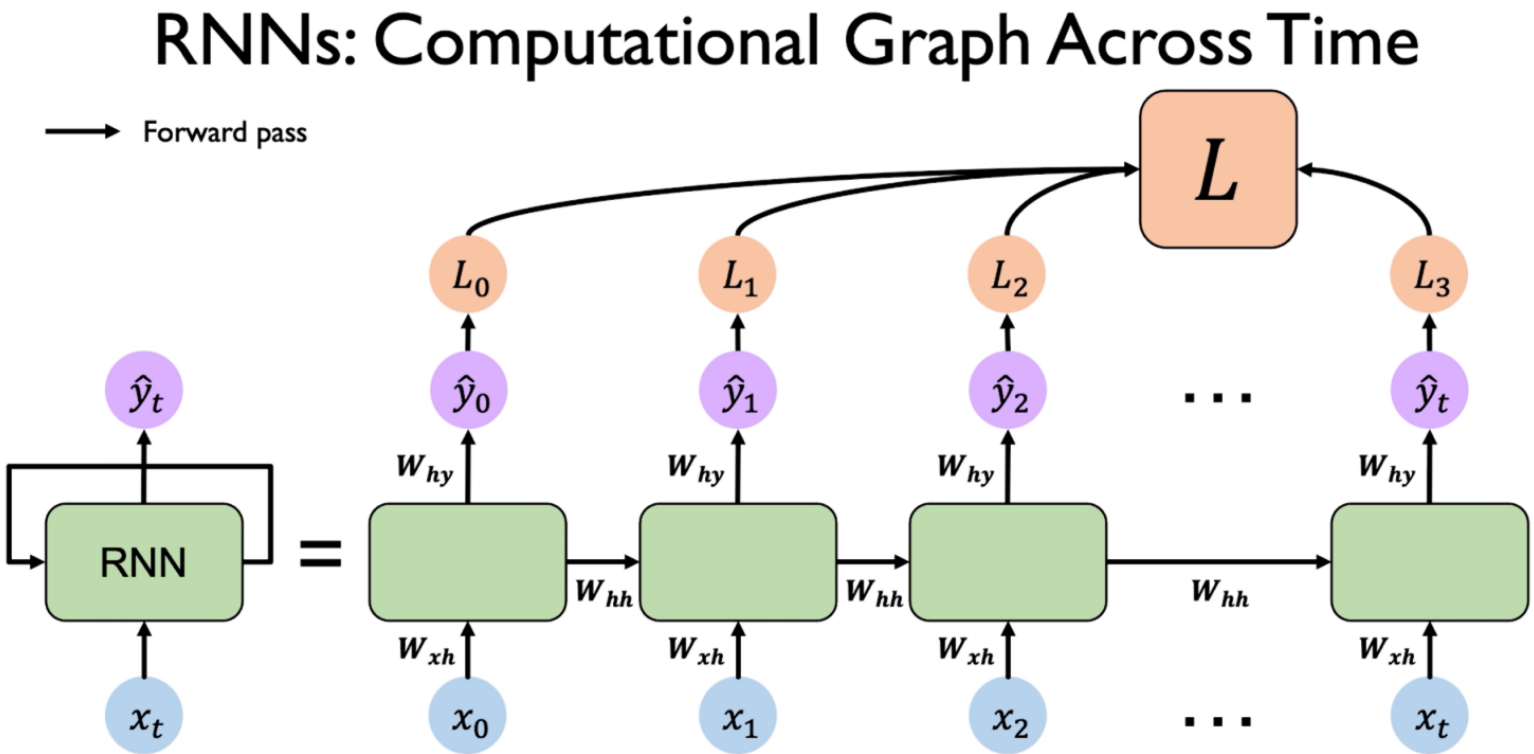
Intuición Arquitectura RNN



A, B and C are the parameters

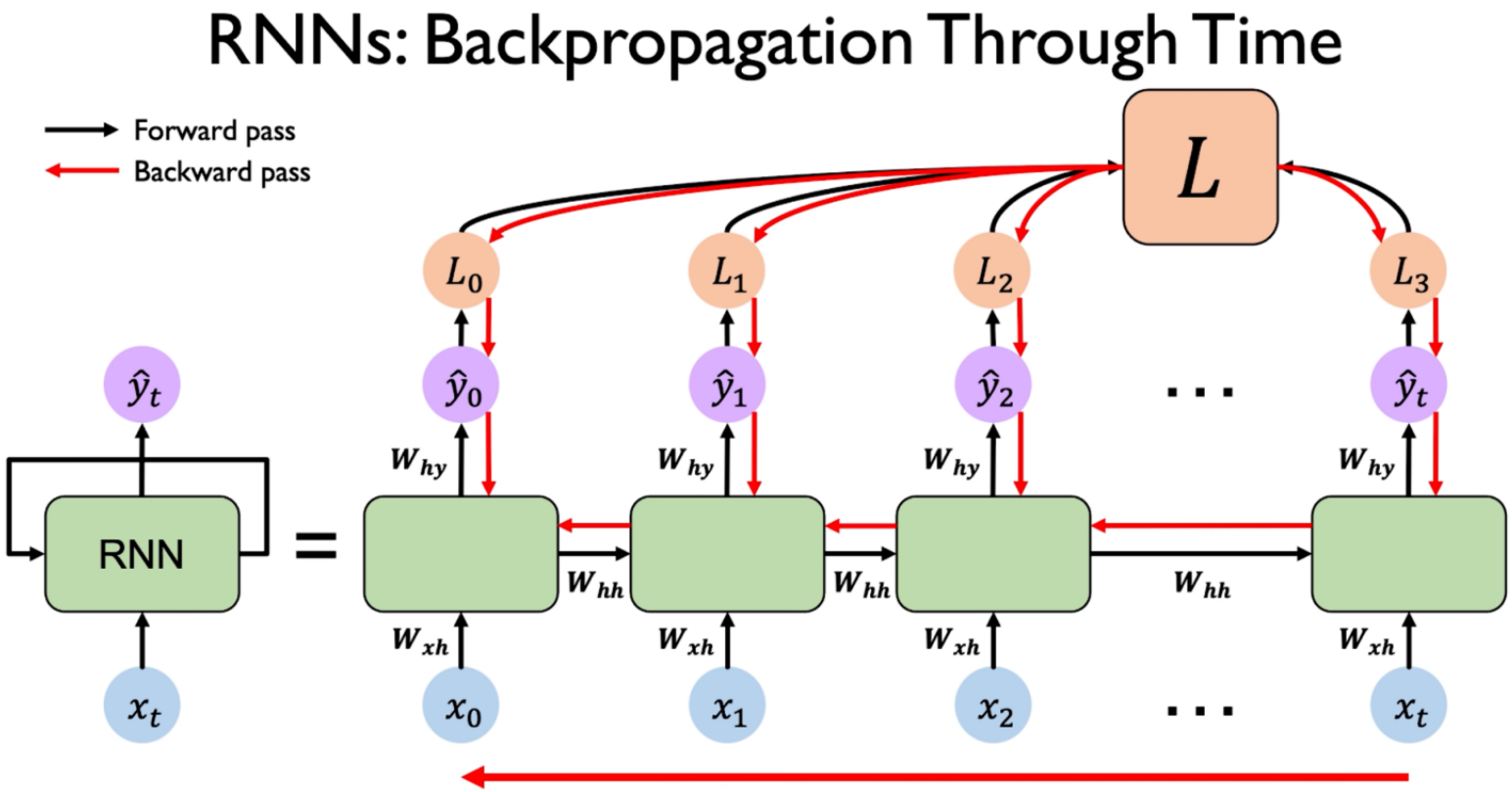
Forward propagation en RNN

Veamos la propagación hacia adelante: una sola capa RNN (lado izquierdo) puede pensarse como una secuencia de la misma celda operando en diferentes momentos y computando una pérdida L_t. La pérdida total L surge de todas las pérdidas de cada output.



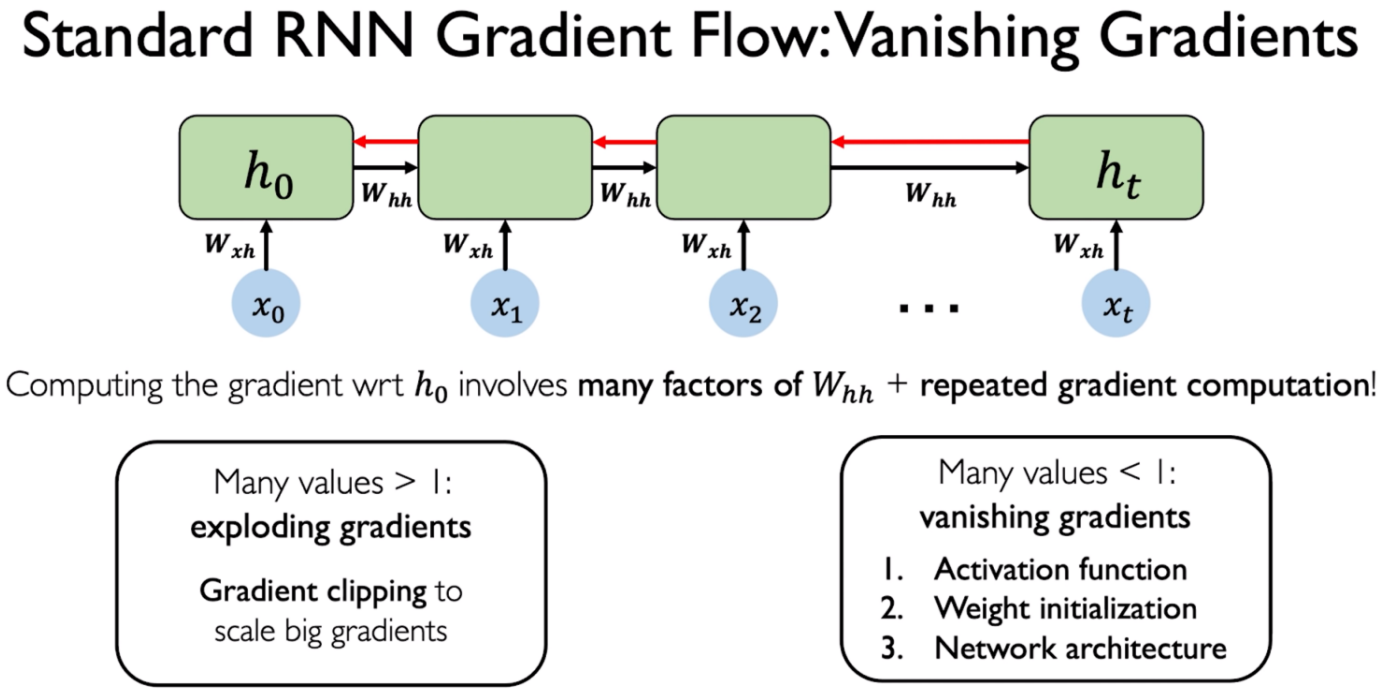
Back propagation en RNN

Veamos la propagación hacia atrás: se minimiza la función de pérdida aplicando el gradiente descendiente (u otro algoritmo de optimización) desde el fin al principio de la secuencia.



Vanishing Gradients en RNN

Sin embargo... cuando hay muchos números menores a 1, en tantas multiplicaciones el gradiente comienza a achicarse sucesivamente llegando a valores cuasi infinitesimales. En otras palabras, el principio de la secuencia no captura el aprendizaje.



Vanishing Gradients en RNN

En secuencias muy largas las RNN sufren de lo que se llama "amnesia"...
¿Cómo podemos solucionar este problema?

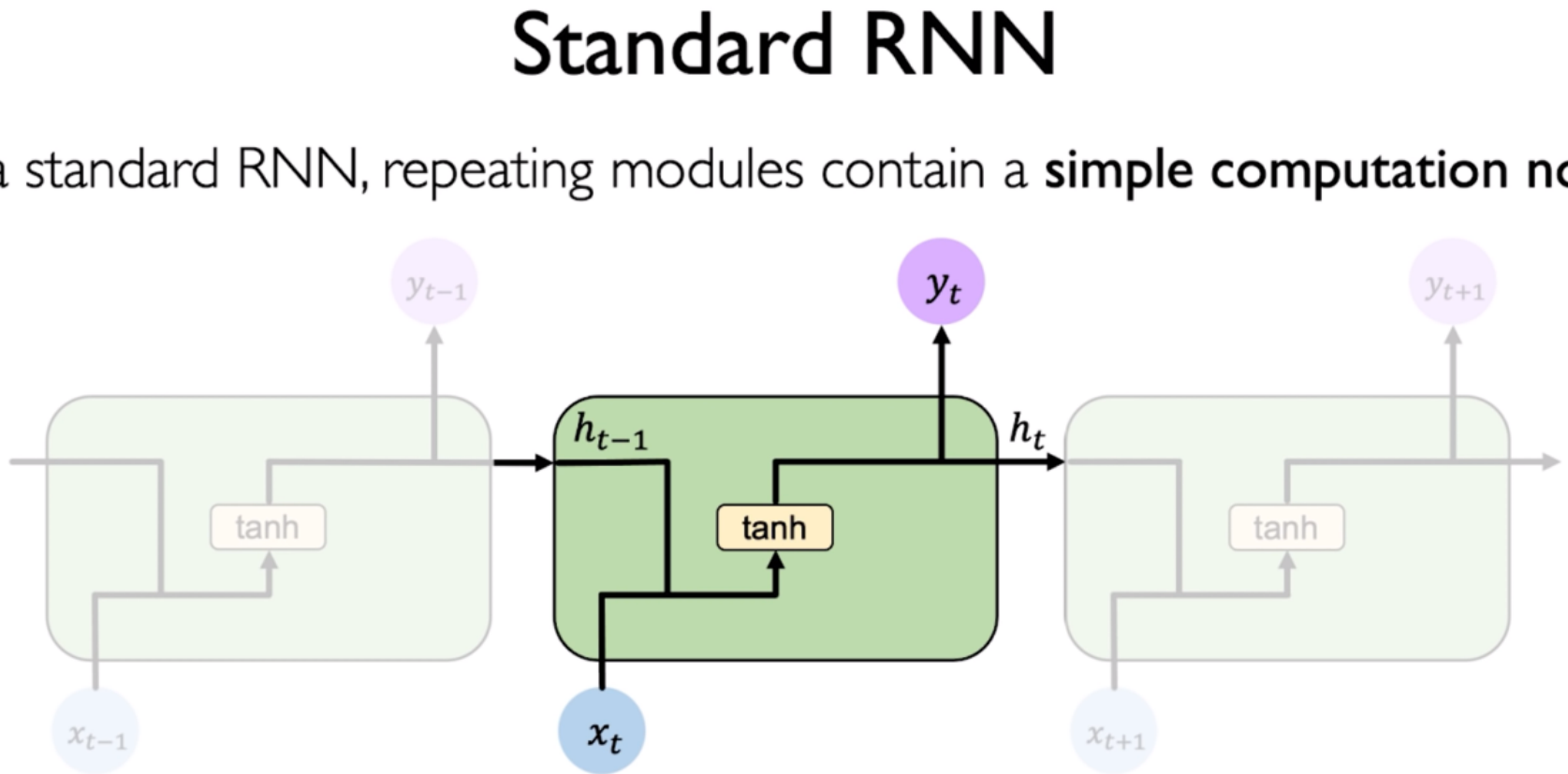
Vanishing Gradients en RNN

Las posibles soluciones a este problema pueden ser:

1. Usar ReLU activation function: su derivada es 1 para cualquier valor $x > 0$ y por ende no achica los gradientes.
2. Inicializar los parámetros como matriz identidad y los sesgos como ceros. Esto previene que los parámetros disminuyan a cero.
3. Usar una unidad recurrente más compleja con puertas que controlan la información: actualmente, las principales estructuras de RNN son las Long Short Term Memories (LSTM) y Gated Recurrent Units (GRU).

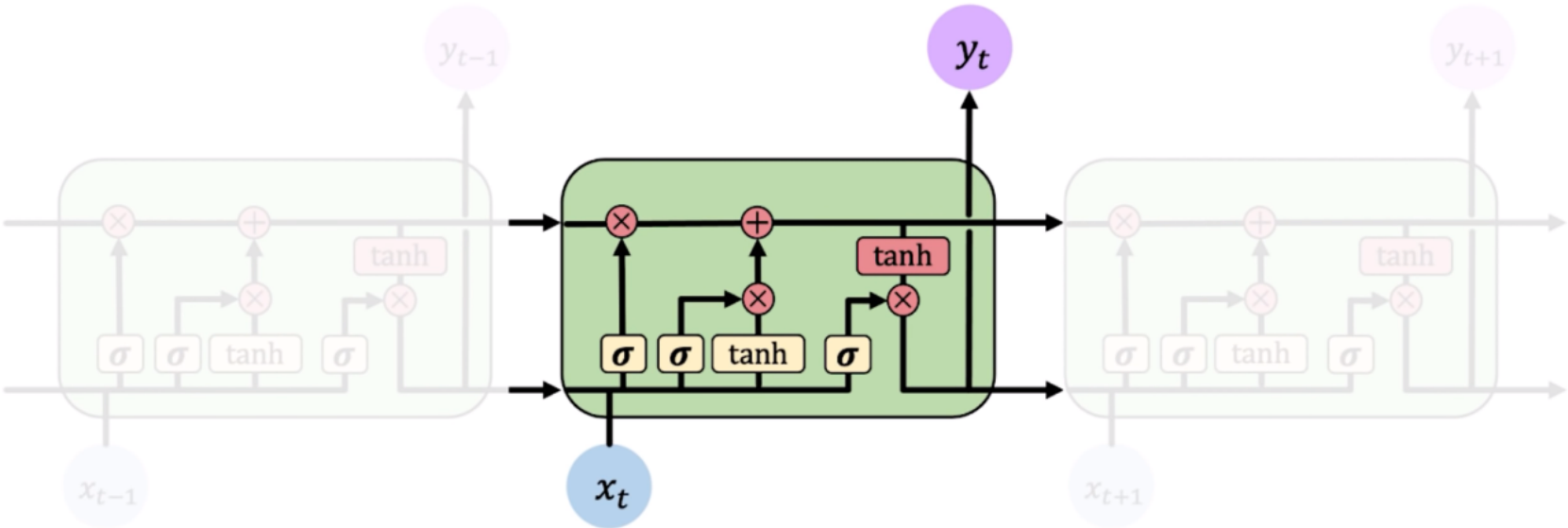
Long Short-Term Memory

Recordemos:



Long Short Term Memory (LSTMs)

LSTM modules contain **computational blocks** that **control information flow**



Long Short-Term Memory

¿Cómo trabaja una Long Short-Term Memory a través de sus puertas?

1. Decidir cuánta información **olvidar** (y cuánta recordar):

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f)$$

2. Decidir cuánto **almacenar** al estado de la celda de memoria:

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

3. **Actualizar** la celda de memoria:

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

4. Computar la **salida**:

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o)$$

Vectorización de una LSTM

Matemáticamente, supongamos que hay h unidades ocultas, el batch size es n, y el número de inputs es d. El input será $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ y el estado oculto del período previo es $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$. Las puertas en el período t se definen cómo a continuación:

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i), \tag{3}$$

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f), \tag{4}$$

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o), \tag{5}$$

Vectorización de una LSTM

La celda de memoria se construye como:

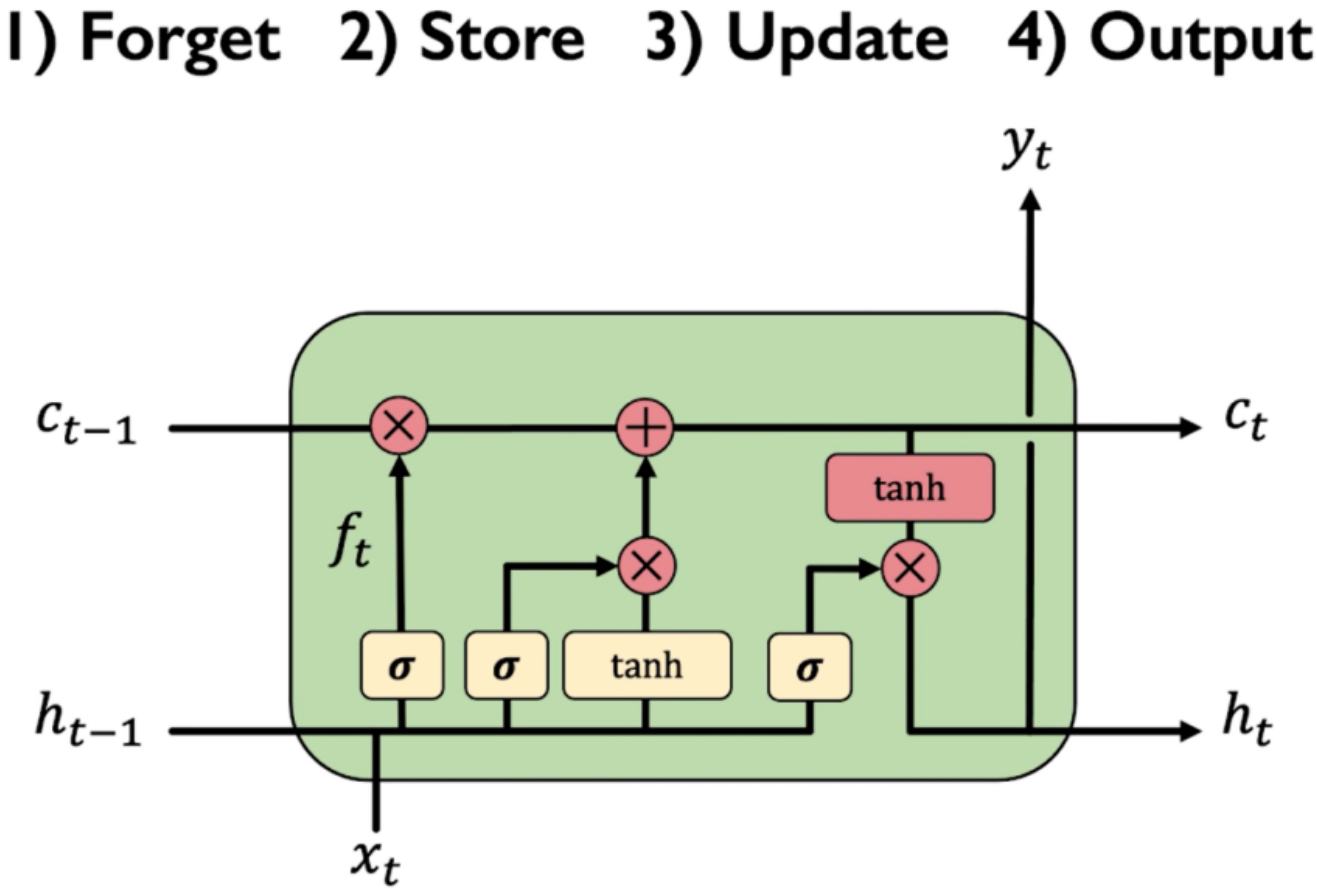
$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c), \tag{6}$$

donde usa una función $\tanh$ para trabajar en un rango de valores (-1,1). Usando el producto Hadamard (recordar que es el operador elementwise $\odot$), llegamos a la siguiente ecuación de actualización:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t \tag{7}$$

Si la puerta de olvido es siempre aproximadamente 1 y la puerta de entrada es siempre aproximadamente 0, las celdas de memoria pasadas $\mathbf{C}_{t-1}$ se guardarán con el tiempo y pasarán período de tiempo actual t.

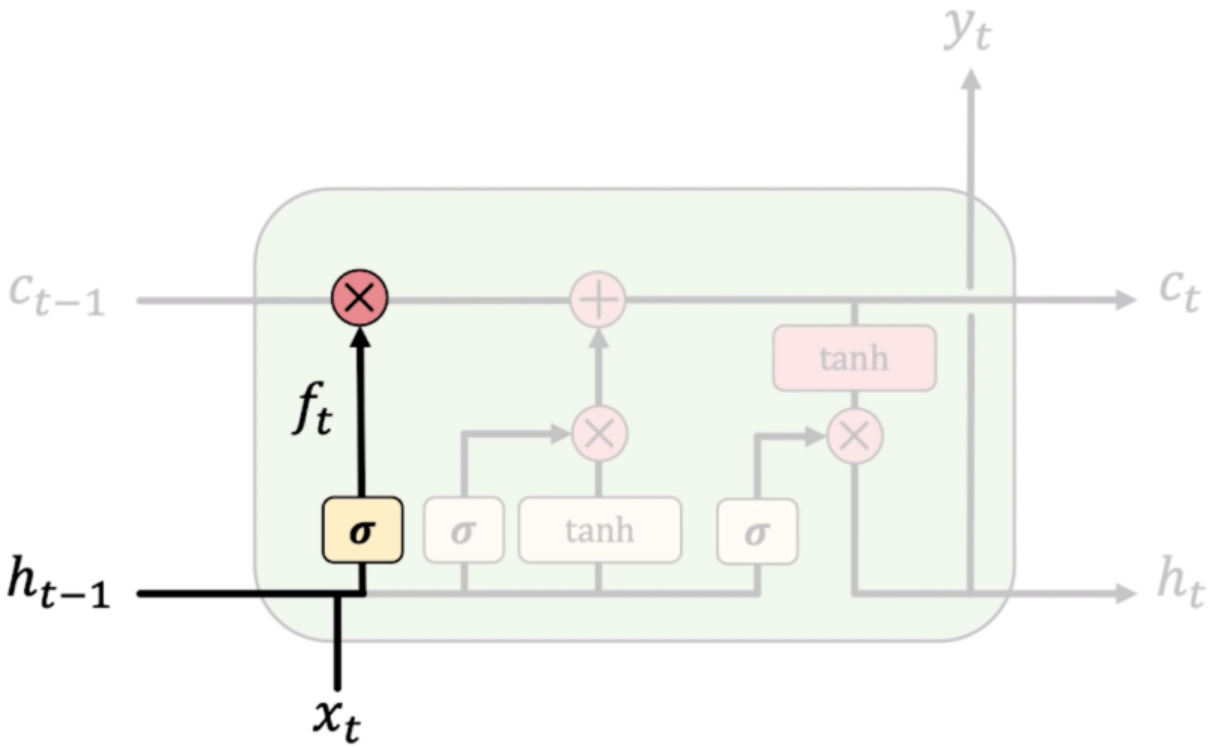
Long Short-Term Memory



Long Short-Term Memory

1) Forget 2) Store 3) Update 4) Output

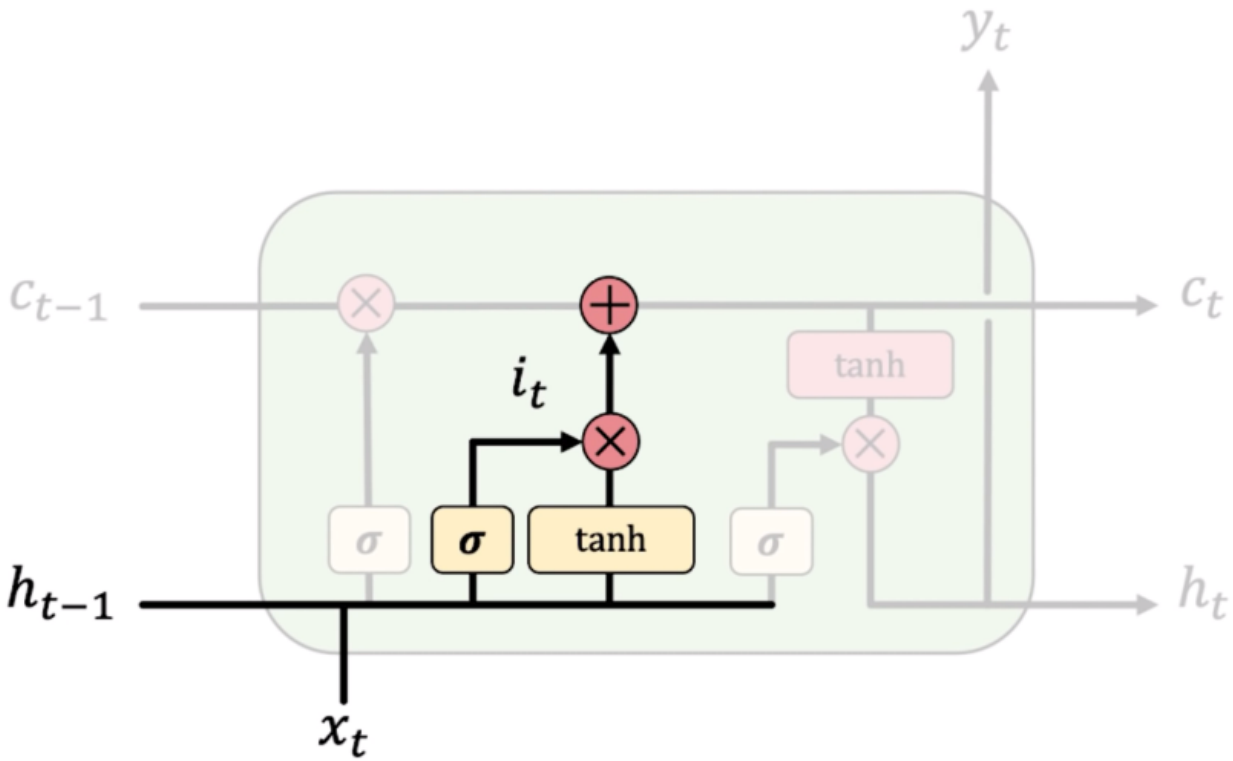
LSTMs **forget irrelevant** parts of the previous state



Long Short-Term Memory

1) Forget **2) Store** 3) Update 4) Output

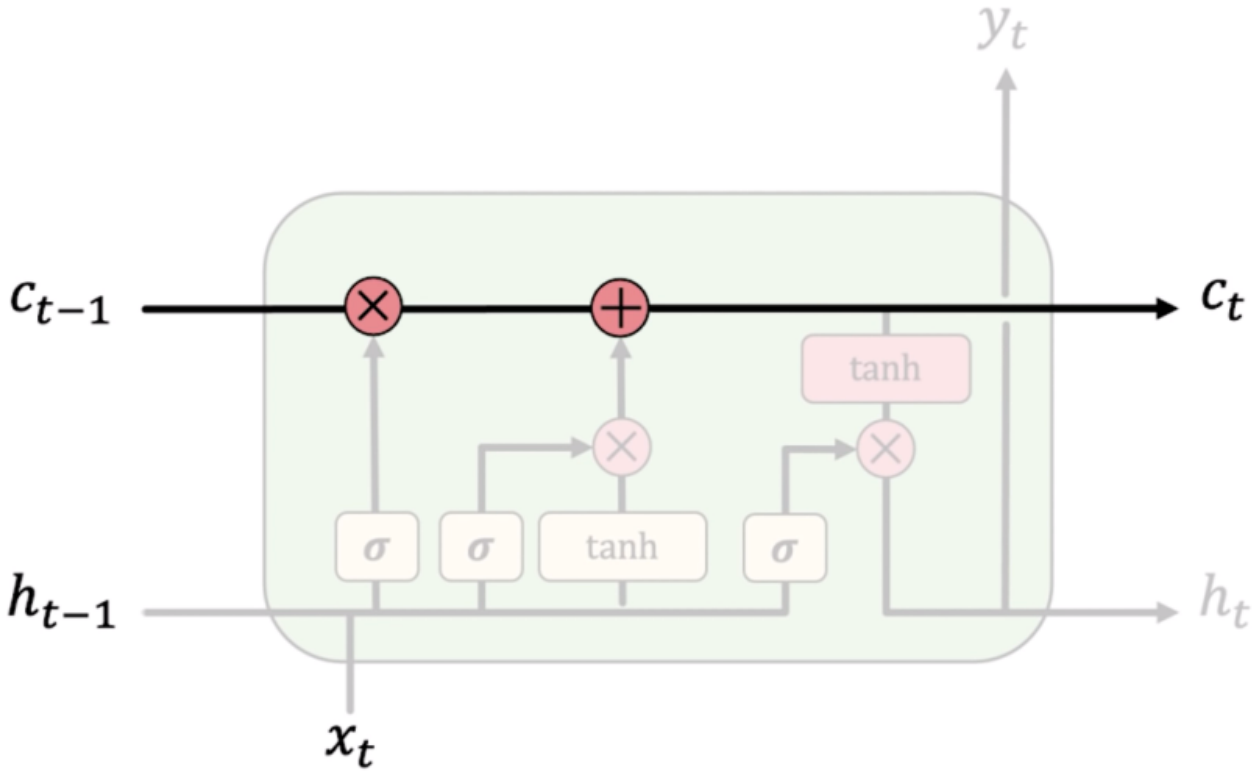
LSTMs **store relevant** new information into the cell state



Long Short-Term Memory

1) Forget 2) Store **3) Update** 4) Output

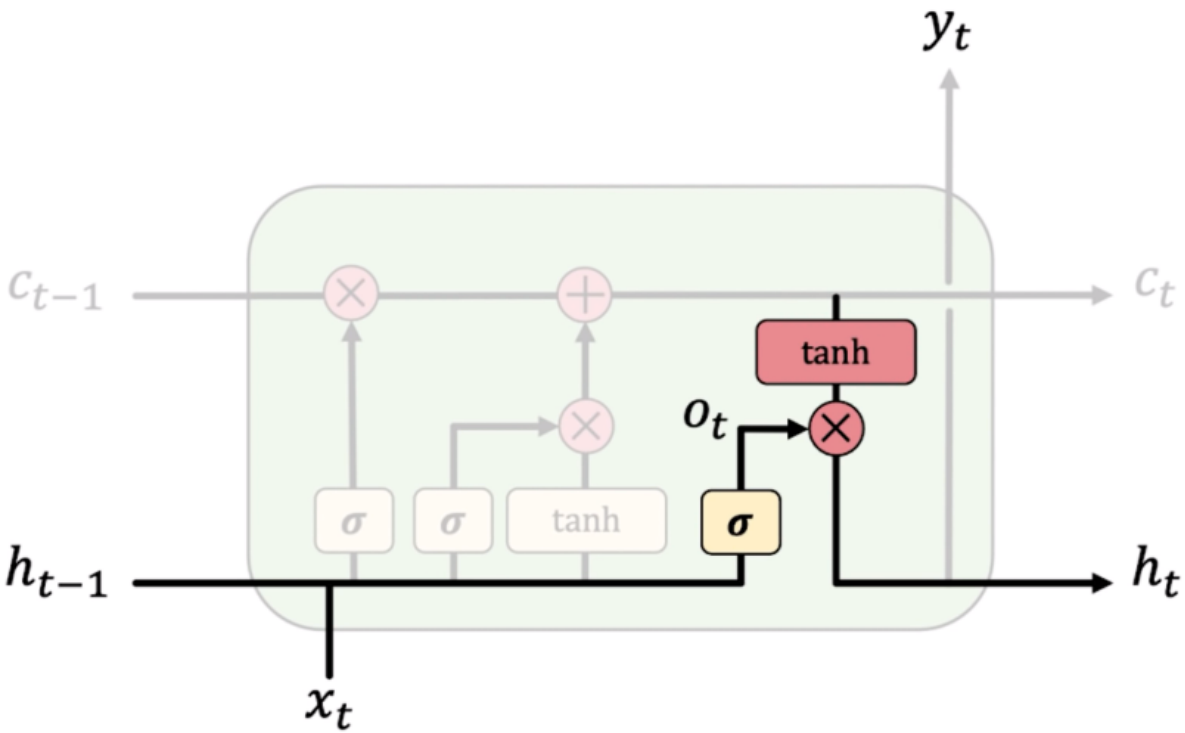
LSTMs **selectively update** cell state values



Long Short-Term Memory

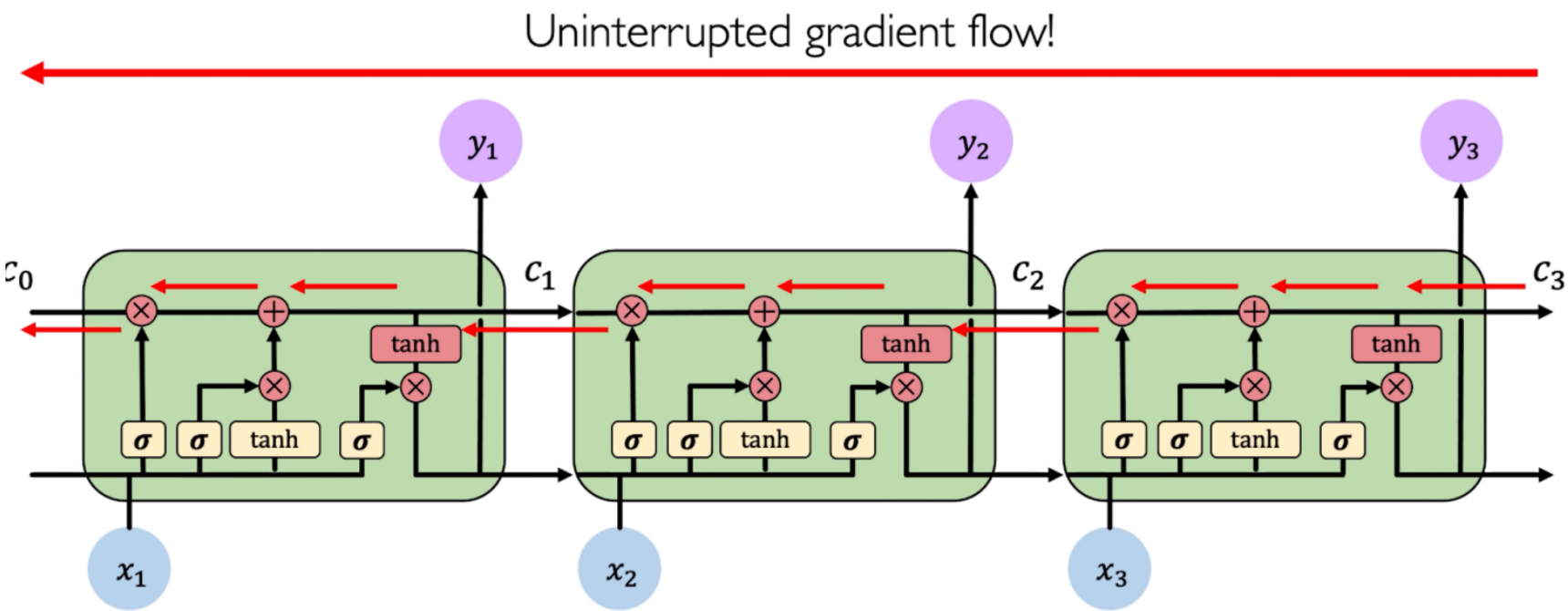
1) Forget 2) Store 3) Update **4) Output**

The **output gate** controls what information is sent to the next time step



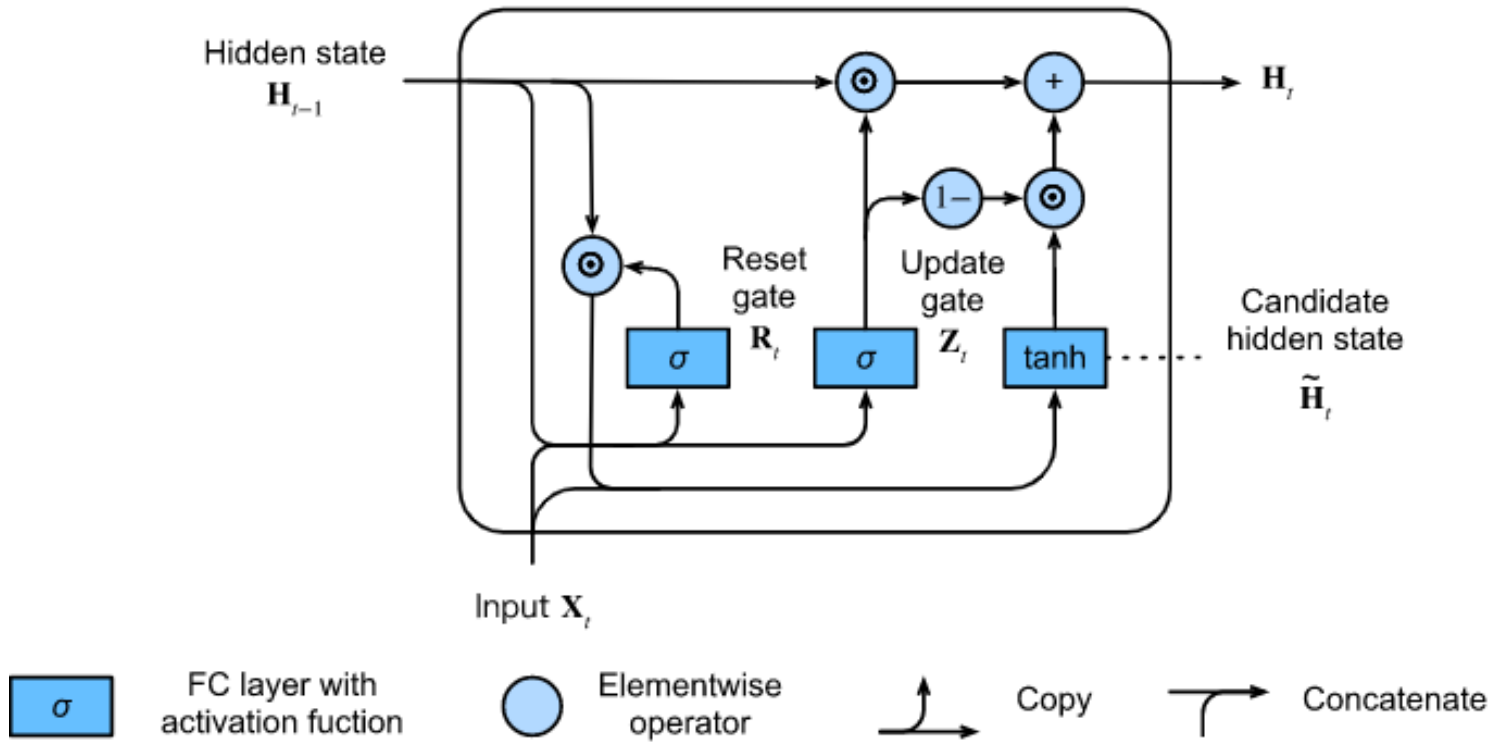
LSTM Gradient Flow

Y... ¿las puertas afectan de algún modo a la retropropagación del gradiente? En absoluto, hay un flujo constante del gradiente como en cualquier otra estructura.



Gated Recurrent Unit

Las GRU son una variación más reciente de las LSTM con menos parámetros y por ende, más eficientes. Las redes GRU no aprovechan el estado de la celda $\mathbf{C}_t$ y utilizan solo el estado oculto $\mathbf{H}_t$ para transferir información. A diferencia de LSTM, las GRU contienen solo dos puertas.



Gated Recurrent Unit

¿Cómo trabaja una Gated Recurrent Unit a través de sus puertas?

1. Decidir cuánta información **olvidar** para construir el estado oculto candidato:

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

2. Decidir qué información **actualizar** para el estado oculto final:

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

Vectorización de una GRU

Matemáticamente, para un período de tiempo t, supongamos que el input es un minibatch $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ (número de ejemplos: n, número de entradas: d) y que el estado oculto del período anterior es $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$ (número de unidades ocultas: h). Entonces, la puerta de reset o reinicio es $\mathbf{R}_t \in \mathbb{R}^{n \times h}$ y la de update o actualización es $\mathbf{Z}_t \in \mathbb{R}^{n \times h}$ y se computan como a continuación:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r), \tag{8}$$

$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z), \tag{9}$$

donde $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d \times h}$ y $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h \times h}$ son los ponderadores y $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1 \times h}$ los sesgos. Nótese que una operación de broadcasting es empleada durante esta suma. Usamos una función sigmoidea (σ) para transformar los valores en un intervalo (0,1).

Vectorización de una GRU

Debemos integrar la puerta de reinicio $\mathbf{R}_t$ con el mecanismo de actualización de estado latente (como en la ecuación 1), que conlleva al siguiente candidato de estado oculto $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ en el período t:

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h) \tag{10}$$

usamos una nolinearidad de la forma de **tanh** para asegurarnos que los valores en el estado oculto candidato permanecen en el intervalo (-1,1). Finalmente, debemos incorporar el efecto de la puerta de actualización $\mathbf{Z}_t$. Este determina hasta qué punto el nuevo estado oculto $\mathbf{H}_t$ es solamente el estado oculto anterior $\mathbf{H}_{t-1}$ y en qué medida se utiliza el nuevo estado oculto candidato $\tilde{\mathbf{H}}_t$.

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t \tag{11}$$

Índice

Energy Consumption Forecast with RNN

Dentro del sector energético, podría haber muchas formas en las que la previsión de la demanda puede ayudar a resolver problemas empresariales críticos. De hecho, la previsión de la demanda puede considerarse la base de muchos casos de uso básicos en la industria. En general, consideramos dos tipos de previsiones de demanda de energía: a **corto plazo** (STLF) como 1 a 24 horas y **largo plazo** (LTLF) como semanas a meses. Cada una puede tener un propósito diferente y utilizar un enfoque diferente. La principal diferencia entre los dos es el horizonte de pronóstico.

Base de datos

Para esta aplicación específica de Deep Learning, utilizaremos el set de datos públicos "Load Forecasting Data" de la competencia GEFCom2014. La consigna principal de dicha competencia de pronóstico de energía de GEFCom2014 se centró en el pronóstico probabilístico de esta. Los datos completos se publicaron como apéndice del paper GEFCom2014: (Hong et al.2016). Este set de datos consta de tres años de valores de carga eléctrica por hora y valores de temperatura entre 2012 y 2014.

Librerias empleadas de Python

Para esta aplicación haremos uso de las siguientes librerias:

- NumPy: computación científica.
- Matplotlib: visualización de datos.
- Pandas: procesamiento y análisis de datos.

Y los frameworks usados para Machine Learning:

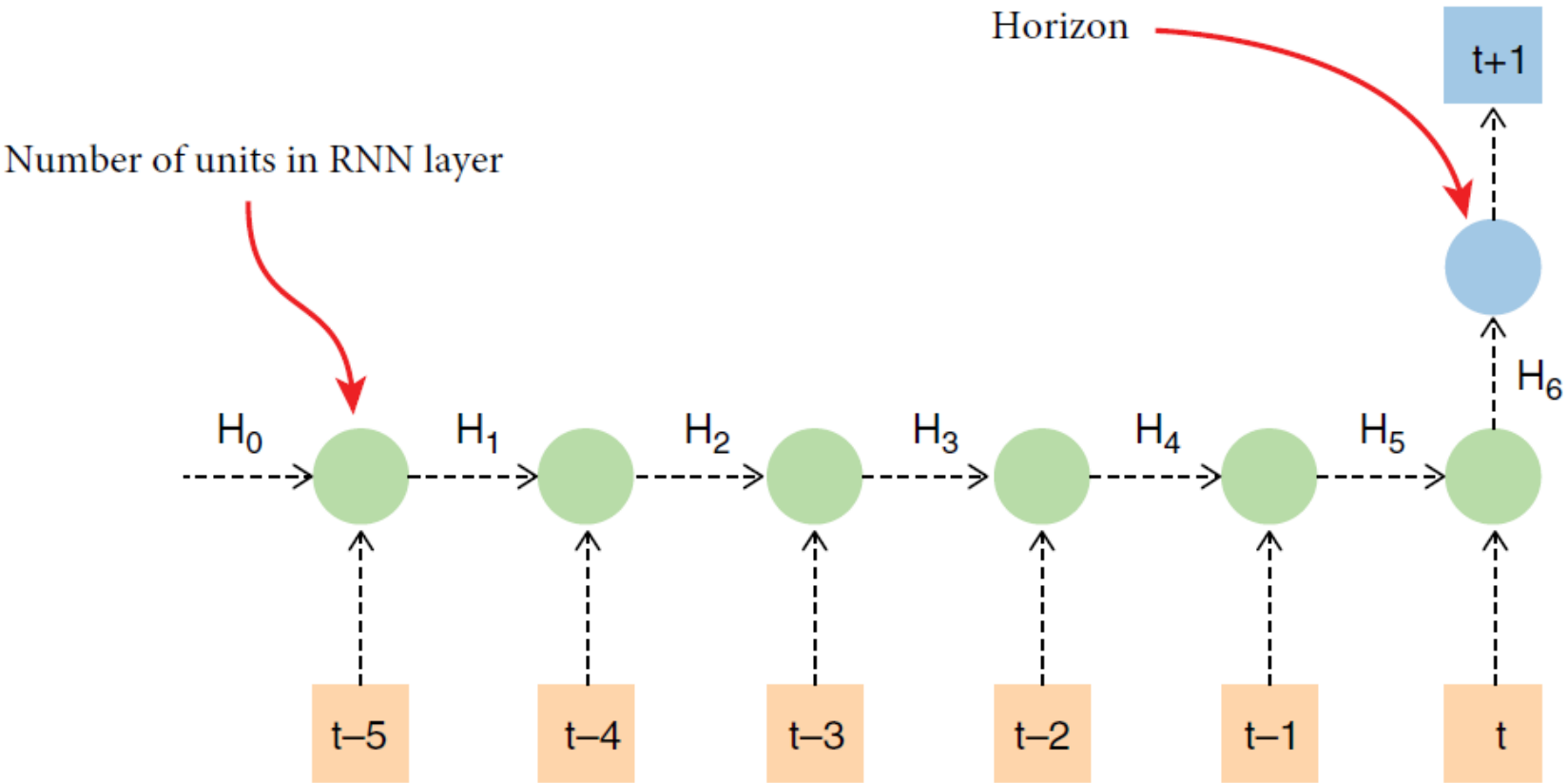
- Keras: API que admite redes neuronales como RNN, CNN y otras arquitecturas.
- TensorFlow: libreria de código abierto para muchas tareas en general de Machine Learning.

Construcción del modelo

- Construiremos nuestro modelo como **univariable**: el único input que usa es el mismo tipo de dato de output, es decir, la carga energética (load).
- Esto nos permite construir conceptualmente un modelo básico e intuitivo. Usaremos una estructura **GRU** (Gated Recurrent Unit).
- El tamaño de la secuencia del input en términos rezagados será 6: la entrada para cada muestra es un vector de las 6 horas previas de los valores de energía.
- El horizonte de predicción es 1: el output es una predicción del consumo de energía en la hora siguiente.

Construcción del modelo

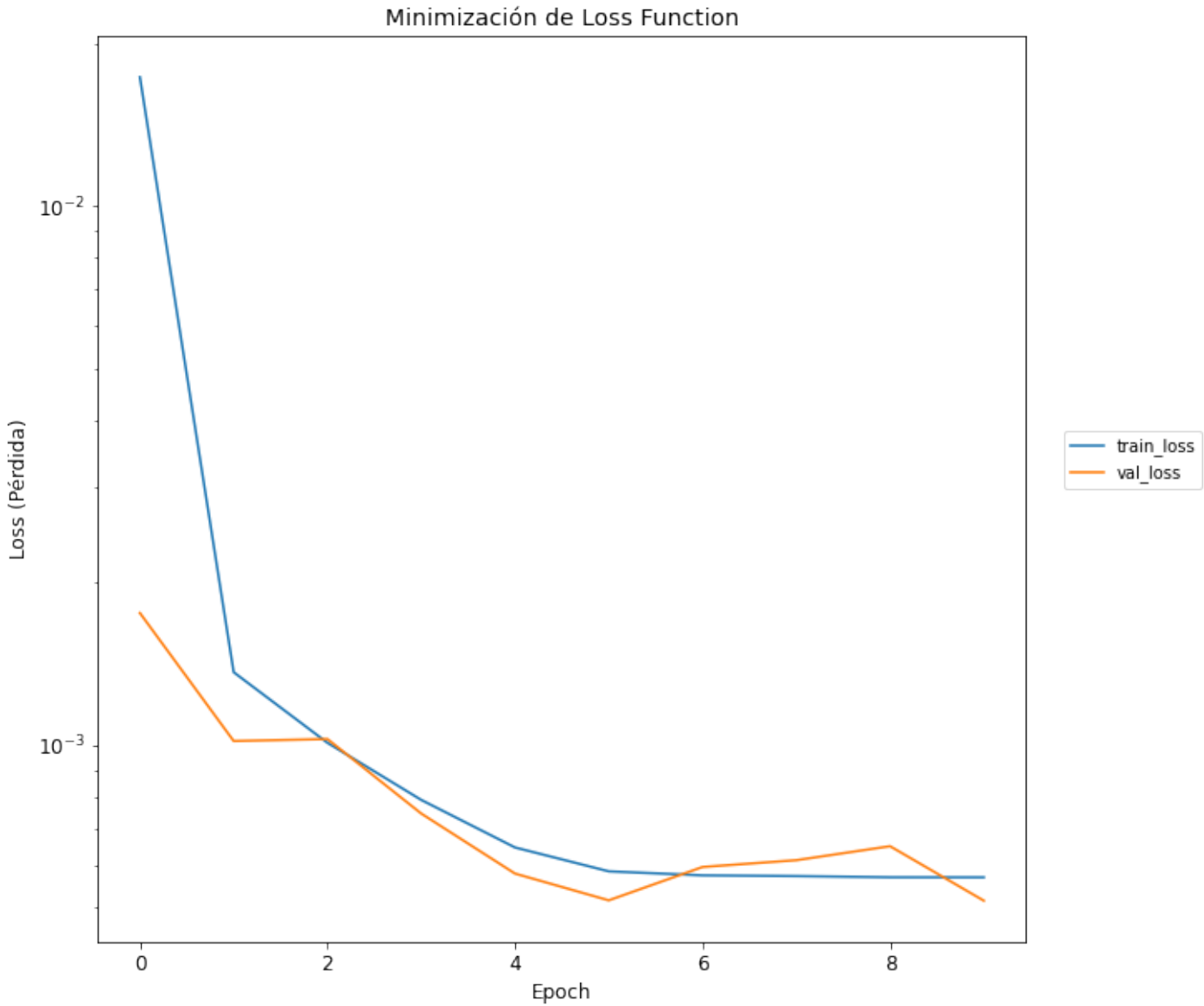
El modelo a utilizar tiene la siguiente arquitectura:



Entrenamiento

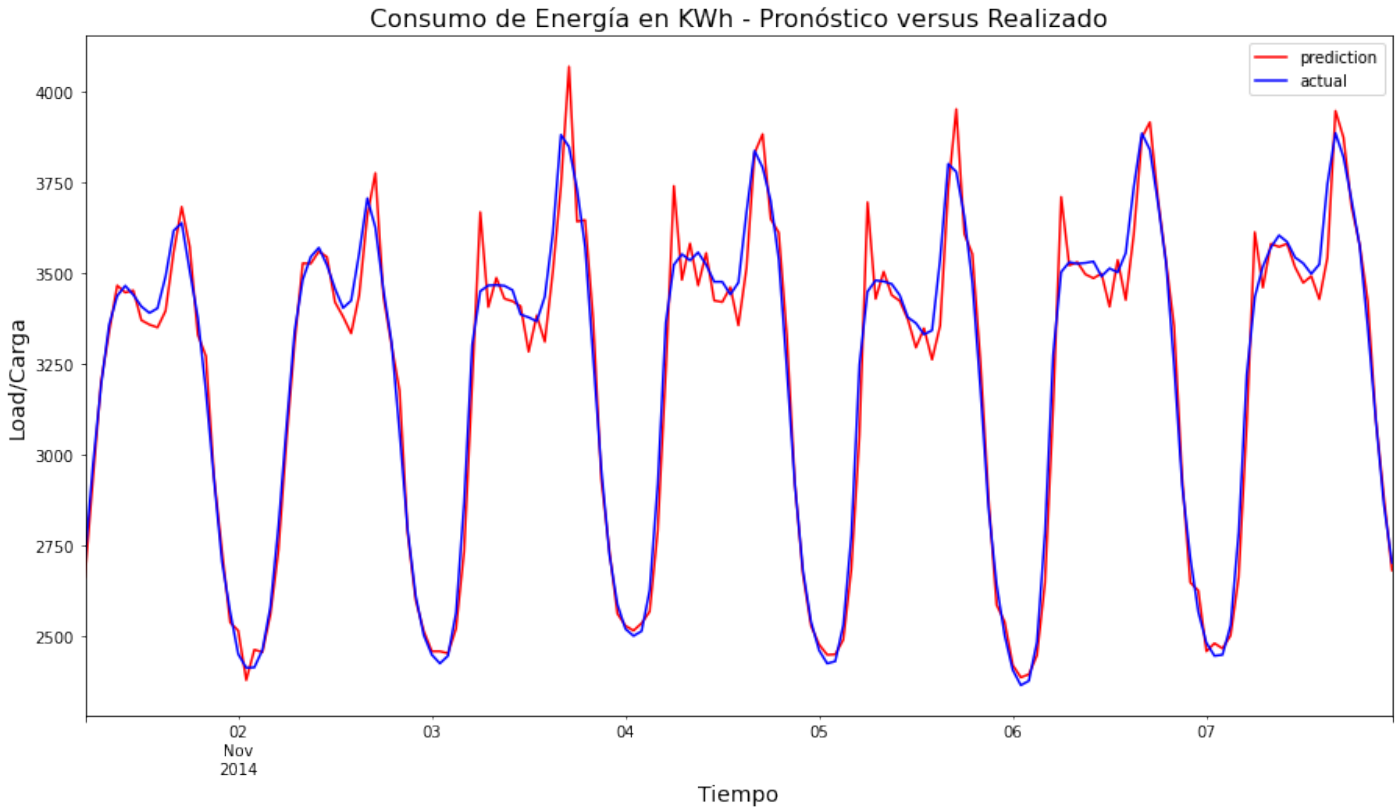
- El entrenamiento consta de minimizar la función de pérdida, que será un **Mean Squared Error**.
- El algoritmo de optimización es **Root Mean Square Propagation** (RMSprop). Emplea un promedio móvil de gradientes cuadrados para normalizar el gradiente. Esta normalización equilibra el tamaño del paso (momentum), disminuyendo el paso para gradientes grandes para evitar explosiones y aumentando el paso para gradientes pequeños para evitar desaparecer.

Entrenamiento



Resultados

- Se evalúa el modelo en el test set de datos.
- La métrica de error para medir la diferencia entre lo predicho y lo realizado será el **MAPE** (Mean Absolute Percentage Error). Resultado: 0.0161



Índice

Energy Consumption Forecast with ARIMA

- Nuestra primera aplicación fue del pronóstico del consumo energético de corto plazo, con muchos datos, empleando RNN.
- Nuestro segundo objetivo es realizar un pronóstico de largo plazo con una técnica estadística conocida como ARIMA (Autoregressive Integrated Moving Average), para contextualizar el caso cuando no se disponen de muchos datos.

Base de datos

- Utilizaremos los datos disponibles de la Secretaría de Energía de la República Argentina, específicamente de los datos de la Compañía Administradora del Mercado Mayorista Eléctrico S.A. (CAMMESA).
- Este dataset posee el consumo energético (o demanda) desde el año 2012 hasta febrero de 2020 (pre COVID-19), desglosado por categoría de usuario y región del país.
- Para nuestro modelo obtendremos 98 observaciones en total para un tipo de usuario, región y tarifa específica. 98 samples es exactamente la cantidad de meses desde enero 2012 hasta febrero 2020. Una cantidad mínima relativa al dataset utilizado de la primer aplicación.

Librerias empleadas de Python

Para esta aplicación haremos uso de las siguientes librerias:

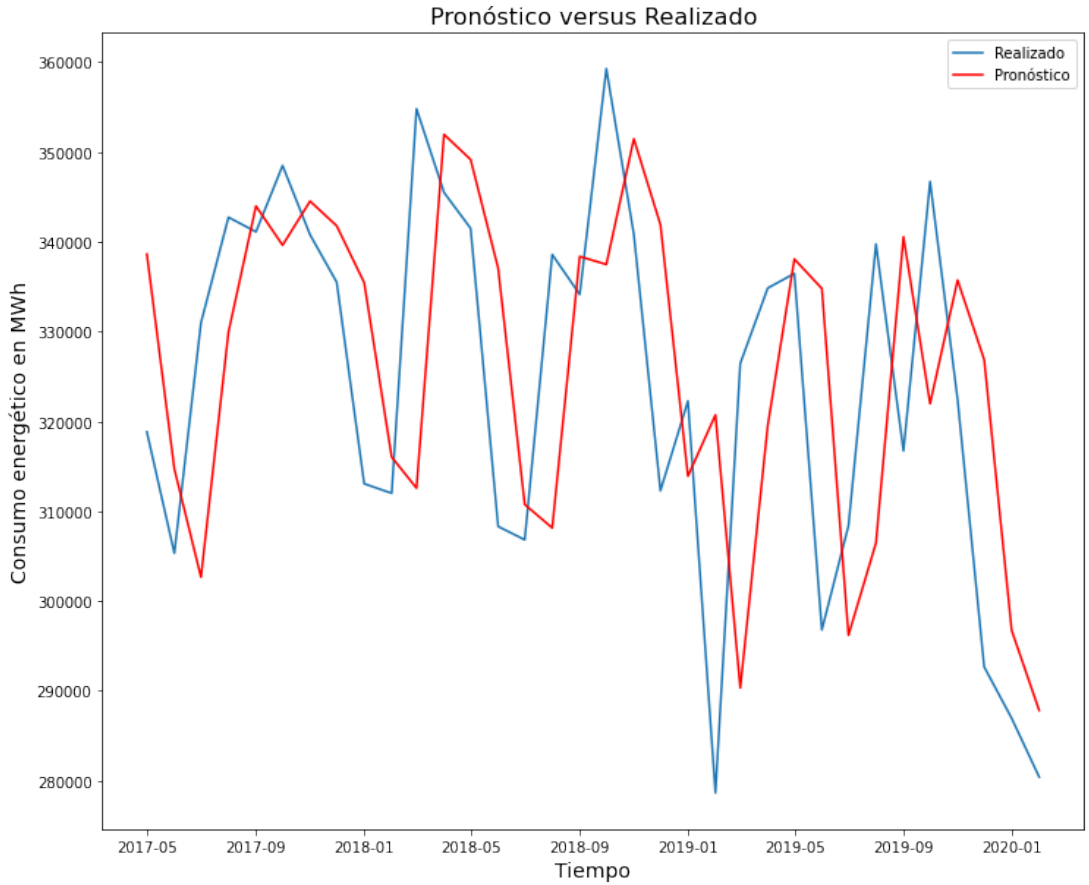
- NumPy: computación científica.
- Matplotlib: visualización de datos.
- Pandas: procesamiento y análisis de datos.
- Statsmodels: herramientas de estadística aplicada.
- Scikit learn: computación científica y estadística aplicada.

Construcción del modelo

- Un modelo de promedio móvil integrado autorregresivo (ARIMA) es una forma de análisis de regresión que mide la fuerza de una variable dependiente en relación con otras variables cambiantes. Es una generalización del modelo ARMA.
- Los parámetros del modelo son p, d, q:
  - p: orden de los rezagos (orden de la autoregresión).
  - d: grado de diferenciación.
  - q: tamaño u orden de la media móvil.

Resultados

- Se computan las predicciones en el test set y se comparan con los valores realizados.
- La métrica de evaluación es el Root Mean Squared Error (RMSE).



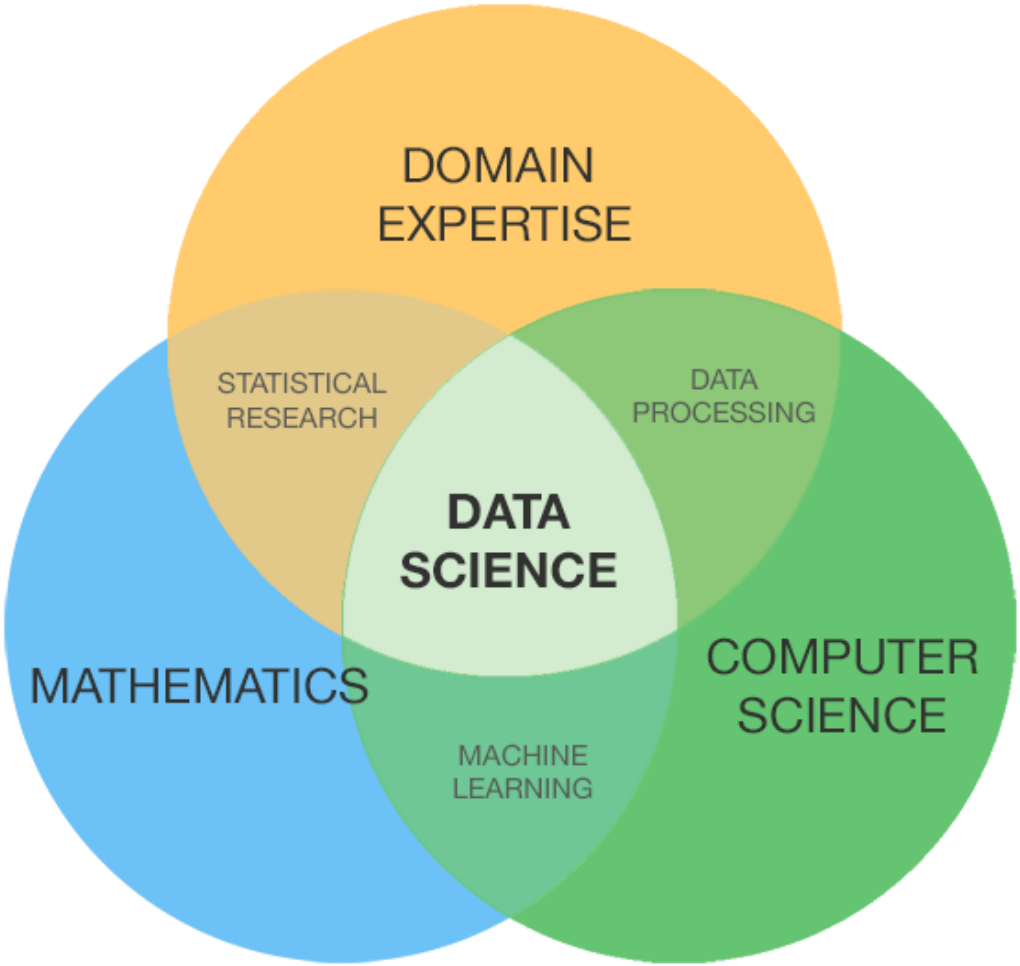
Índice

Conclusiones

- No se pueden comparar las aplicaciones entre sí, porque difieren en la implementación (dataset) y son técnicas diferentes.
- Siempre será ideal disponer de más datos y poder así aplicar modelos *data hungry* como las redes neuronales, que soportan más el ruido, la falta de estacionariedad en una serie, etc.
- Es difícil de todas maneras poder ampliar un dataset de time series con interpolación lineal, computando medias, etc. ya que sesgará nuestros resultados (ampliar datasets de Computer Vision u otras áreas es más fácil).
- Ninguna implementación de Machine Learning será del todo eficaz si no se cuenta con business expertise del problema a resolver: tenemos que entender el negocio.
- Nuestras implementaciones "de juguete" mostraron resultados interesantes.

Conclusiones

Vale la pena destacar que el mundo de la ciencia de datos requiere de los siguientes conocimientos o skills:



Source: Palmer, Shelly. Data Science for the C-Suite. New York: Digital Living Press, 2015. Print.