Traballo Fin de Grao

Predición do tempo con Machine Learning

Sofía Sánchez-Brunete Facal

2022/2023

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Predición do tempo con Machine Learning

Sofía Sánchez-Brunete Facal

Xullo, 2023

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Traballo proposto

Área de Coñecemento: Matemática Aplicada
Título: Predición do tempo con Machine Learning
Breve descrición do contido
O obxectivo do traballo é desenvolver unha ferramenta para a predición meteorolóxica utilizando técnicas do Machine Learning. Para elo estudiaranse as diferentes alternativas de aprendizaxe supervisado e non supervisado, tanto desde un punto de vista teórico como práctico, analizando as vantaxes e inconvenientes de cada unha delas. Por último, utilizarase como linguaxe de programación Matlab ou Python e se validará a ferramenta estudiando un problema relacionado ca enerxía eólica.

Índice

Resumo

Neste traballo búscase construír un modelo de Machine Learning capaz de realizar predicións meteorolóxicas. Para isto, introducirase o funcionamento dos algoritmos de Machine Learning e, en particular, das Redes Neuronais. Exporanse tamén as ferramentas matemáticas detrás do funcionamento destes algoritmos.
Posto que os modelos de Machine Learning traballan a partir dun conxunto de información dada, no Capítulo 1 escollerase e analizarase o conxunto de datos que será empregado posteriormente para adestrar o modelo que queremos construír.
No Capítulo 2, introduciremos os algoritmos de Machine Learning e as redes neuronais, centrándonos na base matemática que os sostén.
No Capítulo 3, a partir dos datos expostos no Capítulo 1 e da base teórica proporcionada no Capítulo 2, construirase un modelo capaz de predicir a velocidade do vento, empregando redes neuronais, e estudarase a súa efectividade.

Abstract

This paper aims to build a Machine Learning model capable of performing weather forecasting. For this purpose, we introduce the operation of Machine Learning algorithms and, in particular, Neural Networks. There will be also exposed the mathematical tools behind this algorithms. Due to Machine Learning models work from a given set of information, the dataset that will lately train the model we want to build, will be chosen and analized in Chapter 1.
In Chapter 2, we will inroduce Machine Learning algorithms and neural networks, focusing on the mathematical foundations behind them.
In Chapter 3, from the dataset exposed in Chapter 1 and the teoric foundations provided in Chapter 2, we will build a model capable to predict wind speed, using neural networks, and we will study its accuracy.

Introdución

O Machine Learning, ou Aprendizaxe Automática, é un subcampo da ciencia computacional que nace do estudo de patróns de recoñecemento e da teoría de aprendizaxe computacional na intelixencia artificial. Este consiste na construción e o estudo de algoritmos que poden analizar grandes cantidades de datos, identificar patróns neles e facer predicións a partir dos mesmos. A popularidade destes algoritmos aumentou notablemente nos últimos anos debido a súa ampla utilidade en diversas áreas e a súa eficacia á hora de resolver problemas complexos.

Dentro do Machine Learning, destacan pola súa eficacia as Redes Neuronais. Estas consisten en algoritmos que se inspiran no funcionamento do cerebro humano e son capaces de procesar grandes cantidades de datos e atopar patróns neles. As redes neuronais están compostas de unidades de procesamento coñecidas como nodos ou neuronas que están conectadas entre si. Ditas neuronas organízanse en distintas capas.

Os algoritmos de Machine Learning en xeral, e as redes neuronais en particular, constrúen as normas e regras nas que se basean os modelos a partir do estudo de conxuntos de datos dos que se coñece a información recibida e a información de saída esperada. Para lograr a obtención correcta destas regras as redes neuronais baseanse en algoritmos como o método do gradiente e a diferenciación automática.

Dado que unha das principais funcións das redes neuronais é a predición de eventos futuros, unha posible aplicación destas é a predición meteorolóxica. Esta está a cobrar unha gran importancia nestes tempos; xa sexa no ámbito empresarial, en sectores como a agricultura, a produción enerxética ou os transportes, como para previr condicións climatolóxicas extremas ou para a toma de decisións na vida cotiá. A gran cantidade de parámetros que inflúen na predición das condicións climáticas e o comportamento non linear destes fan que o emprego de redes neuronais para a resolución deste tipo de problemas cobre unha gran interese.

De cara á produción enerxética, a utilización destes algoritmos para a predición de variables meteorolóxicas, como a velocidade do vento, pode ser de axuda en tarefas como a de previr a cantidade de enerxía xerada por unha central eólica.

A enerxía eólica é un tipo de enerxía renovable que consiste na xeración da electricidade a partir do vento, sendo a velocidade deste un dos factores máis relevantes. Coñecer con certeza

a cantidade de vento dun futuro relativamente próximo axudaranos a predicir a cantidade de enerxía que este chegará a producir, o que resultará clave para optimizar o funcionamento dos parques eólicos.

Neste traballo búscase construír un modelo que sexa capaz de realizar predicións meteorolóxicas, en particular, da velocidade do vento. Para isto requiriirase dun conxunto de datos a partir do cal dito modelo sexa capaz de extraer certos patróns para construír as regras de predición.

Capítulo 1

Introdución ao problema de predición do tempo

A finalidade deste capítulo é presentar un problema de Machine Learning enfocado á predición meteorolóxica, en particular, da velocidade do vento. Nel introducirase un conxunto de datos que será empregado polo modelo para facer predicións.
As observacións deste conxunto están dispoñibles en [6].

1.1. Conxunto de datos

O conxunto de datos que empregaremos está formado por observacións climáticas proporcionadas pola oficina de meteoroloxía do goberno de Australia. Estes datos foron recollidos diariamente por distintas estacións meteorolóxicas situadas en diversas cidades australianas ao longo de preto de 10 anos. Neste caso só serán empregadas as observacións realizadas na cidade de Canberra, que supoñen un total de 3436 entradas, tomadas entre novembro do 2007 e xullo de 2017. Así, traballaremos cun conxunto de datos cuxas primeiras entradas son as que se mostran nos cadros 1.1 e 1.2.

Date	MinTemp	MaxTemp	Rainfall	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm
02/11/07	14.0	26.9	3.6	E	W	4.0	17.0
03/11/07	13.7	23.4	3.6	N	NNE	6.0	6.0
04/11/07	13.3	15.5	39.8	WNW	W	30.0	24.0
05/11/07	7.6	16.1	2.8	SSE	ESE	20.0	28.0
06/11/07	6.2	16.9	0.0	SE	E	20.0	24.0

Cadro 1.1: Primeiras entradas do conxunto de datos

Date	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Temp9am	Temp3pm
02/11/07	80.0	36.0	1012.4	1008.4	17.5	25.7
03/11/07	82.0	69.0	1009.5	1007.2	15.4	20.2
04/11/07	62.0	56.0	1005.5	1007.0	13.5	14.1
05/11/07	68.0	49.0	1018.3	1018.5	11.1	15.4
06/11/07	70.0	57.0	1023.8	1021.7	10.9	14.8

Cadro 1.2: Resto das entradas do conxunto de datos

No cadro 1.1 recóllense:

- MinTemp, MaxTemp: a temperatura mínima e máxima, medida en graos Celsius.
- Rainfall: precipitacións caídas nas 24 horas anteriores ás 9 da maña do día no que se rexistra, medidas en milímetros.
- WindDir9am, WindDir3pm: a dirección do vento media nos 10 minutos anteriores ás 9:00h e ás 15:00h, respectivamente, medida en quilómetros por hora.
- WindSpeed9am,WindSpeed3pm: a velocidade do vento media nos 10 minutos anteriores ás 9:00h e ás 15:00h, respectivamente, medida en quilómetros por hora.

No cadro 1.2:

- Hummidity9am, Hummidity3pm: a humidade relativa ás 09:00h e ás 15:00h, respectivamente, medida en tantos por cento.
- Pressure 9am, Pressure3pm: a presión atmosférica con respecto ao nivel do mar ás 09:00h e ás 15:00h, respectivamente, medido en hectopascais.
- Temp9am, Temp3pm: a temperatura ás 09:00h e ás 15:00h, respectivamente, medida en graos Celsius.

No cadro 1.1 debémonos fixar en que as variables que recollen a dirección do vento son non numéricas. Para poder traballar con elas deberemos asignarlles valores numéricos.

Observamos que as distintas categorías nas que se clasifica a dirección do vento son:

```
[3]: df['WindDir9am'].unique()

[3]: array(['E', 'N', 'WNW', 'SSE', 'SE', 'S', 'WSW', 'SW', 'NNE', 'NNW',
            'ENE', 'SSW', 'NW', 'ESE', 'NE', 'W', nan], dtype=object)
```

É dicir, as direccións representadas na figura 1.1, ademais dos datos non recollidos ('nan').

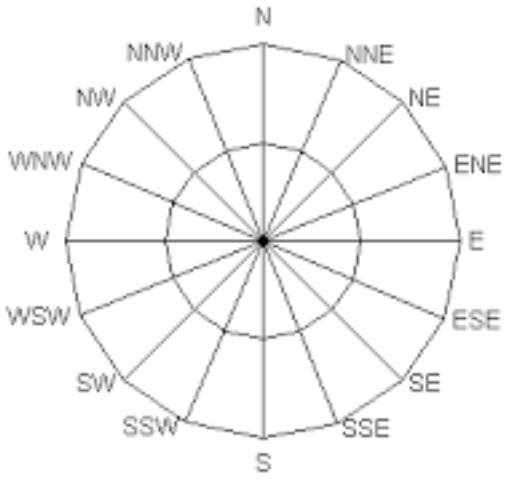

Figura 1.1: Representación das direccións do vento. Ver [9]

.

Tras asignar un valor numérico a cada dirección, e substituir os valores nulos por cero, podemos observar a evolución das distintas variables ao longo do tempo como se pode observar nas figuras 1.3, 1.2 e 1.4.

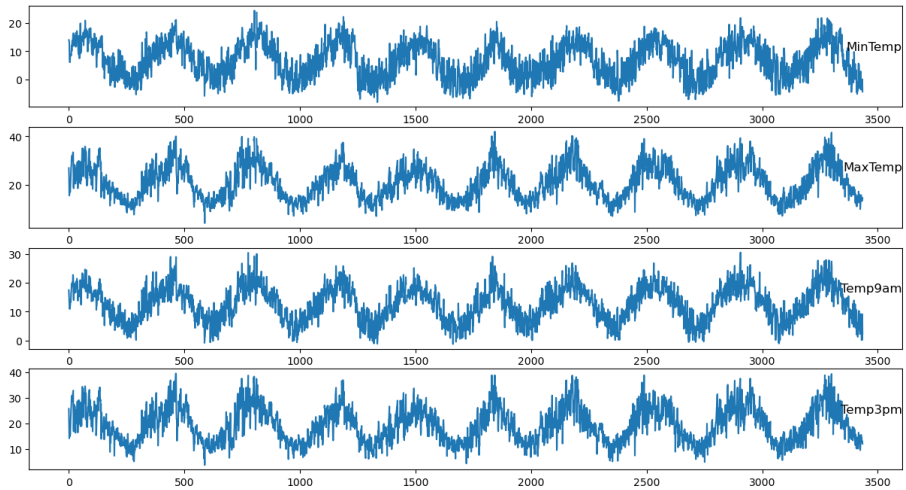

Figura 1.2: Evolución da temperatura

A figura 1.2 representa a evolución das variables relacionadas coa temperatura. En todas elas apreciamos un claro comportamento periódico, consecuencia da estacionalidade da mesma.

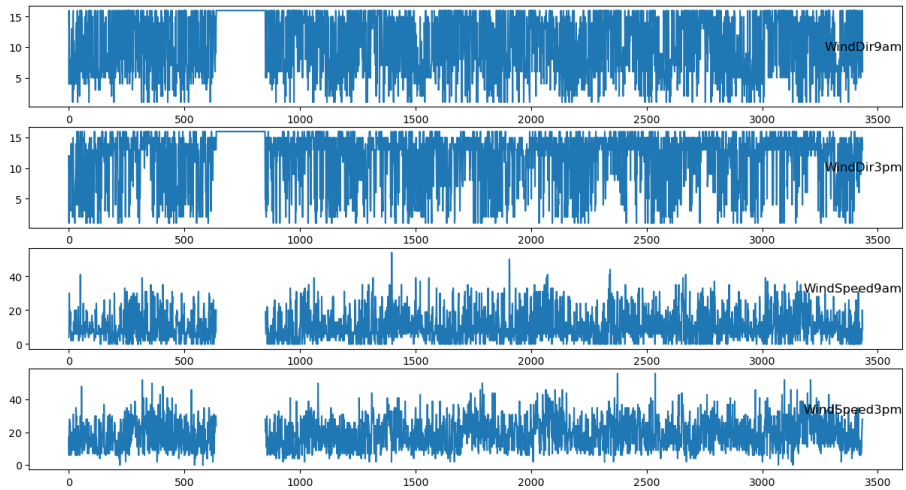

Figura 1.3: Evolución da velocidade e da dirección do vento

A figura 1.3 móstranos como as variables relacionadas co vento presentan un comportamento anómalo durante un período considerable de tempo. O mesmo observamos na figura 1.4 con respecto as variables relacionadas ca presión. É razoable pensar que isto é debido a ausencia de datos. Para evitar que isto afecte á predición, eliminaremos as filas que teñan polo menos tres valores nulos.

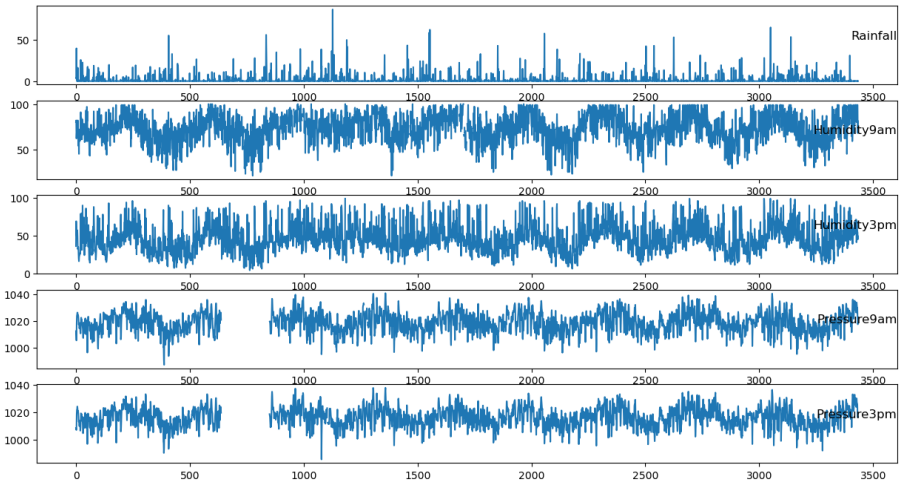

Figura 1.4: Evolución das precipitacións, da humidade e da presión

Ademais, na figura 1.4 móstrase a evolución das variables relacionadas ca presión, a humidade e a cantidade de chuvia. Nela observamos tamén un comportamento estacional ainda que moito menos explícito que na figura 1.2.

Podemos observar tamén a posible correlación entre as distintas variables na figura 1.5.

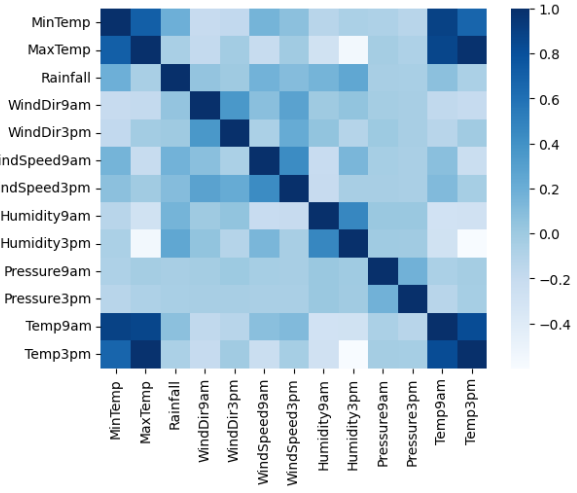

Figura 1.5: Correlación das distintas variables

A gráfica anterior é un mapa de calor que nos mostra como de relacionadas están as distintas variables entre si. Xunto co mapa, está representada a escala coa que se mide dita correlación. Se esta toma valores positivos, entre o 0 e o 1, o aumento dunha das variables implicará o aumento da outra. Pola contra, se os valores son negativos, o aumento dunha variable implicará

a diminución daquela coa que se relaciona.
Sabemos ademais que, canto maior sexa o módulo da correlación, maior será o aumento ou a diminución dunha variable en función do aumento doutra. Na nosa gráfica observamos fortes correlacións en variables relacionadas entre si. Vemos tamén que, en xeral, existe certa relación entre o resto das variables. Deste xeito, parece lóxico concluir que sexan necesarias o resto das variables para facer un estudo das variables relacionadas co vento. Antes de continuar coa preparación dos datos e a elección dun modelo, exporase unha base teórica do funcionamento do Machine Learning en xeral e das redes neuronais en particular.

Capítulo 2

Machine Learning

Neste capítulo estudarase a base matemática dos algoritmos de Machine Learning e, en particular, das redes neuronais. Definirase a función de custo e explicarase como minimizala empregando o método do gradiente. Por último, introducirase o algoritmo de diferenciación automática de cara a calcular o gradiente da función de custo dunha rede neuronal.
Para facelo seguiremos o segundo e o terceiro capítulo de [1], o primeiro capítulo de [3], o segundo capítulo de [2] e o artigo [4]. Ademais, durante este capítulo exporanse algúns resultados de análise matemática extraídos do terceiro capítulo de [5].

2.1. Aprendizaxe dos algoritmos

Os algoritmos que se engloban dentro do Machine Learning constrúen as normas a partir dun conxunto de datos dado.

Definición 2.1. Chamaremos **datos de entrada** ao conxuto de datos ou obxectos aos que terá acceso a máquina, que denotaremos por $\mathcal{X} \subset \mathbb{R}^n$, sendo n a dimensión dos datos.

Definición 2.2. Denotaremos por $\mathcal{Y} \subset \mathbb{R}^m$ ao conxunto de **etiquetas** posibles ou datos de saída, sendo m a dimensión destes.

Dependendo do tipo de aprendizaxe o noso obxectivo será etiquetar os datos de entrada, sendo $\mathcal{Y}$ o conxunto de etiquetas, ou atopar patróns existentes entre eles.
En función de se dispoñemos ou non deste conxunto $\mathcal{Y}$ distinguimos os seguintes tres tipos de aprendizaxe:

1. **Aprendizaxe supervisada**
   Na aprendizaxe supervisada contamos con pares de datos $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$, aos que nos

referiremos como pares etiquetados.
O obxectivo desta aprendizaxe será que, a partir dun conxunto de pares de datos, que chamaremos datos de adestramento, o ordenador sexa capaz de establecer unha función que leve as entradas nas saídas.

2. **Aprendizaxe non supervisada**
Na aprendizaxe non supervisada non contamos con estes pares etiquetados. O algoritmo recibirá unicamente un conxunto de datos de entrada $\mathbf{x}_i \in \mathcal{X}$, que tratará como un conxunto de variables aleatorias. O seu obxectivo será o de analizar os datos de entrada e atopar patróns ou relacións entre eles.

3. **Aprendizaxe semi-supervisada**
Esta aprendizaxe contará a súa vez con datos de adestramento etiquetados e non etiquetados. Así, este tipo de aprendizaxe resulta unha mestura das dúas anteriores, tratando fundamentalmente con datos sen etiquetar. O seu funcionamento consistirá nun primeiro lugar en agrupar os datos de entrada (aprendizaxe non supervisada) e a partir desta agrupación poder establecer as saídas desexadas (aprendizaxe supervisada).

A continuación, centrarémonos na aprendizaxe supervisada, é dicir, naqueles algoritmos que traballan con conxuntos de datos etiquetados.

Notación 2.3. Por comodidade denotaremos ao conxunto de datos etiquetados como $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$.

Observación 2.4. Imos supoñer de momento, sen perda de xeneralidade, que tanto os datos de entrada como as etiquetas, son unidimensionais.

Definición 2.5. Sexan $n, m \in \mathbb{N}$, con $n < m$. Dado o subconxunto finito de pares etiquetados $\mathcal{Z} = ((x_1, y_1), ..., (x_m, y_m))$, chamamos **datos de adestramento** ao subconxunto $E = ((x_1, y_1), ..., (x_n, y_n))$ de $\mathcal{Z}$. A partir deste conxunto o modelo será capaz de definir as funcións e relacións desexadas.

Este conxunto de datos empregarase na que será a primeira etapa da aprendizaxe, a **fase de adestramento**. Nela, a partir do conxunto E, a máquina deberá atopar patróns e definir funcións que relacionan cada dato de entrada coa etiqueta correspondente.

Notación 2.6. Denotamos por $h : \mathcal{X} \longrightarrow \mathcal{Y}$ ás funcións que lle estamos a pedir á máquina. Denotaremos por $\mathcal{H}$ ao conxunto de funcións de predición. Así, para cada $x \in \mathcal{X}$, $h(x)$ será a predición realizada pola función $h \in \mathcal{H}$.

Unha vez finalizada esta etapa, para avaliar os algoritmos adestrados, introducimos o seguinte conxunto de datos:

Definición 2.7. Dado o conxunto finito de pares etiquetados $\mathcal{Z} = ((x_1, y_1), ..., (x_m, y_m))$ para un $m \in \mathbb{N}$, chamamos **datos de proba**, ou de validación, ao subconxunto $T \subset \mathcal{Z}$, tal que $E \cap T = \emptyset$ e $E \cup T = \mathcal{Z}$.
É dicir, o conxunto de datos de proba é un conxunto de datos, dentro dos datos etiquetados, que é totalmente distinto ao conxunto de datos de adestramento. Por unha cuestión de eficiencia, entre o conxunto de adestramento e o conxunto de proba, atoparanse todos os pares de datos etiquetados.

Nesta fase, á que nos referiremos como **fase de proba**, tan só se lle proporcionará á máquina un conxunto de datos sen etiquetar. O traballo a realizar consistirá, entón, en etiquetar estes datos a partir das funcións e relacións establecidas na etapa de adestramento e en comprobar se estas se axustan ás etiquetas dadas.

2.1.1. Función de custo

Para avaliar a aprendizaxe destes algoritmos empregaremos un tipo de funcións coñecidas como funcións de custo.

Definición 2.8. Dado o conxunto de funcións de predición $\mathcal{H}$ e sexa $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ o conxunto de datos etiquetados, definimos a función de perda como unha función da forma:

$$\ell : \mathcal{H} \times \mathcal{Z} \longrightarrow \mathbb{R}^+.$$

Sexa $x \in \mathcal{X}$ un dato de entrada calquera e sexa $y \in \mathcal{Y}$ a súa etiqueta correspondente, dada $h(x) \in \mathcal{Y}$ a predición realizada por unha función $h \in \mathcal{H}$, a función de perda ℓ mide a distancia entre $h(x)$ e y.

Existen múltiples funcións de perda, porén neste traballo centrarémonos na función de perda cuadrática.

Definición 2.9. Sexa ℓ unha función de perda, diremos que é de perda cuadrática cando é da forma:

$$\ell_{sq}(h, (x, y)) := \frac{1}{2}(h(x) - y)^2.$$

Observación 2.10. Cabe destacar, que podemos definir a función de perda deste xeito porque estabamos a asumir na observación 2.4, $\mathcal{X}, \mathbf{y} \subset \mathbb{R}$. No caso de que $\mathcal{Y} \subset \mathbb{R}^n$, con $n > 1$, definimos a función de perda cuadrática como:

$$\ell_{sq}(h, (\mathbf{x}, \mathbf{y})) := \frac{1}{2} \parallel h(\mathbf{x}) - \mathbf{y} \parallel^2 .$$

Notación 2.11. Denotamos por $\mathcal{G}$ a distribución de probabilidade sobre $\mathcal{X}$. É dicir, estamos a asumir que os $\mathbf{x}_i$ están xerados por algunha distribución de probabilidade, que será descoñecida para a máquina.

A continuación definiremos os conceptos de función de custo e función de custo empírica, para posteriormente introducir a función de custo cuadrática.

Definición 2.12. A función de custo será a esperanza da función de perda para unha función $h \in \mathcal{H}$ con respecto a probabilidade $\mathcal{G}$, é dicir:

$$\mathcal{C}_{\mathcal{G}}(h) := \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{G}}[\ell(h,(\mathbf{x},\mathbf{y}))],$$

sendo $(\mathbf{x},\mathbf{y}) \in \mathcal{Z}$.

Definición 2.13. A función de custo empírico será a esperanza de ℓ sobre un conxunto dado $S = ((\mathbf{x}_1,\mathbf{y}_1),...,(\mathbf{x}_l,\mathbf{y}_l)) \subset \mathcal{Z}$:

$$\mathcal{C}_S(h) := \frac{1}{l}\sum_{i=1}^{l} \ell(h,(\mathbf{x}_i,\mathbf{y}_i)),$$

sendo $l \in \{1,..,m\}$.

Definición 2.14. Definimos, en consecuencia, a función de custo cuadrática como:

$$\mathcal{C}_{sq}(h) = \frac{1}{2m}\sum_{i=1}^{m} \parallel h(\mathbf{x}_i) - \mathbf{y}_i \parallel^2 .$$

A función de custo é unha función que compara o valor obtido co desexado, de forma que mide o nivel de adestramento do algoritmo. Esta función é non negativa e aproxímase a $\mathbf{0}$ cando $h(\mathbf{x})$ se aproxima a $\mathbf{y}$. Así, a maior custo peores serán as nosas predicións. O obxectivo do noso adestramento será minimizar a función de custo, en particular, traballaremos coa función de custo cuadrática.
O problema que buscamos resolver será:

$$(P)\begin{cases} \textit{Atopar un } h^* \in \mathcal{H} \textit{ tal que} \\ \mathcal{C}_{\mathcal{G}}(h^*) = \min_{h\in\mathcal{H}} \mathcal{C}_{\mathcal{G}}(h). \end{cases}$$

Para explicar os resultados que nos permitirán resolver o problema (P), consideraremos primeiro unha función de custo diferenciable do tipo:

$$\begin{array}{rccc} C: & D \subset \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ & \mathbf{x} & \longrightarrow & C(\mathbf{x}), \end{array}$$

onde D é un aberto de $\mathbb{R}^n$. Buscamos agora resolver o seguinte problema:

$$(P')\begin{cases} \textit{Atopar un } \mathbf{x}^* \in \mathcal{D} \textit{ tal que} \\ C(\mathbf{x}^*) = \min_{\mathbf{x}\in\mathcal{D}} C(\mathbf{x}). \end{cases}$$

2.1.2. Método do gradiente

Para resolver o problema (P') empregarase un método de descenso coñecido como **método do gradiente**.
Os métodos de descenso son métodos iterativos que, partindo de $\mathbf{x}_0 \in D$ dado, xeran unha sucesión de aproximacións da solución:

$$\{\mathbf{x}_k\}_{k\geq 0},$$

tal que

$$C(\mathbf{x}_k) > C(\mathbf{x}_{k+1}), \tag{2.1}$$

para $k \geq 0$. Búscase que esta sucesión sexa converxente a un $\overline{\mathbf{x}} \in D$, de xeito que $\mathbf{x}^* = \overline{\mathbf{x}}$.
Podemos escribir o termo $\mathbf{x}_{k+1}$, con $k \geq 0$, como:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta_k,$$

sendo Δ_k un vector, ao que nos referiremos como incremento, e que nos permitirá actualizar cada iterante. Este vén definido como:

$$\Delta_k = \alpha_k \mathbf{d}_k,$$

sendo $\mathbf{d}_k$ un vector dirección e α_k un escalar que modula canto se avanza na dirección do vector. Dado que buscamos unha sucesión tal que C decreza, buscamos que o vector $\mathbf{d}_k$ indique unha dirección de descenso. En particular, introduciremos o método do gradiente, que traballa ca dirección do máximo descenso.
A continuación, introducimos algúns conceptos de análise que serán necesarios para calcular dita dirección e que, en xeral, facilitarán a comprensión do método.

Definición 2.15 (Derivada direccional)**.** Sexa D un aberto de $\mathbb{R}^n$, supoñamos C unha función diferenciable en D. Sexa $\mathbf{x} \in D$ e $\mathbf{v} \in \mathbb{R}^n$ un vector unitario, definimos a derivada direccional de C en $\mathbf{x}$ ao longo de $\mathbf{v}$ como:

$$\frac{\partial C(\mathbf{x})}{\partial \mathbf{v}} := \lim_{h\to 0} \frac{C(\mathbf{x}+h\mathbf{v}) - C(\mathbf{x})}{h},$$

sendo $h \in \mathbb{R}$.

Definición 2.16 (Derivada parcial)**.** Sexa $i \in \{1, ..., n\}$, definimos a derivada parcial respecto da variable $\mathbf{v}_i$ como a derivada direccional na dirección do vector da base canónica $\mathbf{v} = \mathbf{e}_i = (0, ..., 1, ..., 0)$. Denotarémola como $D_i C(\mathbf{x})$.

Definición 2.17 (Gradiente)**.** Sexa D un aberto de $\mathbb{R}^n$ e $C : D \to \mathbb{R}$ unha función con valores reais. Definimos o gradiente dun campo escalar C nun punto $\mathbf{x}$ como un campo vectorial da

forma:

$$\begin{array}{rccl} \nabla C: & D & \longrightarrow & \mathbb{R}^n \\ & \mathbf{x} & \longmapsto & \nabla C(\mathbf{x}) = (D_1 C(\mathbf{x}), ..., D_n C(\mathbf{x}))^t . \end{array}$$

Observación 2.18. En base ás definicións anteriores (2.15), (2.16),(2.17), observamos a seguinte relación entre as derivadas direccionais e o gradiente:

$$\frac{\partial C}{\partial \mathbf{u}}(\mathbf{x}) = \nabla C(\mathbf{x})^t \cdot \mathbf{u}.$$

A continuación, probarase unha interpretación xeométrica do gradiente que fai que este nos sirva para calcular a dirección de máximo incremento dunha función nun punto dado.

Proposición 2.19. *Dada C unha función diferenciable en $\boldsymbol{x}$ con $\nabla C(\boldsymbol{x}) \neq \boldsymbol{0}$. O gradiente determina a dirección de máximo incremento da función no punto.*

Demostración. Consideramos $\mathbf{v}$ un vector unitario. Dada C unha función diferenciable con diferencial distinta de cero. Vimos en 2.18 que a derivada direccional en $\mathbf{x}$ na dirección de $\mathbf{v}$ é:

$$\frac{\partial C}{\partial \mathbf{v}}(\mathbf{x}) = \nabla C(\mathbf{x})^t \cdot \mathbf{v} = \parallel \nabla f(\mathbf{x}) \parallel \parallel \mathbf{v} \parallel cos\gamma,$$

sendo γ o ángulo entre $\nabla C(\mathbf{x})$ e $\mathbf{v}$, e $\parallel \mathbf{v} \parallel = 1$.
Observamos entón, que a derivada direccional toma o seu máximo valor cando $cos\alpha = 1$, é dicir, cando o vector $\mathbf{v}$ e o gradiente de $C(\mathbf{x})$ teñen a mesma dirección e sentido. Así, por ser $\mathbf{v}$ unitario,

$$\mathbf{v} = \frac{\nabla C(\mathbf{x})}{\parallel \nabla C(\mathbf{x}) \parallel},$$

será o vector que maximiza a derivada direccional. É dicir, $\nabla C(\mathbf{x})$ daranos a dirección de máximo incremento de C en $\mathbf{x}$. □

Observación 2.20. Acabamos de probar que o vector $\nabla C(\mathbf{x})$ nos indicará a dirección de máximo crecimento de C no punto $\mathbf{x}$. Para obter a dirección de máximo descenso de C no punto $\mathbf{x}$ simplemente debemos considerar o vector $-\nabla C(\mathbf{x})$.

Polo tanto, se relacionamos estes conceptos co noso caso particular, para un punto $\mathbf{x}_k$, con $k \geq 0$, resulta natural definir a dirección de máximo descenso como $\mathbf{d}_k = -\nabla C(\mathbf{x}_k)$.
Unha vez elexido $\mathbf{d}_k$ buscamos un parámetro α_k axeitado.

Definición 2.21. Definimos o **paso** α_k, para $k \geq 0$, como un parámetro positivo, tomando valores en $[0, 1]$. Este parámetro serve para escalar o gradiente.

Observación 2.22. O valor do paso toma unha gran importancia no funcionamento do proceso. De escoller un valor demasiado alto provocará movementos oscilatorios que poderían levar a que se acabase obtendo un incremento da función de custo positivo, $\Delta C > 0$. Porén, tomar un α_k demasiado pequeno pode levar a unha idea de falsa converxencia. Buscamos polo tanto un punto intermedio, que nos permita chegar ao resultado desexado no menor número de etapas posibles.

Para elixir o paso, supoñemos que a iteración se atopa no punto $\mathbf{x}_k$. Unha vez seleccionada unha dirección de descenso dende $\mathbf{x}_k$, $\mathbf{d}_k$, introducimos a seguinte función real de variable real:

$$\begin{array}{rccl} j: & I \subset \mathbb{R}^+ & \longrightarrow & \mathbb{R}, \\ & \alpha & \longrightarrow & j(\alpha) = C(\mathbf{x}_k + \alpha \mathbf{d}_k). \end{array} \tag{2.2}$$

O conxunto I será un intervalo de $\mathbb{R}$ tal que se $\alpha \in I$, entón $\mathbf{x}_k + \alpha \mathbf{d}_k \in D$. Por comodidade, e sen perda de xeneralidade, podemos considerar $I = [0, \infty)$.

Observación 2.23. Nótese que, ao ser C unha función diferenciable, por construción da función j, esta tamén o será. Por ser j unha función real de variable real, podemos denotar a súa derivada como $j'(\alpha)$. Deste xeito:

$$j'(\alpha) = \frac{\partial j}{\partial \alpha} = \frac{\partial C(\mathbf{x}_k + \alpha \mathbf{d}_k)}{\partial \alpha} = \nabla C(\mathbf{x}_k + \alpha \mathbf{d}_k)^t \cdot \mathbf{d}_k,$$

polo visto na observación 2.18.

Ao introducir os métodos de descenso en (2.1), establecíase

$$C(\mathbf{x}_k) > C(\mathbf{x}_{k+1}).$$

Por como está definida j en (2.2), deducimos que a condición que se debe verificar é:

$$j(\alpha_k) < j(0). \tag{2.3}$$

Polo visto na observación 2.23, para $\alpha = 0$ terase :

$$j'(0) = - \parallel \nabla C(\mathbf{x}_k) \parallel^2 < 0.$$

Isto implicará que vai existir un $\alpha_k > 0$ tal que:

$$j(\alpha_k) < j(0).$$

Como xa dixemos na observación 2.22, ademais de verificar (2.3) o paso α_k non debe ser nin demasiado grande nin demasiado pequeno. Para isto podémonos servir dos coñecidos como métodos de descenso con paso variable, entre os que se atopan a regra de Goldstein e a regra de Wolfe-Powell. En ambos casos, para un $\alpha > 0$, disporémos:

- Criterio lóxico de paso grande, CPG, de xeito que se CPG(α)==verdadeiro, o paso será rexeitado por ser demasiado grande.
- Criterio lóxico de paso pequeno, CPP, de xeito que se CPP(α)==verdadeiro, o paso será rexeitado por ser demasiado pequeno.

Cando CPG(α)==falso e CPP(α)==falso, o paso considerarase admisible.

Regra de Goldstein

A partir dos parámetros $\rho \in (0,1)$ e $\sigma \in (\rho, 1)$, definimos as rectas:

$$r_1(\alpha) = j(0) + \rho j'(0)\alpha,$$

$$r_2(\alpha) = j(0) + \sigma j'(0)\alpha.$$

A regra de Goldstein propón elixir un $\alpha > 0$ tal que:

$$j(\alpha) \leq r_1(\alpha),$$

$$j(\alpha) \geq r_2(\alpha).$$

No caso de $j(\alpha) > r_1(\alpha)$, α considerarase un paso demasiado grande ($CPG(\alpha) == verdadeiro$) e cando $j(\alpha) < r_2(\alpha)$, α considerarase un paso demasiado pequeno ($CPP(\alpha) == verdadeiro$).

Regra de Wolfe-Powell

A partir dos parámetros $\rho \in (0,1)$ e $\sigma \in (\rho, 1)$, consideramos de novo a recta

$$r_1(\alpha) = j(0) + \rho j'(0)\alpha.$$

Neste caso a regra de Wolfe-Powell propón escoller un $\alpha > 0$ que cumpra:

$$j(\alpha) \leq r_1(\alpha),$$

$$j'(\alpha) \geq \sigma j'(0).$$

Polo descrito anteriormente, fixados o número de iteracións ($nitmax \in \mathbb{N}$), $\epsilon > 0$, $\eta > 0$ e $\mathbf{x}_0 \in D$, o algoritmo do método do descenso ten a seguinte estrutura:

Para $k = 0, 1, ..., nitmax$,

- Coñecido $\mathbf{x}_k$ calcúlase $\mathbf{x}_{k+1}$ tal que $C(\mathbf{x}_{k+1}) < C(\mathbf{x}_k)$ como segue:
  1. Calcúlase a dirección de máximo descenso $\mathbf{d}_k = -\nabla C(\mathbf{x}_k)$.
  2. Empregando a regra de Goldstein ou a regra de Wolfe-Powell calculamos o paso $\alpha_k > 0$ de xeito que $C(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < C(\mathbf{x}_k)$.
  3. Consideramos $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$.
- Comprobamos os criterios de converxencia:

$$\text{a)} \ \frac{\| \mathbf{x}_{k+1} - \mathbf{x}_k \|}{1 + \| \mathbf{x}_{k+1} \|} \leq \epsilon,$$
$$\text{b)} \ \| \nabla C(\mathbf{x}_{k+1}) \| \leq \eta.$$

  Se se cumpren ambos, párase o proceso e considérase $\mathbf{x}_{k+1} = \mathbf{x}^*$. De non ser así, tomarase $\mathbf{x}_k = \mathbf{x}_{k+1}$ e calcularase un novo $\mathbf{x}_{k+1}$.

O seguinte resultado establece as condicións que garanten a converxencia do método, de xeito que o problema (P') teña unha única solución.

Definición 2.24. Sexa $\{\mathbf{x}_k\}_{k \geq 0}$ unha sucesión tal que $\lim_{k \to \infty} \mathbf{x}_k = \mathbf{x}^*$, Dicimos que a sucesión ten orde de converxencia polo menos 1, se existen $k_0 \geq 0$ e $\beta \in (0, 1)$ tales que:

$$\| \mathbf{x}_{k+1} - \mathbf{x}^* \| \leq \beta \| \mathbf{x}_k - \mathbf{x}^* \|, \ \textit{para todo } k \geq k_0.$$

Teorema 2.25. *Supoñemos C unha función de custo diferenciable. Se existen dúas constantes $K > 0$ e $m > 0$ tales que, para todo $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$, verifícase que:*

$$\nabla \text{ C é lipschitziana, é dicir , } \| \nabla C(\boldsymbol{v}) - \nabla C(\boldsymbol{w}) \| \leq K \| \boldsymbol{v} - \boldsymbol{w} \|,$$

$$C \text{ é elíptica, é dicir, } (\nabla C(\boldsymbol{v}) - \nabla C(\boldsymbol{w}), \boldsymbol{v} - \boldsymbol{w}) \geq m \| \boldsymbol{v} - \boldsymbol{w} \|^2 .$$

Para todo a, b tales que $0 < a < b < \frac{2m}{K^2}$, se $\alpha_k \in [a, b]$, o problema (P') ten unha única solución e o método do gradiente é globalmente converxente a dita solución. A converxencia será de polo menos orde 1.

2.1.3. Gradiente Estocástico

Volvendo á función de custo cuadrática descrita en (2.14). Consideramos

$$C(\mathbf{x}) = \ell_{sq}(h, (\mathbf{x}, \mathbf{y})) = \frac{1}{2} \| h(\mathbf{x}) - \mathbf{y} \|^2, \text{ para cada } \mathbf{x} \in \mathcal{X}.$$

Podemos logo, entender a función de custo como

$$C = \frac{1}{n} \sum_{\mathbf{x} \in \mathcal{X}} C(\mathbf{x}),$$

sendo n a dimensión de $\mathbf{x}$. Polo tanto, para cada entrada $\mathbf{x} \in \mathcal{X}$ debemos calcular $\nabla C(\mathbf{x})$ para obter ∇C, o que suporá un gran custo computacional.

Para reducir dito custo proponse o algoritmo do Gradiente Estocástico. Este consiste en dividir os datos de adestramento en pequenos conxuntos aleatorios, aos que chamaremos mini-lotes e denotaremos por $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_m$, con $m \in \mathbb{N}$, menor que n pero o suficientemente grande como para que se verifique:

$$\frac{1}{n} \sum_{\mathbf{x} \in \mathcal{X}} \nabla C(\mathbf{x}) \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C(\mathbf{X}_j).$$

Polo tanto podemos aproximar o gradiente de C como:

$$\nabla C \approx \frac{\sum_{j=1}^{m} \nabla C(\mathbf{X}_j)}{m}$$

Observamos na ecuación anterior que en cada paso se empregara un mini-lote distinto (de xeito que se minimicen os erros con respecto ao valor real do gradiente). Podemos comparar nas seguintes figuras as traxectorias seguidas por ambos métodos.

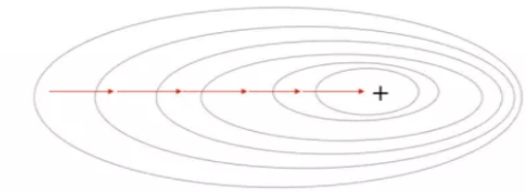

Figura 2.1: Descenso do gradiente

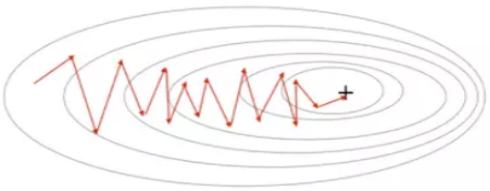

Figura 2.2: Descenso do gradiente estocástico. Ver [10]

.

Comparando as figuras 2.1-2.2, observamos que o gradiente converxe en menos iteracións que o gradiente estocástico. Porén, polo explicado anteriormente o custo computacional do segundo é moito menor, polo que, a pesar de que se realizan máis iteracións, o tempo requirido para minimizar a función de custo polo método do gradiente estocástico será moito menor.
A continuación, centrarémonos no funcionamento das redes neuronais, tratando de entender o seu funcionamento e de escoller cal dos algoritmos englobados nelas resolverá mellor o noso problema.

2.2. Redes Neuronais

Definimos as Redes Neuronais como un algoritmo dentro do Machine Learning inspirado no funcionamento do cerebro humano. Este constará dunha serie de unidades de procesamento, ás que nos referiremos como nodos ou neuronas, que estarán conectados entre si formando unha rede organizada en distintas capas. Dentro das distintas capas que forman a rede neuronal distinguimos: unha capa de entrada, outra de saída e unha serie de capas intermedias ás que chamaremos capas ocultas. A nosa rede recibirá unha serie de datos que asumiremos numéricos. No caso de ser categóricos, procederíase á binarización dos mesmos.

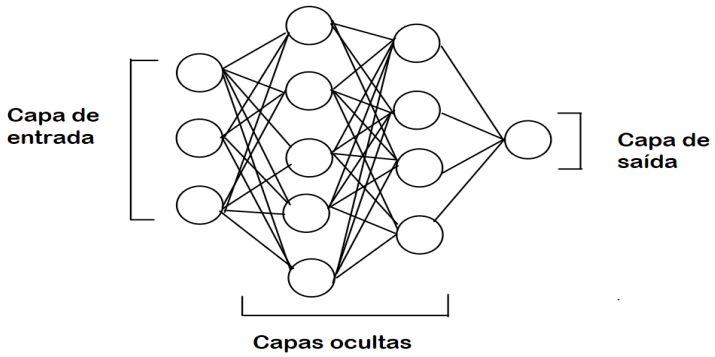

Figura 2.3: Rede neuronal

2.2.1. Perceptrón

Para poder entender ben como funcionan as redes neuronais comezaremos falando da súa unidade máis básica, o perceptrón. Este caracterízase por estar formado tan só por unha capa de entrada e outra de saída. Consideraremos en particular unha capa de saída composta por unha única neurona.

Denotaremos por x_i aos nodos da capa de entrada, con $i \in \{1, ..., n\}$, sendo n o número de datos subxacentes.
A función da capa de entrada, que está formada por tantos nodos como a dimensionalidade dos datos, é a de recibir e transmitir un único atributo numérico á capa de saída. Esta será a que se encargue de procesar estes valores matematicamente para xerar unha saída binaria, á que chamaremos **activación**. Esta virá dada en función dun limiar determinado, é dicir, un número real que é un parámetro da neurona.

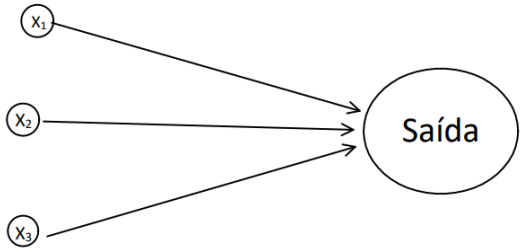

Figura 2.4: Perceptrón

A continuación introduciremos o concepto de peso.

Definición 2.26. Definimos o peso como o número real que lle é asignado a cada entrada en función da importancia que se busca que esta teña na saída.
O peso, no canto de estar relacionado cun nodo en si, está asociado ao eixo que une dito nodo de entrada co de saída. Deste xeito, sexa o peso asociado ao eixo que relaciona unha neurona de entrada k cunha neurona de saída j, denotarémolo por w_{jk}. Como no caso que estamos a tratar temos unha única neurona de saída, denotarémolo como w_i, indicando o índice i a entrada á que vén asignado.

Función de activación

Definimos a función de activación, como a aplicación pola que a neurona de saída transforma os valores que lle son dados polas neuronas de entrada, para obter un valor de saída.
Esta, no caso do perceptrón, terá en conta os valores de entrada, os seus pesos, e o limiar como se mostra a continuación:

$$saida = \begin{cases} 0 \quad se \quad \sum_{i=1}^{n} w_i x_i \ \leq \ limiar, \\ 1 \quad se \quad \sum_{i=1}^{n} w_i x_i \ > limiar. \end{cases} \tag{2.4}$$

Observamos que a rede neuronal nos dá unha solución determinada, que dependerá de se a suma ponderada dos distintos valores das entradas cos seus respectivos pesos, supera ou non o limiar establecido.
Definiremos como nesgo ao oposto do limiar $(-limiar)$, que denotaremos por b.
Consideremos $\mathbf{x} = (x_1, x_2, ..., x_n)$ o vector de entrada, $\mathbf{w} = (w_1, w_2, ..., w_n)$ o vector dos pesos e b o nesgo. Podemos expresar entón a ecuación anterior, como un produto escalar da forma:

$$saida = \begin{cases} 0 \quad se \quad \mathbf{w} \cdot \mathbf{x} + b \ \leq \ 0, \\ 1 \quad se \quad \mathbf{w} \cdot \mathbf{x} + b \ > 0. \end{cases} \tag{2.5}$$

O que esperamos da rede neuronal é que o resultado da función de activación coincida co valor esperado de saída. Así, o adestramento da rede consistirá en axustar os pesos e os nesgos de cara a obter os resultados esperados.

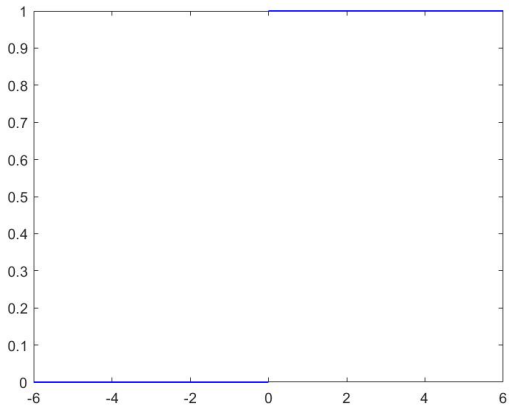

Figura 2.5: Función de activación do perceptrón

Observando a función de activación do perceptrón (2.5), vemos que un pequeno cambio nestes parámetros pode xerar grandes cambios na saída da rede.
Isto dificúltanos ver que modificacións graduais debemos realizar para achegarnos aos resultados desexados.
Para solventar este problema introduciuse un novo tipo de neurona artificial coñecida como neurona sigmoidea.

2.2.2. Neurona sigmoidea

As neuronas sigmoideas son un tipo de unidades de procesamento cunha estrutura similar á do perceptrón, pero diseñadas de forma que os pequenos cambios nos pesos e nesgos causen pequenas modificacións nas saídas; o que facilitará amplamente a aprendizaxe da rede.
A neurona sigmoidea está formada por unha serie de nodos de entrada, que a diferencia do perceptrón, tomarán valores continuos entre 0 e 1. Esta terá tamén unha serie de pesos $w_1, w_2, ..., w_n$ e un nesgo b.
Así, considerando igualmente $\mathbf{x} = (x_1, x_2, ..., x_n)$ e $\mathbf{w} = (w_1, w_2, ..., w_n)$, denotaremos por z ao cálculo intermedio $z = \mathbf{w} \cdot \mathbf{x} + b$, ao que nos referiremos como entrada ponderada. De donde, definimos a función de activación da neurona sigmoidea, á que chamaremos función sigmoidea, como segue:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.6}$$

Como podemos observar na figura 2.6, a saída dada por esta función de activación tomará tamén valores entre o 0 e o 1, pero será moito máis suave que a anterior.

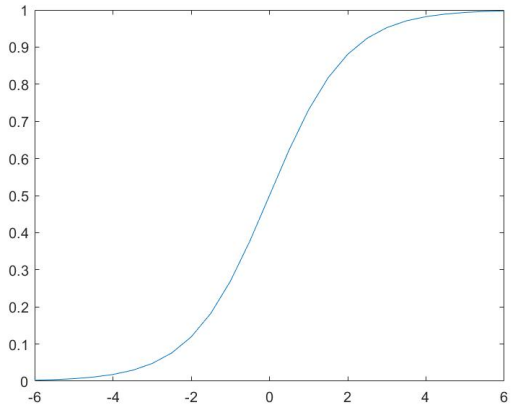

Figura 2.6: Función sigmoidea

A suavidade desta función implica que pequenos cambios nos pesos Δw_j e nos nesgos Δb, producen pequenos cambios na saída da rede $\Delta saida$. De feito, podemos calcular os cambios na saída como segue:

$$\Delta saida \approx \sum_j \frac{\partial saida}{\partial w_j} \Delta w_j + \frac{\partial saida}{\partial b} \Delta b.$$

Observamos que estamos ante unha función continua, o que, como observabamos na figura 2.6, fará que pequenos cambios nos pesos e no nesgo xeren pequenos cambios nas saídas.
A diferencia do que sucedía no perceptrón, a saída da función sigmoidea non toma valores binarios, senón que oscila entre o 0 e o 1. Isto resultará de utilidade nalgúns casos, porén noutros necesitaremos establecer regras para identificar un resultado co 0 ou co 1 segundo corresponda.

Función tanh

Dun xeito similar podemos definir a función de activación tanxente hiperbólica. Esta se calcula como:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{2.7}$$

Representamos esta función na figura 2.7. Nela podemos apreciar a súa similitude coa función sigmoidea.

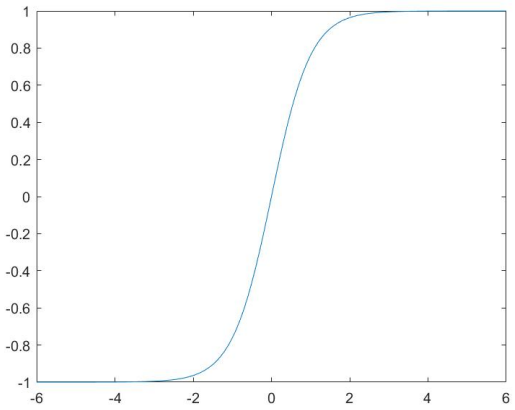

Figura 2.7: Función tanh

2.2.3. Redes neuronais multicapa

A partir da unión de varias neuronas sigmoideas, podemos crear redes neuronais multicapa. Estas contarán, ademais de coa capa de entrada e a de saída, cunha serie de capas intermedias, coñecidas como capas ocultas.

Notación 2.27. Dentotando por L á última capa da rede, referímonos ás capas intermedias ou ocultas como $l \in \{1, ..., L-1\}$.
Ademais, como xa dixemos cando introducíamos os pesos na definición 2.26, denotaremos por w^l_{jk} aos pesos para as conexións dende a neurona k na capa $l-1$ ata a neurona j na capa l. Neste caso, $\mathbf{w}^l$ será una matriz $n_l \times n_{l-1}$, sendo n_{l-1} e n_l o número de neuronas da capa $l-1$ e da capa l respectivamente.
A súa vez, denotaremos por b^l_k ao nesgo da neurona k na capa l, polo que $\mathbf{b}^l$ será un vector de dimensión n_l.

Definición 2.28. Para unha entrada $\mathbf{x}$, definiremos como $a(\mathbf{x}, \mathbf{w}, \mathbf{b})$ á saída da rede. Por como foron definidas a funcións de activación(2.4), (2.6), o vector de saídas dependerá de $\mathbf{x}$, $\mathbf{w}$ e b. Porén, para non complicar a notación denotarémola por $\mathbf{a}$.

Observación 2.29. Como entendemos a rede neuronal multicapa como a unión de distintas neuronas sigmoideas, podemos entender a^l_k como a saída dunha neurona k na capa l.
Podemos considerar ademais $a^0_k = x_k$, e dicir, entenderemos as entradas da rede neuronal como a activación da "capa 0".

Deste xeito, para $l \geq 1$, podemos considerar a entrada ponderada da neurona k na capa l como $z_k^l = w_{jk}^l a_k^{l-1} + b_k^l$. De xeito que a entrada ponderada da capa l calcularáse como $\mathbf{z}^l = \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l$.

Notación 2.30. Resumimos enton que estamos a denotar por:

1. z_k^l a entrada ponderada da neurona k na capa l.
2. a_k^l a activación da neurona k na capa l.

Esta, para unha función de activación σ, calcúlase como $a_k^l = \sigma(z_k^l)$.

Vemos un exemplo desta notación na figura 2.8.

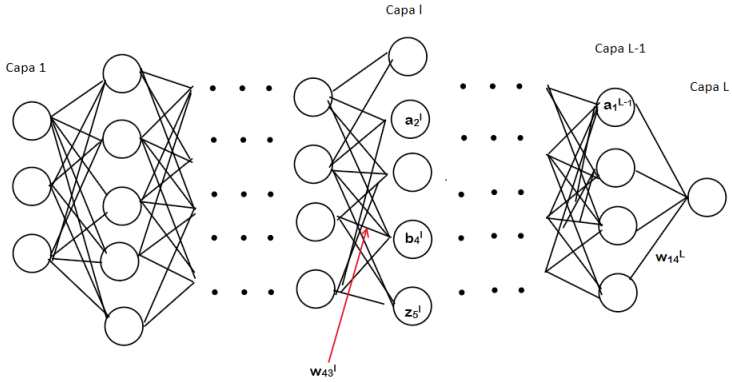

Figura 2.8: Representación da notación a empregar

Dependendo da organización das capas distinguimos os seguintes tipos de redes neuronais:

1. **Rede neuronal *feedforward***
   Chamamos rede neuronal *feedforward* ou rede neuronal de avance a aquelas redes nas que, como viñamos describindo ata o momento, a saída dunha capa resulta a entrada da seguinte. Neste tipo de redes non existen bucles, de forma que a información dunha neurona sempre se transmite á seguinte, nunca se retroalimenta.

2. **Rede neuronal recorrente**
   Chamamos rede neuronal recorrente a aquelas que, pola contra, si que contan con bucles. Nestas redes a saída dunha neurona pode afectar posteriormente á entrada da mesma. A idea detrás deste tipo de redes son neuronas que se activan por un periodo de tempo antes de quedar inactivas. Estas poderán estimular outras neuronas, que estarán activas a súa vez por un tempo limitado e así sucesivamente. Así, mentres non se activa a seguinte neurona, a saída dun nodo será empregada na entrada do mesmo, o que fai que os bucles non resulten un problema no funcionamente da rede.

2.2.4. Método do Gradiente en Redes Neuronais

Nas Redes Neuronais, o método do Gradiente é análogo ao do Machine Learning en xeral. Con respecto á función de custo, o que estamos a avaliar é ata que punto os pesos e o nesgo empregados nos dan os resultados que buscamos.

Definición 2.31. Sexa $\mathbf{x}$ unha entrada da rede e $y(\mathbf{x})$ a saída esperada correspondente, sexa $\mathbf{a}$ o vector de saídas da rede, definimos a función de custo cuadrática para redes neuronais como:

$$C(\mathbf{w}, \mathbf{b}) := \frac{1}{2n} \sum_x \parallel y(\mathbf{x}) - \mathbf{a} \parallel^2, \tag{2.8}$$

sendo n o número de entradas dos datos de adestramento.

Observación 2.32. Por como está definido o vector de saídas na definición 2.28, a función de custo (2.8) depende de $\mathbf{w}$ e $\mathbf{b}$.

Imos considerar que a capa de saída esta formada por unha única neurona. Estamos a buscar un vector $(\mathbf{w}, b)$ que minimice a función de custo, sendo o nesgo un escalar b e os pesos un vector $\mathbf{w}$ de dimensión n_{L-1}.
Denotamos por w_j cada elemento de $\mathbf{w}$, sendo $j \in \{1, .., n_{L-1}\}$.
Tal e como viamos na sección 2.1.2, partimos dun $(\mathbf{w}_0, b_0)$ dado, sendo

$$\mathbf{w}_0 = ((w_1)_0, ..., (w_j)_0, ..., (w_{n_{L-1}})_0)$$

e b_0 un escalar. Empregamos o método do gradiente para un paso fixo. Para $k \geq 0$ calculamos:

$$(w_j)_{k+1} = (w_j)_k - \eta \frac{\partial C}{\partial (w_j)_k}, \ j \in \{1, .., n_{L-1}\},$$

$$b_{k+1} = b_k - \eta \frac{\partial C}{\partial b_k}.$$

Do mesmo xeito para o Descenso Gradiente estocástico, dividimos o conxunto de pesos $\mathbf{w}$ e o conxunto de nesgos b en minilotes que denotamos por $\mathbf{W}_1, ..., \mathbf{W}_m$ e $b_1, ..., b_m$ respectivamente. Contamos cun $m \in \mathbb{N}$ o suficientemente grande como para que verifique a condición exposta no capítulo anterior para o algoritmo do gradiente estocástico. Calculamos os novos pesos e nesgos como segue:

$$(\mathbf{w})_{k+1} = (\mathbf{w})_k - \frac{\eta}{m} \sum_{i=1}^{m} \frac{\partial C(\mathbf{W}_i)}{\partial (\mathbf{w})_k},$$

$$b_{k+1} = b_k - \frac{\eta}{m} \sum_{i=1}^{m} \frac{\partial C(b_i)}{\partial b_k}.$$

2.3. Diferenciación automática

Co fin de calcular as derivadas da función de custo en función do peso e do nesgo, introducimos un algoritmo coñecido como diferenciación automática. Mentres que a derivación simbólica chega á solución mediante a aplicación das regras de derivación, e a derivación numérica busca a derivada nun punto mediante unha aproximación; a derivación automática preséntase como unha mestura de ambas que busca evitar as súas limitacións e aproveitar as súas vantaxes. Deste xeito, no canto das cadeas de expresións empregadas na diferenciación simbólica, utiliza fórmulas exactas. Ademais, a diferenza da diferenciación numérica, non implica erros de aproximación.
A idea de base é implementar nun entorno numérico as regras básicas de derivación do cálculo.

Exemplo 2.33. Consideramos a función $f(x) = x\,\mathrm{sen}^2(e^x)$. Derivando a función por medio da diferenciación automática obteremos uns resultados como os do cadro 2.1.

$x = 2$	$x' = 1$
$y_1 = e^x = 7.39$	$y_1' = e^x x' = 7.39$
$y_2 = sen(y_1) = 0.89$	$y_2' = cos(y_1) y_1' = 3.31$
$y_3 = x y_2 = 1.79$	$y_3' = x' y_2 + x y_2' = 7.52$

Cadro 2.1: Resultados da derivación automática para a función $f(x) = x\,\mathrm{sen}^2(e^x)$

Búscase un programa que, a partir dunha función dada, nos dea todas as sentencias da cadro 2.1. Para isto empregaremos a programación orientada a obxectos. Esta permítenos definir x como un obxecto que contén tanto o valor de x como o valor da derivada de x en x.

Exemplo 2.34. Sexa $f(x) = 2x + 3$ en $x = 2$, definimos $x = [2, 1]$ que indica o valor de x, 2, e o da súa derivada, 1. Como 3 é unha constante con derivada 0, teremos $[2*2+3, 2*1+0] = [7, 2]$.

Exemplo 2.35. Do mesmo xeito, considerando as derivadas das funcións e^x e $\mathrm{sen}(x)$ retomamos o exemplo 2.33. Para calcular o valor da derivada de $f(x) = x\,\mathrm{sen}(e^x)$ en $x = 2$, establecemos $x = [2, 1]$ e esta será da forma: $[2 * sen(e^2), 1 * sen(e^2) + 2 * cos(e^2) * e^2 * 1] = [1.79, 7.52]$.

Para calcular a gradiente o procedemento será análogo; bastará considerar a derivada como un vector. Vémolo máis claramente no seguinte exemplo.

Exemplo 2.36. Consideramos $x = (2, [1, 0])$ $y = (3, [0, 1])$. Deste xeito calculamos o gradiente da función $x * y$ como $[1, 0] * 3 + 2 * [0, 1] = [3, 2]$.

A continuación, imos ver como se calcula o gradiente para funcións máis complexas. Este poderase calcular con dous métodos dentro da diferenciación automática. Neste traballo empregaremos unicamente un deles, a diferenciación automática cara atrás; porén para explicar este, exporase brevemente a diferenciación automática cara diante. Para facilitar a comprensión de ambos algoritmos, estes serán explicados a partir do seguinte exemplo.

Exemplo 2.37. Dada a función $f(x, y, z) = (x * y) * sen(y * z)$, representase o seu grafo computacional na figura 2.9, no que cada nodo representa unha operación intermedia.

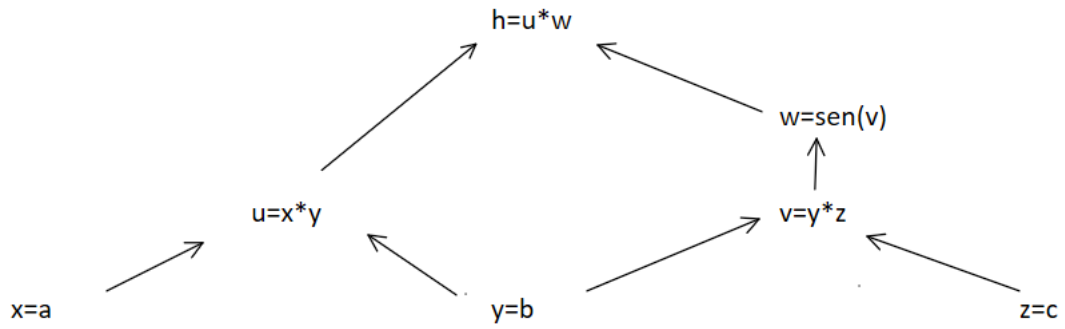

Figura 2.9: Grafo computacional da función $f(x, y, z) = (x * y) * sen(y * z)$

A diferenciación automática cara diante é o método no que se traballa na dirección das frechas da figura 2.9, calculando en cada nodo o gradiente, ademais do valor da variable.

A diferenciación automática cara atrás comeza cunha avaliación na mesma dirección. Nesta calcúlanse e gárdanse os valores de cada variable, ademais das derivadas parciais dos nodos con respecto aos argumentos inmediatamente anteriores. Para o exemplo da figura 2.9 calcúlase como segue:

$$\begin{array}{lll} x = a, & & \\ y = b, & & \\ z = c, & & \\ u = ab, & \frac{\partial u}{\partial x} = b, & u\frac{\partial u}{\partial y} = a, \\ v = bc, & \frac{\partial v}{\partial x} = c, & \frac{\partial v}{\partial z} = b, \\ w = \text{sen}(v), & \frac{\partial w}{\partial v} = cos(v), & \\ h = wu, & \frac{\partial h}{\partial u} = w, & \frac{\partial h}{\partial u} = u. \end{array}$$

Unha vez percorrido todo o grafo, este percorrerase cara atrás; acumulando os produtos das

derivadas parciais que acabamos de calcular. Definiremos a acumulación en cada nodo como unha variable adxunta, denotada cunha barra sobre a variable. Consideramos primeiro $\bar{h} = 1$, $\bar{w} = \frac{\partial h}{\partial w}$ e $\bar{u} = \frac{\partial h}{\partial u}$. Para calcular as variables adxuntas do resto dos nodos, empregaremos a regra da cadea. Avanzamos no sentido contrario das frechas do grafo da figura 2.9, e calculamos, por exemplo,

$$\bar{v} = \bar{w}\frac{\partial w}{\partial v} = \frac{\partial h}{\partial w}\frac{\partial w}{\partial v} = \frac{\partial h}{\partial v},$$

calculándose o resto de xeito análogo. Exemplificamos o caso da variable $\bar{x}$ xa que é a única na que atopamos dous eixos saíndo dun mesmo nodo e, aínda que o razonamento é o mesmo, alxébricamente calcúlase dun xeito diferente. Esta será da forma:

$$\bar{y} = \bar{u}\frac{\partial u}{\partial y} + \bar{v}\frac{\partial v}{\partial y} = \frac{\partial h}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial h}{\partial v}\frac{\partial v}{\partial y} = \frac{\partial h}{\partial y}. \tag{2.9}$$

Así, no noso caso, estas variables adxuntas serán da forma:

$$\begin{gathered}
\bar{h} = 1, \\
\bar{w} = \bar{h}\tfrac{\partial h}{\partial w}, \\
\bar{v} = \bar{w}\tfrac{\partial w}{\partial v}, \\
\bar{u} = \bar{h}\tfrac{\partial h}{\partial u}, \\
\bar{x} = \bar{u}\tfrac{\partial u}{\partial x}, \\
\bar{y} = \bar{u}\tfrac{\partial u}{\partial y} + \bar{v}\tfrac{\partial v}{\partial y}, \\
\bar{z} = \bar{v}\tfrac{\partial v}{\partial z}.
\end{gathered}$$

Vexamos como podemos relacionar o que acabamos de explicar coas redes neuronais, para poder calcular grazas a ela as derivadas parciais que buscabamos. Comecemos primeiro cunha rede neuronal simple.

Exemplo 2.38. Dada unha rede neuronal simple como a da figura 2.10.

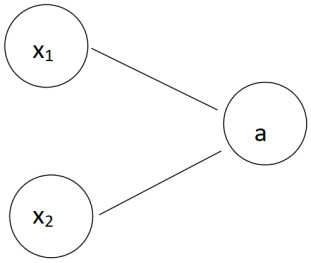

Figura 2.10: Representación dunha rede neuronal simple

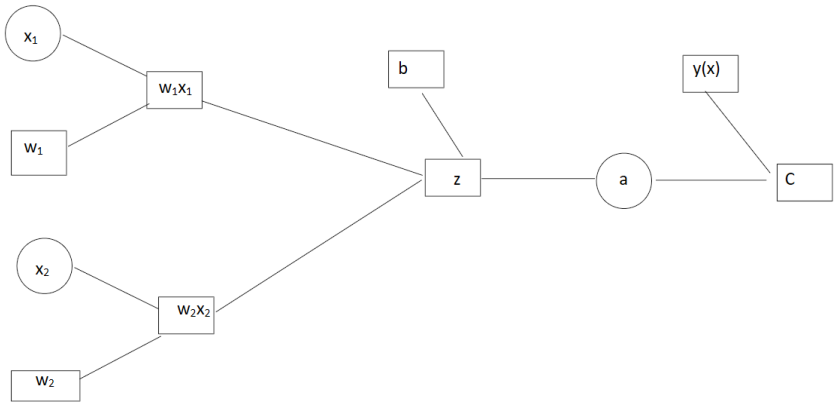

Figura 2.11: Grafo computacional para unha rede neuronal simple

Vemos que a podemos relacionar cun grafo computacional como o da figura 2.11.
Sendo $z = (w_1x_1 + w_2x_2) + b$, $a = \sigma(z)$ e $C = \frac{1}{2} \parallel y(x) - a \parallel^2$. Aplicando a diferenciación automática cara atrás temos:

$$
\begin{array}{rlll}
x_1 = a, & & & \\
w_1 = b, & & & \\
x_2 = c, & & & \\
w_2 = d, & & & \\
u = ab, & \frac{\partial u}{\partial x_1} = b, & \frac{\partial u}{\partial w_1} = a, & \\
v = cd, & \frac{\partial v}{\partial x_2} = d, & \frac{\partial v}{\partial w_2} = c, & \\
z = (u + v) + b, & \frac{\partial z}{\partial u} = 1, & \frac{\partial z}{,}\partial v = 1, & \frac{\partial z}{,}\partial b = 1, \\
a = \sigma(z), & \frac{\partial a}{,}\partial z = \sigma'(z), & & \\
C = \frac{1}{2} \parallel y(x) - a \parallel^2, & \frac{\partial C}{\partial a}. & &
\end{array}
$$

Calculamos tamén as variables adxuntas:

$$\begin{aligned}
\bar{C} &= 1, \\
\bar{w} &= \bar{C}\frac{\partial C}{\partial a}, \\
\bar{y}(x) &= \bar{C}\frac{\partial C}{\partial y(x)}, \\
\bar{a} &= \bar{C}\frac{\partial C}{\partial a}, \\
\bar{z} &= \bar{a}\frac{\partial a}{\partial z}, \\
\bar{b} &= \bar{z}\frac{\partial z}{\partial b}, \\
\overline{w_1 x_1} &= \bar{z}\frac{\partial z}{\partial w_1 x_1}, \\
\bar{x_1} &= \overline{w_1 x_1}\frac{\partial w_1 x_1}{\partial x_1}, \\
\bar{w_1} &= \overline{w_1 x_1}\frac{\partial w_1 x_1}{\partial w_1}, \\
\overline{w_2 x_2} &= \bar{z}\frac{\partial z}{\partial w_2 x_2}, \\
\bar{x_2} &= \overline{w_2 x_2}\frac{\partial w_2 x_2}{\partial x_2}, \\
\bar{w_2} &= \overline{w_2 x_2}\frac{\partial w_2 x_2}{\partial w_2}.
\end{aligned}$$

De aquí concluímos que :

$$\begin{aligned}
\frac{\partial C}{\partial b} &= \frac{\partial C}{\partial a}\sigma'(z), \\
\frac{\partial C}{\partial w_1} &= \frac{\partial C}{\partial a}\sigma'(z)x_1, \\
\frac{\partial C}{\partial w_2} &= \frac{\partial C}{\partial a}\sigma'(z)x_2.
\end{aligned}$$

Unha vez vimos como funciona o proceso para un caso sinxelo, podemos extrapolalo a un caso xeral.

Imaxinemos agora que estas capas teñen varias neuronas. Sexa n_{l-1} o número de neuronas na capa $l-1$ e n_l o número de neuronas na capa l, facendo referencia k a unha neurona na capa $l-1$ e j a unha neurona na capa l; posto que w^l_{jk} fai referencia ao peso do eixo que une dúas únicas neuronas, o razonamento é análogo a o que viamos no caso anterior, e polo tanto:

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial a j^l}\sigma'(z^l_j)a^{l-1}_k.$$

Porén, un mesmo nesgo emprégase no cálculo da activación de todas as neuronas da capa l. Polo tanto, recordando o que viamos en (2.9) chegamos a que:

$$\frac{\partial C}{\partial b^{l-1}_k} = \sum_{j=1}^{n_l}\frac{\partial C}{\partial a^l_j}\sigma'(z^l_j).$$

Capítulo 3

Resolución do problema de predición do tempo con Redes Neuronais

Neste capítulo retomaremos o problema exposto no Capítulo 1. A partir do conxunto de datos xa presentado, preténdese crear un programa capaz de predicir a velocidade do vento baseándonos na metodoloxía exposta no Capítulo 2. Construiremos o modelo de cara a predicir a variable 'WindSpeed9am', que nos da información sobre a velocidade do vento ás 09:00h, podendo modificar o modelo de cara a predicir calquera outra variable.
Para construír o noso modelo, baseámonos no código recollido en [8]. Ademais, neste capítulo, describirase un tipo de redes neuronais coñecidas por LSTM. Isto foi posible grazas aos resultados descritos en [7]. Antes de comezar, introduciremos brevemente as librarías necesarias para construír o noso modelo.

3.1. Instalación de librarías

Tensorflow é unha plataforma de código aberto destinada á aprendizaxe automática ou Machine Learning. Dentro desta empregaremos a libraría *keras*, que nos permitirá construír e adestrar modelos de aprendizaxe profunda.

Na figura 3.1 vemos como se importan distintas funcións da libraría *keras*. Ainda que as explicaremos máis adiante, cabe destacar que a función *SGD* fai referencia a un método de optimización que xa estudamos no capítulo anterior: o método do gradiente estocástico.
A libraría *sklearn* proporcionaranos ferramentas para implementar o noso modelo. En particular empregaremos a función *MinMaxScaler* para escalar os nosos datos.

```
import sklearn
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow import keras
from keras import regularizers
from keras.models import Sequential
from keras.layers import Dense,Input, LSTM
from keras.optimizers import SGD
```

Figura 3.1: Instalación das librarías necesarias para construír e adestrar o modelo

3.2. Pre-procesamento dos datos

Tras o estudo e a preparación dos datos do Capítulo 1, onde escollimos aqueles cos que nos interesaba traballar e nos aseguramos de que as súas variables foran numéricas, continuamos a preparación dos datos, adaptándoos ao modelo co que queremos traballar.
En primeiro lugar, e tal e como viamos no Capítulo 2, debemos dividir o noso conxunto de datos en dous conxuntos totalmente distintos: o conxunto de adestramento e o conxunto de proba ou validación.
É importante destacar que, neste caso, debido á natureza temporal dos nosos datos, a súa orde importa á hora de realizar novas predicións, polo que escolleremos estes conxuntos dun xeito 'ordenado'. Sabemos ademais, que o conxunto de adestramento debe ser maior ao conxunto de proba. Neste caso, consideraremos un conxunto de adestramento que supón o 75 % dos datos e un conxunto de proba suporá, polo tanto, o 25 % destes. Deste xeito, o noso conxunto de datos de adestramento estará formado polas primeiras 2041 entradas ou filas do conxunto de datos e o conxunto de proba polas restantes.

Viamos tamén no mapa de calor do Capítulo 1, na figura 1.5, unha forte correlación entre a temperatura máxima e mínima, a presión ás 09:00h e ás 15:00h e a temperatura ás 09:00h e ás 15:00h. Posto que estes parámetros son redundantes, traballaremos só cun deles; por exemplo, no caso da presión, traballaremos unicamente coa presión ás 09:00h.
Unha vez seleccionadas as variables coas que imos traballar, posto que estas se moven en diferentes rangos, debemos levar a cabo un reescalamento do noso conxunto de datos. Facendo uso da función de *sklearn.preprocessing*, *MinMaxScaler*, seremos capaces de escalar os nosos datos nun rango entre 0 e 1.

O obxectivo do traballo é realizar futuras predicións a partir das observacións xa recollidas no noso conxunto de datos. O noso modelo adestrarase facendo predicións para un período de tempo a partir das observacións recollidas durante un período anterior e maior. Referirémonos ao número de días para os que se realiza a predición como 'future' e ao número de días dos que se toman as observacións como 'past'.
Para adestrar o noso modelo deste xeito, debemos dividir os nosos datos en intervalos iguais.

Supoñamos $'past' = n$ e $'future' = m$ para $n, m \in \mathbb{N}$.
Para o conxunto de datos de adestramento, definimos un conxunto de datos de entrada que se corresponderá cos valores dos datos de adestramento. O seu conxunto de etiquetas comezará a partir das n primeiras observacións do conxunto de adestramento e rematará pasadas n observacións do mesmo.
Para os datos de entrada do conxunto de proba e as súas etiquetas, consideramos as entradas como as filas do conxunto de proba agás as $n+m$ últimas , e as etiquetas a partir das n primeiras filas.
Para dividir a nosa secuencia de datos en intervalos iguais, empregaremos a función de *keras.preprocessing*: timeseries-dataset-from-array.
Keras.preprocessing é un módulo da libraría *keras* que se encarga do preprocesamento e do tratamento dos datos. Esta función producirá lotes de entradas e etiquetas de series temporais, tanto para o conxunto de adestramento como para o de proba.
Unha vez establecidos estes lotes, definimos as entradas e as etiquetas coa función da figura 3.2.

```
for batch in dataset_train.take(1):
    inputs, targets = batch
```

Figura 3.2: Entradas e etiquetas

A continuación, considerando $'past' = 60, 'future' = 6$, vemos na figura 3.3 a forma das entradas (inputs) e das etiquetas (targets) grazas á función *shape*, que nos proporciona unha tupla que recolle o número de elementos de cada dimensión dos obxectos.

```
Input shape: (32, 60, 10)
Targets shape: (32, 1)
```

Figura 3.3: Dimesionalidade das entradas e das etiquetas

3.3. Construción do modelo

Unha vez listo o noso conxunto de datos, pasamos a construír o modelo.

Explicabamos no capítulo anterior, na sección 2.2.3, que existían dous tipos de redes neuronais: as redes neuronais de avance e as redes neuronais recorrentes. Dado que buscamos predicir sucesos futuros a partir de entradas secuenciais anteriores, traballaremos con redes neuronais recorrentes. Dentro destas, empregaremos un tipo de rede neuronal coñecidas como LSTM (Long

short-term memory), posto que detectan o comportamento estacional e as tendencias propias deste tipo de información.

3.3.1. Redes neuronais LSTM

As redes LSTM son un tipo de rede neuronal recorrente que se caracteriza por ser capaz de gardar datos da secuencia temporal.
Nas redes neuronais recorrentes, e en particular nas LSTM, cada nodo procesa información durante un número predeterminado de pasos de tempo. Denotamos a estes por t. O paso de tempo indicaranos o número de veces que pasa a información por un mesmo nodo. Cada neurona dunha rede LSTM conta con tres tipos de entradas: a entrada usual, que existe en todas as redes neuronais; a saída do paso de tempo previo, propia das redes neuronais recorrentes, e a memoria almacenada ata o momento, específica deste tipo de redes neuronais. Así mesmo, estas redes contarán con dous tipos de saídas: a saída usual e a memoria almacenada logo deste paso.

Notación 3.1. Para un paso de tempo t:
Denotamos por X_t a entrada usual, por H_{t-1} a saída do paso de tempo previo e por C_{t-1} a memoria almacenada.
U é o vector de pesos asociado a variable X_{t-1} e W o vector de pesos asociado a variable H_{t-1}.
Así mesmo, denotamos por H_t a saída usual e por C_t a memoria almacenada logo deste paso de tempo.

Unha unidade LSTM está composta de:

- Porta de olvido (ten unha función de activación sigmoidea(2.6)).
- Capa candidata (ten unha función de activación tanxente(2.7)).
- Porta de entrada (ten unha función de activación sigmoidea).
- Porta de saída (ten unha función de activación sigmoidea).

Antes de continuar, debemos introducir unha operación coñecida como producto de Hadamard ou producto por elemento.

Definición 3.2. Dados dous vectores $\mathbf{u}$ e $\mathbf{v}$ de dimensión $m \in \mathbb{N}$ definimos a operación de Hadamard, que denotamos por $\mathbf{u} * \mathbf{v}$, como a operación que, para cada $j \in \{1, ..., m\}$, cumple $(\mathbf{u} * \mathbf{v})_j = u_j v_j$.

Notación 3.3. Para un paso de tempo t:
Denotamos a porta de olvido como f_t. Así mesmo, denotamos os pesos introducidos anteriormente, U e W, asociados á porta de olvido como U_f e W_f, respectivamente.
A función de activación da porta de olvido virá dada por :

$$f_t = \sigma(X_t * U_f + H_{t-1} * W_f).$$

Denotamos a capa candidata como $\bar{C}_t$, denotando os pesos U e W asociados a ela como U_c e W_c respectivamente.
A súa función de activación está definida por:

$$\bar{C}_t = \tanh(X_t * U_c + H_{t-1} * W_c).$$

Denotamos a porta de entrada como I_t, denotando os pesos U e W asociados a esta como U_i e W_i. A función de activación da porta de entrada ven dada por:

$$I_t = \sigma(X_t * U_i + H_{t-1} * W_i).$$

Por último, denotamos a porta de saída como O_t, denotando os pesos U e W asociados a esta como U_o e W_o. Defínese a súa función de activación como:

$$O_t = \sigma(X_t * U_o + H_{t-1} * W_o),$$

Coñecendo isto, podemos observar na figura 3.4 o funcionamento dun paso de tempo dun nodo LSTM.

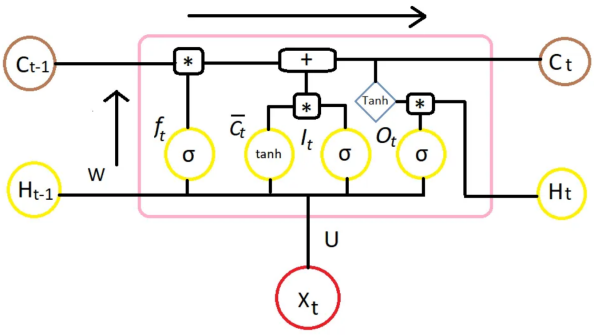

Figura 3.4: Paso de tempo dunha neurona dunha rede LSTM. Ver [7]

.

A porta de olvido ten como función controlar a cantidade de memoria que se continua a almacenar. Ademais, a capa candidata e a porta de entrada controlan canta información nova se almacena na memoria. Desde xeito, a memoria almacenada no paso de tempo t calcúlase como:

$$C_t = f_t * C_{t-1} + I_t * \bar{C}_t.$$

Por último, a saída virá dada en función da capa de saída como segue:

$$H_t = O_t * tan(C_t).$$

Unha vez presentadas as capas LSTM, construímos o noso modelo como segue:

```
inputs = keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
lstm_out = keras.layers.LSTM(32)(inputs)
outputs = keras.layers.Dense(1,kernel_regularizer=regularizers.l2(0.1))(lstm_out)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=tf.keras.optimizers.experimental.SGD(learning_rate=learning_rate), loss="mse")
```

Figura 3.5: Estrutura do modelo

Importamos de *keras.layers* as distintas capas que constituirán o modelo.
En primeiro lugar, vemos na figura 3.5, que este conta cunha capa de entrada (Input). Esta esperará que as entradas sexan lotes de dimensión 60×10, polo dimensionalidade exposta na figura 3.3 . A maiores, o modelo consta dunha capa LSTM con 32 nodos, como a que acabamos de explicar. Por último, conta cunha capa *Dense*, que será empregada como capa de saída. Esta capa estará conectada a todas as neuronas da capa anterior e terá como obxectivo cambiar a dimensionalidade da rede, de xeito que unicamente teñamos un nodo de saída; o que facilitará amplamente a avaliación do modelo.
Nesta capa empregamos tamén o regulador L2 de *keras*, que nos axudará a previr problemas na aprendizaxe como o sobreaxuste e o subaxuste.

Observación 3.4. O sobreaxuste e o subaxuste son fenómenos que poden ocorrer durante o adestramento do modelo. O sobreaxuste fai referencia a un modelo que se axusta demasiado ben os datos de adestramento, de xeito que é incapaz de xeneralizar o aprendido e aplicalo aos datos de validación. Pola súa parte, o subaxuste ocorre cando o modelo non é capaz de modelar os datos de adestramento e, polo tanto, tampouco de realizar boas predicións para o conxunto de validación.

Configuramos o noso modelo para o adestramento empregando como optimizador SDG (stochastic descent gradient). Este optimizador traballa co método de descenso que espoñíamos no capítulo anterior. Como función de custo usamos a función de custo cuadrárica.

3.4. Adestramento e análise do modelo

Unha vez diseñado o noso modelo establecemos os parámetros cos que procederemos a adestralo.
Consideramos un paso fixo $\eta = 0.1$. O tamaño dos minilotes que definíamos cando introducíamos o gradiente estocástico será 32. Nun principio establecerase un total de 250 iteracións. Ainda así, empregamos a función *EarlyStopping* de *Keras* para parar o adestramento cando a función de

custo do conxunto de validación deixe de mellorar, entendendo como mellorar o feito de minorarse. Deste xeito, o adestramento do noso modelo pode non chegar a realizar todas as iteracións indicadas nun principio.

A continuación, imos adestrar o noso modelo para un $'past' = 60$ e un $'future' = 6$. Durante o adestramento mídense as funcións de custo para o conxunto de adestramento e o de proba. Comparámolos na figura 3.6 para avaliar o adestramento do noso modelo.

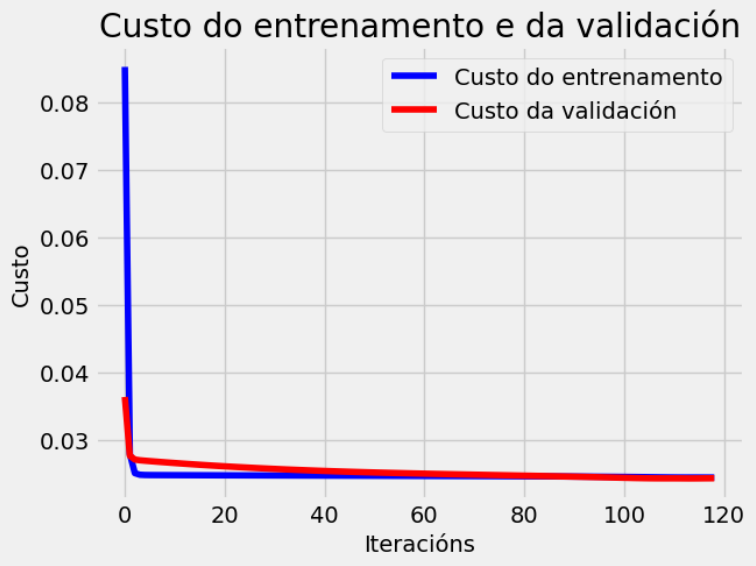

Figura 3.6: Curvas de custo do adestramento e da validación para 'past'=60

Observamos que ambas curvas decrecen sen moita diferencia entre elas, ata que os valores das funcións de custo se estabilizan. Concluímos de aquí que o modelo está ben axustado e que non hai problemas de sobreaxuste nin subaxuste.

Durante o adestramento do modelo medíronse tamén dous erros que nos axudaron a determinar a precisión do mesmo. Estes miden as diferenzas entre os valores esperados e os valores preditos polo modelo.

Mediuse o erro cuadrático medio (MSE), que foi empregado á súa vez como función de custo. Este calcúlase como a media da diferencia ao cadrado dos valores esperados e os valores preditos. Estudouse tamén a raíz do erro cuadrático medio (RMSE) que se calcula simplemente como a raíz cadrada do erro anterior. Esta fai que a escala dos erros sexa igual á dos datos cos que estabamos a traballar, polo que nos permite comparalos máis facilmente.

Representamos o resultado de ambos para o adestramento e a validación no cadro 3.1. Ambos erros son considerablemente baixos, o que nos leva a supoñer que o noso modelo esta predicindo relativamente ben.

	Adestramento	Validación
MSE	0.0241	0.0239
RMSE	0.1551	0.1546

Cadro 3.1: Erros de adestramento e validación para 'past'=60

Vexamos entón cales son as predicións que o noso modelo é capaz de calcular. A figura 3.7 mostra as predicións para a velocidade do vento ás 09:00h realizadas polo modelo e as compara co valor esperado.

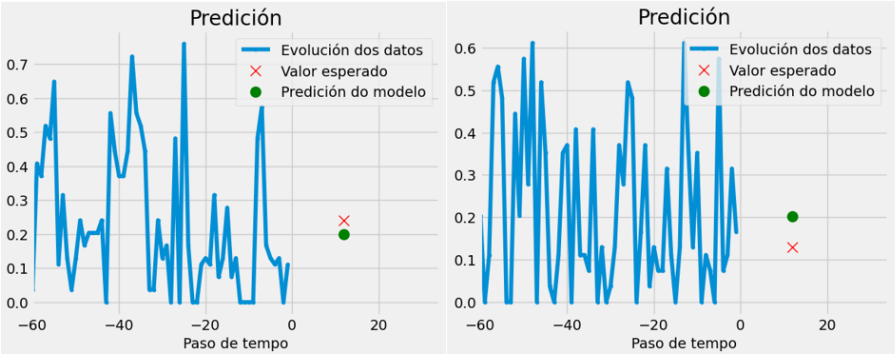

Figura 3.7: predicións da velocidade do vento ás 09:00h

Vemos, como era de esperar, que as nosas predicións non son exactas, pero que se axustan relativamente ben.

O feito de que os nosos datos tivesen un comportamento estacional e periódico, fainos pensar que igual o noso modelo é capaz de predicir mellor se, no canto de adestrarse coas observacións recollidas durante 60 días, se adéstrase con aquelas tomadas durante un ano enteiro. adestramos o noso modelo para 'past'=365 e 'future'=6. Neste caso escollemos o paso $\eta = 0.01$.

Analizamos as curvas da función custo do adestramento e a validación da figura 3.8.

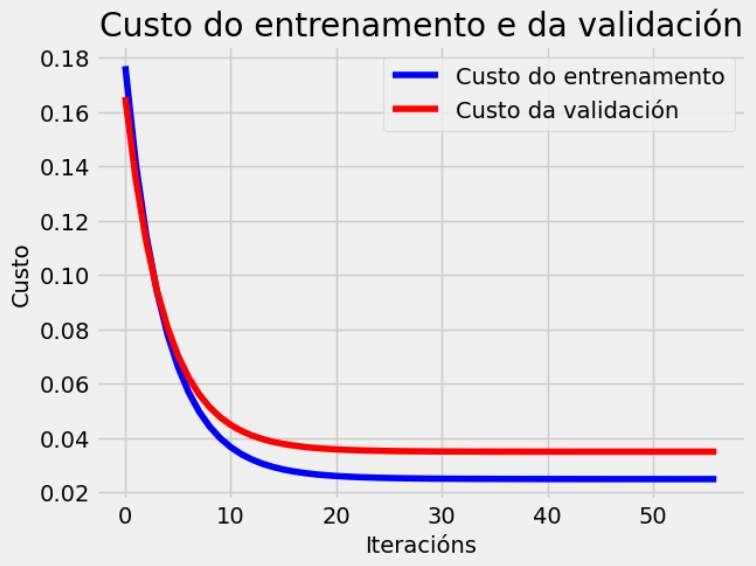

Figura 3.8: Curvas de custo do adestramento e da validación para 'past'=365

Volvemos a observar un bo axuste do modelo, xa que ambas curvas se estabilizan sen que haxa moita diferenza entre elas.
Calculamos tamén o erro cuadrático medio (MSE) e a raíz do erro cuadrático medio (RMSE), e obtivemos os resultados que se mostran no cadro 3.2.

	Adestramento	Validación
MSE	0.0246	0.1570
RMSE	0.0368	0.1919

Cadro 3.2: Erros de adestramento e validación para 'past'=365

Obtemos uns erros moi similares ainda que lixeiramente maiores, o que nos da a pensar que as predicións neste caso son algo peores.

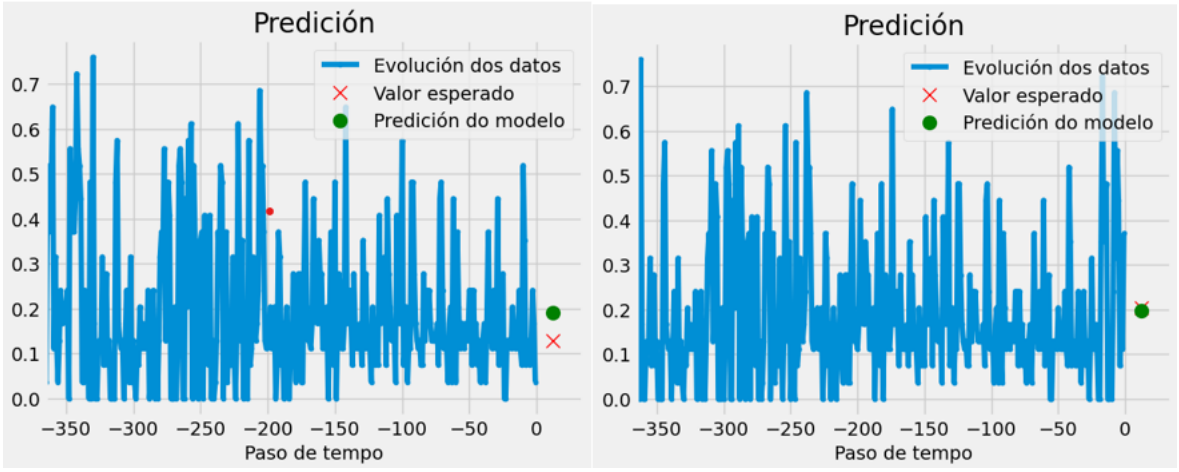

Figura 3.9: predicións da velocidade do vento ás 09:00h para 'past'=365

Porén, ao representar na figura 3.9 as predicións para a velocidade do vento ás 09:00h, vemos que estas se axustan mellor aos valores esperados.
A maiores, observabamos nas figuras 3.6 e 3.8, como o modelo converxe en moitas menos iteracións para 'future'=360. Isto sumado ao feito de que o paso escollido neste caso é menor, implica que o modelo remata o seu adestramento moito máis rápido no segundo caso. Isto pódese deber a que ao dividir anualmente as observacións, lle resulta moito máis fácil identificar os patróns do noso conxunto de datos.
Concluímos así, que o modelo construído funciona adecuadamente en calquera dos dous casos. Aínda así, o segundo pode resultar máis interesante polo feito de que converxe máis rápidamente.

Capítulo 4

Conclusións

Para a construción do modelo que buscabamos obter a través deste traballo, empregouse un algoritmo dentro do Machine Learning, coñecido como Redes Neuronais. Este modelo está constituído por tres capas: unha capa de entrada, unha capa de neuronas LSTM e unha capa de saída. As redes neuronais que conteñen neuronas LSTM están incluídas dentro dun tipo de redes ás que nos referimos como redes neuronais recorrentes.

Para avaliar o noso modelo empregouse unha función de custo cuadrática. Esta foi minimizada mediante a aplicación do algoritmo do descenso de gradiente estocástico. Ambos conceptos foron introducidos no Capítulo 2.

Para adestrar o noso modelo de cara a obter predicións climatolóxicas acerca da velocidade do vento, empregouse un conxunto de observacións meteorolóxicas proporcionadas por [6]. Estes foron visualizados no Capítulo 1 e procesados de cara a ser empregados no noso modelo no Capítulo 3. Este conxunto foi dividido en dous subconxuntos: o conxunto de adestramento e o conxunto de proba.

O modelo foi creado coa linguaxe de programación Python, empregando a libraría *Keras* de *Tensorflow*.

Para avaliar a súa eficacia representaronse as funcións de custo do entrenamento e da proba. A visualización destas permitiunos comprobar que o noso modelo aprende correctamente e que non sofre problemas de sobreaxuste ou subaxuste.

Para o adestramento do noso modelo, dividiuse o noso conxunto de datos en intervalos iguais. Comparáronse os resultados das predicións e os erros do noso modelo para o caso de que os intervalos escollidos constasen de 60 observacións e para o caso de que ditos intervalos constasen de 365 observacións. En ambos casos se obtivo un erro cuadrático medio e unha raíz do erro cuadrático medio similares. Ademais, a comparación entre predicións realizadas e os datos esperados foi bastante boa nas dúas situacións. Porén, observamos que no caso no que os nosos intervalos tiñan 365 observacións o noso modelo se adestraba moito máis rápido. Pensamos que isto podía

deberse a unha periodicidade observada na visualización dos datos realizada no Capítulo 1. Concluímos finalmente, dado que en ambos casos as predicións son boas, que tanto para os intervalos de 60 observacións como os de 365, o noso modelo funciona correctamente. Polo que se conseguiu, aplicando as bases matemáticas introducidas no Capítulo 2, construír unha ferramenta capaz de predicir a velocidade do vento a partir de observacións climatolóxicas pasadas.

Anexo A

Código empregado nos capítulos 1 e 3

Recóllese neste anexo o código ao que se fai referencia nos capítulos 1 e 3.

```

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot
plt.style.use('fivethirtyeight')
import pandas as pd
import seaborn as sns
import sklearn
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow import keras
from keras import regularizers
from keras.models import Sequential
from keras.layers import Dense,Input, LSTM, Dropout
from keras.optimizers import SGD
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
#Lemos os datos
```

```
df=pd.read_csv('/Users/Sofia/weatherAUS.csv')
#Seleccionamos os datos que nos interesan
df=df.drop(df.index[:45588],axis=0)
df=df.head(3435)
df=df.drop(['Location','Evaporation','Sunshine','WindGustSpeed','WindGustDir','
    RainToday','RainTomorrow','Cloud9am','Cloud3pm'],axis=1)

df['Date']=pd.to_datetime(df['Date'])
#convirteo DATE nun obxecto pandas de tipo datatime, e dicir, datas e horas
df=df.set_index('Date')
#Establecense as datas como ndice do noso dataframe
```

```
    #Damos un valor numrico a cada direccin
def func(s):
    if s == "NNE":
        return 1
    elif s == "NE":
        return 2
    elif s == "ENE":
        return 3
    elif s == "E":
        return 4
    elif s == "ESE":
        return 5
    elif s == "SE":
        return 6
    elif s == "SSE":
        return 7
    elif s == "S":
        return 8
    elif s == "SSW":
        return 9
    elif s == "SW":
        return 10
    elif s == "WSW":
        return 11
    elif s == "W":
```

```
        return 12
    elif s == "WNW":
        return 13
    elif s == "NW":
        return 14
    elif s == "NNW":
        return 15
    else:
        return 16

df["WindDir3pm"] = df["WindDir3pm"].apply(func)
df["WindDir9am"] = df["WindDir9am"].apply(func)
```

```
#Eliminamos as filas nos que non estea recollidas mis de de 10 variables
df=df.dropna(thresh=10)
df=df.dropna(subset='Pressure3pm')
df=df.dropna(subset='Pressure9am')
df=df.fillna(0)
titles=['Temperatura mnima','Temperatura mxima','Chuvia',' Direccin do vento 9
    am',' Direccin do vento 3 pm',' Velocidade do vento 9 am',' Velocidade do
    vento 3 pm','Humidade as 9 am', 'Humidade as 3 pm','Presin as 9 am', 'Presin
     as 3pm',' Temperatura as 9 am',' Temperatura as 3 pm']
feature_keys=["MinTemp","MaxTemp","Rainfall","WindDir9am","WindDir3pm","
    WindSpeed9am","WindSpeed3pm","Humidity9am","Humidity3pm","Pressure9am","
    Pressure3pm","Temp9am","Temp3pm"]
```

```

# Grfica da evolucin das variables
values = df.values
groups = [2,7,8,9,10]
i = 1
# plot each column
pyplot.figure(figsize=(15,8))
for group in groups:
    pyplot.subplot(len(groups), 1, i)
    pyplot.plot(values[:, group])
    pyplot.title(df.columns[group], y=0.5, loc='right')
```

```
    i += 1
pyplot.show()

#mapa de calor
sns.heatmap(df.corr(),annot=True, cbar=True, cmap='Blues')
```

```
    #Establecemos os parmetros do modelo
split_fraction=0.75
train_split=int(split_fraction*int(df.shape[0]))

past=60
future=6
learning_rate=0.1
batch_size=32
epochs=250



#Seleccionamos as variables que non son redundantes
selected_features = [feature_keys[i] for i in [0, 2, 3, 4 ,5,6, 7, 8,9, 11]]
features = df[selected_features]
#Reescalamos estas variables
sc=MinMaxScaler(feature_range=(0,1))
features= sc.fit_transform(features)
features = pd.DataFrame(features)
train_data = features.loc[0 : train_split - 1]
val_data = features.loc[train_split:]
```

```
#Dividimos o noso conxunto de adestramento en intervalos iguais
start=past
end=start+train_split

x_train=train_data[[i for i in range(10)]].values
y_train=train_data.iloc[start:end][[4]]

sequence_length=past
dataset_train=keras.preprocessing.timeseries_dataset_from_array(x_train,y_train,
```

```
    sequence_length=sequence_length,shuffle=False, batch_size=batch_size)


#Dividimos o noso conxunto de proba en intervalos iguais
x_end=len(val_data)-past-future
label_start=past

x_val=val_data.iloc[:x_end][[i for i in range(10)]].values
y_val=val_data.iloc[label_start:][4]

dataset_val=keras.preprocessing.timeseries_dataset_from_array(x_val,y_val,
    sequence_length=sequence_length, batch_size=batch_size)

for batch in dataset_train.take(1):
    inputs, targets = batch

print("Input shape:", inputs.numpy().shape)
print("Targets shape:", targets.numpy().shape)
```

```
    #Construcin do modelo
inputs = keras.layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
lstm_out = keras.layers.LSTM(32)(inputs)
outputs = keras.layers.Dense(1,kernel_regularizer=regularizers.l2(0.1))(lstm_out
    )

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer=tf.keras.optimizers.experimental.SGD(learning_rate=
    learning_rate), loss="mse")
```

```
    #adestramento do modelo
path_checkpoint = "model_checkpoint.h5"
es_callback = keras.callbacks.EarlyStopping(monitor="val_loss", min_delta=0,
    patience=5)

modelckpt_callback = keras.callbacks.ModelCheckpoint(
    monitor="val_loss",
    filepath=path_checkpoint,
```

```
    verbose=1,
    save_weights_only=True,
    save_best_only=True,
)

history = model.fit(
    dataset_train,
    epochs=epochs,
    validation_data=dataset_val,
    callbacks=[es_callback, modelckpt_callback]
)
```

```
    def visualize_loss(history, title):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="custo do adestramento")
    plt.plot(epochs, val_loss, "r", label="custo da validacin")
    plt.title(title)
    plt.xlabel("Iteracins")
    plt.ylabel("custo")
    plt.legend()
    plt.show()


visualize_loss(history, "custo do adestramento e da validacin")
```

```
    def show_plot(plot_data, delta, title):
    labels = ["Evolucin dos datos", "Valor esperado", "predicin do modelo"]
    marker = [".-", "rx", "go"]
    time_steps = list(range(-(plot_data[0].shape[0]), 0))
    if delta:
        future = delta
    else:
        future = 0

```

```
    plt.title(title)
    for i, val in enumerate(plot_data):
        if i:
            plt.plot(future, plot_data[i], marker[i], markersize=10, label=labels
                [i])
        else:
            plt.plot(time_steps, plot_data[i].flatten(), marker[i], label=labels[
                i])
    plt.legend()
    plt.xlim([time_steps[0], (future + 5) * 2])
    plt.xlabel("Paso")
    plt.show()
    return


for x, y in dataset_val.take(5):
    show_plot(
        [x[0][:, 4].numpy(), y[0].numpy(), model.predict(x)[4]],
        12,
        "Prediccin",
    )
```

Bibliografía

[1] Shalev-Shwartz, S., Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Reino Unido: Cambridge University Press.

[2] Viaño Rey, J. M., Burguera González, M. (2013). *Lecciones de métodos numéricos*. España: Andavira.

[3] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Estados Unidos: Determination Press.

[4] Neidinger, R. D. (2010). *Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming*. SIAM Review, 52(3), 545–563.

[5] Rodríguez, G. (2003). *Diferenciación de funciones de varias variables reales*. España: Universidade de Santiago de Compostela, Servicio de Publicacións e Intercambio Científico.

[6] Bureau of Meteorology. (2022). *Climate Data Online*. Bom.gov.au. http://www.bom.gov.au/climate/data/

[7] Madhu Sanjeevi . *"Chapter 10.1: DeepNLP — LSTM (Long Short Term Memory) Networks with Math."* Medium, 21 Jan. 2018, medium.com/deep-math-machine-learning-ai/chapter-10-1-deepnlp-lstm-long-short-term-memory-networks-with-math-21477f8e4235.

[8] Team, K. (n.d.). *Keras documentation: Timeseries forecasting for weather prediction*. Keras.io. https://keras.io/examples/timeseries/timeseries_weather_forecasting/

[9] *Wind Direction and Degrees*. (s/f). http://snowfence.umn.edu/Components/winddirectionanddegrees.htm

[10] Jacques, M. (2020). *Batch vs mini-batch vs stochastic Gradient Descent with code examples*. DataDrivenInvestor. https://medium.datadriveninvestor.com/batch-vs-mini-batch-vs-stochastic-gradient-descent-with-code-examples-cd8232174e14