

PRÀCTICA 2

INTENT RECONITION NAME ENTITY RECOGNITION



Paula Justo i Júlia Pedrol

3r GIA

Tractament de la Veu i el Diàleg

13 - 11 - 25

1. IMPLEMENTACIÓ REALITZADA PART I

En aquesta primera part de la pràctica vam preparar el conjunt de dades i vam desenvolupar un model base de classificació d'intencions. El treball va consistir a preprocessar les oracions, convertir-les en seqüències numèriques i entrenar una xarxa neuronal senzilla per identificar la intenció de cada frase. Aquesta fase ens va servir per establir la base tècnica sobre la qual després vam provar models més complexos.

1.1. PREPARACIÓ DEL DATASET

L'objectiu d'aquest primer exercici era explorar i entendre el conjunt de dades que hem utilitzat al llarg de la pràctica (data, dividida en train, validation i test).

Hem carregat tots tres fitxers (train.csv, val.csv i test.csv) amb la llibreria de pandas, i mitjançant aquesta hem pogut veure la mida d'aquesta partició del dataset (4078 frases per entrenament, 900 per validació i 893 per test).

Per aquesta primera pràctica, ens vam centrar únicament en dues de les columnes del fixter:

- La primera columna: conté les oracions en anglès introduïdes pels usuaris.
- La tercera columna: conté la intenció associada a cada oració (etiqueta que volem predir).

A continuació, vam extreure i mostra una oració aleatòria amb la seva intenció corresponent, epr així poder veure un exemple real del conjunt de dades i entendre millor les dades que tractarem.

Finalment, vam comptar les etiquetes d'intenció i vam veure que el conjunt de dades conté 22 categories. Seguidament, vam iniciar el preprocessament de les dades mitjançant la tokenització de les oracions. Vam convertir cada paraula del text en un enter per tal de crear un vocabulari més fàcil d'interpretar. Per fer això, vam utilitzar el tokenizer Keras, ens permet dividir es frases en paraules i assignar un ID a cada una. En reusm, aquest primer exercici ens va permetre familiaritzar-nos amb el dataset, entendr el tipus de dades amb les que haurem de treballar i identificar les possibles classes o intencions que ha d'aprendre a reconéixer el nostre model.

1.2. EXERCICI 1

En aquest exercici vam convertir les oracions en seqüències numèriques utilitzant el vocabulari creat anteriorment, de manera que cada paraula es representa pel seu identificador (ID).

Després, vam normalitzar la longitud de les seqüències aplicant padding, és a dir, afegint zeros al principi de les frases més curtes perquè totes tinguessin la mateixa mida. Això permet que el model pugui processar-les de manera uniforme durant l'entrenament.

1.3. EXERCICI 2

En aquest exercici vam ajustar la preparació de les seqüències perquè el padding es realitzes al final de la frase. Com que en els models seqüencials l'ordre de les paraules és rellevant, aquesta decisió preserva intacte l'inici de cada oració i evita desplaçar la informació cap a l'esquerra. Quan la longitud supera el màxim fixat, vam optar també per retallar al final i així mantenir la coherència.

A continuació vam transformar les etiquetes d'intenció: primer les vam codificar a valors numèrics amb LabelEncoder i després les vam convertir a vectors one-hot, de manera que cada exemple queda representat per un vector binari amb un únic 1 a la posició de la seva classe. Aquesta codificació ens permet entrenar un model amb sortida softmax de mida num_classes i pèrdua categorical_crossentropy. Finalment, vam conservar l'objecte label_encoder per poder recuperar els noms de les classes en fase de predicción i vam fixar num_classes com a paràmetre clau de la capa de sortida.

1.4. EXERCICI 3

En aquest tercer exercici vam repetir el mateix procés de preprocessament que ja havíem aplicat anteriorment en el conjunt de train, però en aquest cas, l'hem aplicat sobre els conjunts val i test.

Inicialment, vam convertir les oracions en seqüències numèriques amb el mateix tokenizer utilitzat anteriorment, per així assegurar-nos que tots els paraules es codifiquen amb els mateixos identificadors. Després vam aplicar padding al final de la frase, per tal que tots els seqüències tinguessin la mateixa longitud que les del train. Seguidament vam netejar algunes etiquetes no desitjades o combinacions pot representatives.

Per últim, vam codificar les etiquetes de validació i test utilitzant el mateix LabelEncoder entrenat amb les dades del train, per tal de garantir que totes les pàtions del conjunt tinguessin la mateixa correspondència entre classes i valors numèrics.

1.5. EXERCICI 4

Aquest exercici consistia en definir i entrenar un model base per la classificació d'intencions. L'arquitectura final d'aquest constava de quatre capes:

1. **Capa d'Embedding:** per transformar els índex de paraules en vectors de dimensió fixa.
2. **Capa de Global Max Polling:** per resumir la informació de tota la seqüència en un sol vector rebus als desplaçaments.
3. **Capa Densa:** amb activació ReLU per capturar patrons no lineals.
4. **Capa de Sortida:** amb SoftMax de dimensió num_classes, la qual ens retorna la probabilitat de cada intenció.

!! Vam mantenir el padding al final de les frases per tal de presentar l'ordre natural de la informació.

Un cop definit aquets model, per entrenar-lo, vam utilitzar Adam i Categorical CrossEntropy i com a creteris d'avaluació, per tal de poder comprovar el rendiment, vam fer servir la accuracy dels conjunts de validació i test. Finalment, vam avaluar el model base i vam obtenir resultats sòlids ($\approx 0,95$ en validació i $\approx 0,93$ en test), els quals ens serveixen com a punt de partida per seguir experimentant amb aquests.

Per què vam definir un model amb aquesta arquitectura?

- **Embeddings:** redueix l'esparsitat i capta relacions semàntiques entre paraules en un espai continu, factor que facilita l'entrenament i millora la capacitat de generalització.
- **Global Max Pooling:** extreu els senyals més informatius de la seqüència i evita el cost d'operar estats temporals llargs.
- **Densa + ReLU:** aporta no linealitat i capacitat de combinar característiques.
- **Softmax:** transforma els logit en probabilitats normalitzades, les quals són necessàries per la classificació multiclasse.

Tot i que vam veure que el model havia assolit una bona precisió global, vam decidir realitzar un brue anàlisis dels errors per tal de veure aquelles frases que presentaven confusions o estaven mal etiquetades i, per tant, el model no ha estat capaç de classificar correctament la seva intenció.

Per exemple, en la frase:

"I need a flight from Tampa to Milwaukee" (real: *meal*, predict: *flight*)

El model reconeix el patró *"I need a flight from X to Y"*, molt habitual a les frase de tipus flight i ignora que, m dins del conjunt de dades, aquesta expressió també pot aparèixer en oracions relacionades amb meal (quan s'hi fa referència al servei a bord o als àpats dels vols).

Això mostra que el model ha après patrons superficials basats en paraules freqüents i no enten el context semàntic de la frase. Un possible motiu també podria ser la manca de mostres d'entrenament de la classe meal.

Una possible millora per tal de reduir aquesta manca de contex semàntic, seria reajustar els pesos de classes o utilitzar embeddings contextials que permetin capturar millor el significat complet de la frase.

En resum, aquest exercici ens ha permès dissenyar, entrenar i avaluar un model base de classificació d'intencions amb ua arquitectura senzilla però efectiva. Els resultats obtinguts demostren que el model és capaç de capturar patrons generals del llenguatge i fer una bona distinció entre la majoria d'intencions.

Tot i això, l'anàlisis dels errors ens ha mostrar que tot i els bons resultats, el model encara confon classes semàntiques similars i no capta prou bé el context de certes frases, especialment en categories amb poques mostres.

1.5. EXERCICI 5

Pel que fa a aquest següent exercici, durant aquest vam dissenyar diverses proves sobre el nostre model per tal d'analitzar com diferents decisions d'arquitectura i preprocessament afecten a l'accuracy del nostre classificador d'intencions. La finalitat principal és passar d'un model base senzill a un conjunt de models comparables, aïllant l'efecte de cada canvi i justificant-ho tot.

Per implementar aquest, vam mantenir el mateix dataset i pipeline de preparació (padding al final de la seqüència i mateix esquema d'etiquetes) per així poder garantir comparabilitat. A més a més, vam utilitzar early stopping i un registre sistemàtic de mètodes per detectar overfitting i poder avaluar la generalització i rendiment del model. A partir d'aquí, vam explorar sis línies de treball:

- **Preprocessament del text:** variació de la mida de vocabulari i transformacions (minuscúl·lules, lematització/stemming, etc.) per estudiar l'efecte sobre el soroll i la coberta lèxica.
- **Mida dels embeddings:** prova de diverses dimensions per observar el compromís entre capacitat representacional i risc d'overfitting.
- **Xarxes convolucionals (CNN):** incorporació de convolucions i diferents poolings per capturar patrons locals n-gram i comparar-los amb el model base.
- **Xarxes recurrents:** evaluació de dependències seqüencials llargues i el seu impacte en l'accuracy.
- **Regularització:** ús de Dropout (i ubicació dins la xarxa) per controlar l'overfitting quan augmenten els paràmetres.
- **Balanceig de classes:** càcul de class weights i integració a fit(...) per mitigar el desbalanceig i millorar el rendiment en classes minoritàries.

Mitjançant aquesta estratègia podem comprar objectivament les diferents configuracions implementades, extreure conclusions concretes, **detectar millores reals** i construir un model final robust, capaç de generalitzar millor a noves dades. A continuació, detallarem cadascun dels apartats i els resultats obtinguts.

1.5.1. PREPROCESSAMENT

Inicialment, vam crear un procés complet de preparació i neteja del text amb l'objectiu d'obtenir seqüències coherents i comparables entre models. Primer vam normalitzar totes les frases, convertint-les a minúscules per unificar formes (“Flight” i “flight”), eliminant accents per reduir variacions ortogràfiques i

substituint els noms per <num>, de manera que el model pogués captar el significat d'aparició d'un número sense memoritzar-ne el valor concret. També vam netejar la puntuació innecessària, mantenint només els elements rellevants per al significat.

Després vam entrenar un tokenizer sobre el conjunt d'entrenament per transformar les frases en seqüències numèriques. Hi vam incloure un token especial <OOV> per representar paraules desconegudes i vam limitar el vocabulari per controlar la complexitat del model. Les seqüències es van truncar i emplenar (padding) al final, tal i com havíem fet fins ara, fixant la seva longitud màxima segons el 95% de la longitud de les frases, per mantenir la majoria d'informació i alhora evitar seqüències massa llargues i poc útils.

A partir d'aquest esquema, vam provar quatre configuracions diferents per veure quin preprocessament donava millors resultats:

- **Baseline:** vocabulari de 2.000 paraules i normalització bàsica.
- **Vocab5000:** igual que l'anterior però amb vocabulari de 5.000 paraules.
- **Stemming:** amb arrels lèxiques per reduir variacions morfològiques.
- **Case-sensitive:** mantenint majúscules i accents, i fent servir la longitud màxima total.

Per tal de realitzar una comparació objectiva sobre aquests configuracions, vam entrenar-les tots amb el mateix model base per així mantenir condicions iguals en totes elles. Totes les configuracions es van entrenar amb el mateix model base per comparar-les en condicions iguals. El millor resultat es va obtenir amb cfg_vocab5000, que va assolir una val_accuracy de 0.9544 i una test_accuracy de 0.9216. Aquesta configuració combina una bona cobertura lèxica amb una longitud eficient (maxlen = 19).

Finalment, vam mantenir la configuració cfg_stemming com a base per a tots els experiments següents.

1.5.2. MIDA DELS EMBEDDINGS

En aquest apartat vam voler analitzar com la dimensió dels vectors d'embedding influïa en el rendiment del model. Per fer-ho, vam mantenir el mateix preprocessament òptim obtingut a l'apartat anterior i la mateixa arquitectura base, de manera que l'únic element variable fos la mida dels embeddings.

Vam entrenar diversos models amb dimensions de 32, 64, 128, 256 i 384, tots amb la mateixa estructura i les mateixes condicions d'entrenament:

$$\text{Embedding} \rightarrow \text{GlobalMaxPooling} \rightarrow \text{Dense} \rightarrow \text{Dropout} \rightarrow \text{Softmax}$$

Aquesta estratègia ens va permetre comparar de forma justa l'efecte de cada mida sobre l'accuracy de validació i de test.

L'objectiu era observar el compromís entre capacitat representacional i generalització, ja que embeddings massa petits poden perdre informació semàntica, mentre que embeddings massa grans augmenten el nombre de paràmetres i el risc d'overfitting.

Els resultats mostren clarament aquest equilibri. Les dimensions 384 i 256 ofereixen el millor rendiment, amb una val_accuracy del 0.9511 i 0.9444 respectivament, i test_accuracy del 0.8970 i 0.9082. Les mides petites (32 i 64) redueixen notablement la capacitat del model, amb accuracy inferior al 0.85.

Degut a aquests resultats, vam seleccionar la configuració amb embeddings de 384 dimensions com la millor opció per continuar els experiments següents (xarxa convolucional i recurrent), ja que ofereix el millor balanç entre rendiment i eficiència computacional.

1.5.3. XARXES CONVOLUCIONALS

En aquest apartat vam introduir una xarxa convolucional 1D per analitzar com aquest tipus d'arquitectura pot millorar la detecció de patrons locals al text, utilitzant el mateix preprocessament i embeddings de 384 dimensions obtinguts prèviament.

L'objectiu era capturar relacions entre paraules properes de manera més eficient que amb el model base, mantenint un equilibri entre complexitat i capacitat de generalització.

Vam provar diferents configuracions variades en profunditat, mida de filtre i tipus de pooling. Entre elles, models amb una sola capa convolucional amb filters de mida 5 i pooling màxim (per capturar els patrons més rellevants), versions amb pooling mitjà, i dissenys amb dues convolucions apilades que afegeixen més profunditat a la xarxa. També vam provar una arquitectura amb diversos filters en paral·lel de mides 3, 4 i 5, capaços de detectar patrons de diferent llargada dins de la mateixa frase.

Totes les xarxes segueixen la mateixa estructura i es van entrenar amb el mateix optimitzador:

*Embedding → capes convolucionals → pooling global → capa densa amb ReLU →
Dropout → Sortida Softmax*

Els resultats mostren que la configuració cnn_max_k5_f128 va ser la que va aconseguir el millor rendiment global (Val_Acc = 0.9444 i Test_Acc = 0.9306). Aquesta arquitectura, basada en una sola capa convolucional amb filters de mida 5 i pooling màxim, demostra una gran capacitat per captar patrons locals rellevants i generalitzar bé amb un nombre moderat de paràmetres.

Tot i que altres variants amb filtres múltiples o empilats també van obtenir bons resultats, aquestes van incrementar el nombre de paràmetres sense millorar significativament la precisió. Per tant, el model cnn_max_k5_f128 s'ha consolidat com la millor opció per l'equilibri entre rendiment i eficiència.

1.5.4. XARXES RECURRENTS

En aquest apartat vam explorar l'ús de xarxes neuronals recurrents per captar dependències llargues dins de les frases, un aspecte clau per entendre el context complet d'una intenció. Es van provar diferents variants basades en LSTM, GRU i BiLSTM, mantenint el mateix preprocessament i embeddings de 384 dimensions que en els experiments anteriors, així com *early stopping* i *class weights* per evitar overfitting i desbalanceig de classes.

El millor rendiment es va aconseguir amb la configuració bilstm_last_u64, que combina una capa Bidirectional LSTM amb 64 unitats i processament en ambdues direccions. Aquesta arquitectura va assolir una val_accuracy del 0.8744 i una test_accuracy del 0.8970, mostrant una bona capacitat de generalització i millor rendiment que les versions unidireccionals o més profunes.

Les altres variants (LSTM, GRU i BiLSTM apilada) van obtenir resultats lleugerament inferiors, amb diferències modestes però sense millorar la precisió final.

1.5.5. REGULARITZACIÓ

En aquest apartat vam analitzar l'efecte de la regularització amb Dropout tant en xarxes convolucionals com en recurrents, amb l'objectiu de reduir el overfitting detectat en alguns models i millorar la seva capacitat de generalització.

→ CNN amb SpatialDropout1D

En el cas de la CNN, es van combinar diferents valors de SpatialDropout1D (aplicat a l'entrada de l'embedding) i de Dropout a la capa densa final. Els millors resultats es van obtenir amb la configuració spDrop=0.2 i denseDrop=0.0, que va assolir una val_accuracy de 0.9722 i una test_accuracy de 0.9418, millorant significativament la robustesa del model sense augmentar la complexitat.

Altres configuracions amb Dropout més alt no van aportar millores, ja que reduïen lleugerament la precisió en test a causa d'una pèrdua d'informació excessiva durant l'entrenament.

→ LSTM amb Dropout i Recurrent Dropout

Per a la xarxa recurrent, es van provar combinacions de Dropout (a la capa densa) i Recurrent Dropout (dins la pròpia LSTM). La millor configuració va ser recDrop=0.0 i denseDrop=0.3, amb una val_accuracy de 0.9667 i test_accuracy de 0.9205. Aquesta combinació aconsegueix un bon equilibri entre estabilitat i rendiment, mentre que valors més elevats de recurrent dropout van penalitzar la memòria temporal del model.

→ Comparativa i model final

En comparar ambdós models, la CNN regularitzada va mostrar un rendiment lleugerament superior al de la LSTM (Val_Acc = 0.9722, Test_Acc = 0.9418, vs 0.9667 / 0.9205 respectivament), mantenint un nombre de paràmetres similar.

Així doncs, el model best_reg_cnn_dropout.keras es va seleccionar com a model final per la seva millor capacitat de generalització i eficiència computacional.

1.5.6. BALANCEIG DE CLASSES

En l'últim apartat vam avaluar tres estratègies d'entrenament sobre el mateix model base per mitigar el desbalanceig de classes: entrenament bàsic (sense pesatge), class weights estrictes i oversampling naïf de les classes minoritàries. Es manté el mateix preprocessament i *early stopping* per garantir comparabilitat.

Millors resultats per estratègia:

- Oversampling → Val_Acc = 0.9733, Test_Acc = 0.9619 (*millor test*)
- Class Weight → Val_Acc = 0.9744, Test_Acc = 0.9563
- Bàsic (sense balanceig) → Val_Acc = 0.9767, Test_Acc = 0.9496

Tot i que el mode bàsic pot obtenir una val_acc lleugerament superior, els mètodes de balanceig milloren la generalització: l'oversampling és el que més puja l'*accuracy* de test, mentre que el class weight també ajuda però queda per darrere. Això és coherent amb la distribució fortament esbiaixada.

Per tant, per com a model final, seleccionem best_model_oversampling.keras perquè ofereix el millor compromís entre rendiment global i robustesa en classes minoritàries.

2. IMPLEMENTACIÓ REALITZADA PART II

En aquesta segona part de la pràctica hem treballat el reconeixement d'entitats anomenades (NER), una tasca fonamental del processament del llenguatge natural que consisteix en identificar i classificar entitats concretes dins d'un text, com ara ciutats, dates o tipus de bitllets. A diferència de la primera part, on el model havia de predir la intenció global d'una oració, en aquesta segona part el nostre objectiu és assignar una etiqueta a cada paraula segons la seva categoria semàntica.

Aquesta tasca és especialment rellevant en aplicacions com els sistemes de reserves o els assistents virtuals, on cal reconèixer informació específica dins de la frase d'un usuari (per exemple, detectar la ciutat de sortida o la data del vol). El conjunt de dades utilitzat és el mateix que a la primera part, però ara fem servir una columna addicional amb les etiquetes BIO, que indiquen si una paraula és l'inici (B), l'interior (I), final (L), única (U) o fora (O) d'una entitat.

La metodologia d'aquesta segona part és similar a la de la primera: primer inspeccionem les dades i les preparam per poder entrenar posteriorment el model seqüencial que aprendrà a reconèixer aquestes entitats.

2.1. EXERCICI 1

En aquest primer exercici hem carregat els fitxers train.csv, val.csv i test.csv amb pandas per analitzar la mida i l'estructura del conjunt de dades: 4078 frases per entrenament, 900 per validació i 893 per test.

Cada oració ve acompañada de les seves etiquetes BIO, que indiquen si cada paraula és l'inici (B), l'interior (I), final (L), única (U) o fora (O) d'una entitat com una ciutat, una data o un tipus de bitllet. També s'inclou la intenció general de la frase.

Aquesta exploració inicial ens ha permès entendre com estan organitzades les etiquetes i familiaritzar-nos amb el format BIO, pas imprescindible abans de preparar les dades per a l'entrenament del model de reconeixement d'entitats.

2.2. EXERCICI 2

En aquest pas ens vam centrar en treballar amb les dues primeres columnes del dataset: les frases en anglès i les seves etiquetes BIO corresponents. L'objectiu era construir les llistes de frases i etiquetes per a les particions de train, validació i test, i assegurar-nos que el format fos correcte. Després d'extreure les columnes, vam netejar les etiquetes eliminant cometes sobrants i vam confirmar la mida dels conjunts

(4078 frases per entrenament, 900 per validació i 893 per test). Per comprovar que cada paraula estava correctament alineada amb la seva etiqueta, vam revisar alguns exemples on es podia observar la correspondència entre tokens i entitats, com ciutats o hores de vol. Finalment, vam comptar un total de 119 etiquetes úniques en format BIO, fet que ens confirma que el conjunt de dades està preparat per iniciar la tokenització i la conversió a índexs per entrenar el model NER.

2.3. EXERCICI 3

En aquest apartat hem preparat les dades, mitjançant un preprocessament bàsic, per tal que el model NER les pugui llegir de forma uniforme. Primer hem construït el vocabulari només amb les frases de train mitjançant un Tokenizer que manté la caixa baixa i no filtra signes, i després hem convertit totes les oracions a seqüències d'índexs amb aquest mateix tokenitzador per a train, validació i test. Tot seguit hem guardat la longitud original de cada oració (sense zeros) perquè, a l'avaluació, puguem calcular mètriques sense comptar el padding. Finalment, hem unificat la mida de totes les seqüències aplicant padding al final fins a la longitud màxima observada al train; això preserva l'inici de la frase i garanteix que totes tinguin la mateixa forma d'entrada.

Resultat del procés: vocabulari de 831 paraules, maxlen = 46, i tensors amb dimensions train (4078, 46), val (900, 46) i test (893, 46). Aquesta base ja està llena per codificar etiquetes per paraula i entrenar el model seqüencial.

2.4. EXERCICI 4

En aquest exercici hem preparat les etiquetes del conjunt de dades per tal que el model pugui aprendre a identificar correctament cada tipus d'entitat. Inicialment, vam comptar totes les classes diferents presents al conjunt d'entrenament, considerant per separat els prefixos B-, I-, L-, U- i O, i afegint una etiqueta especial <pad> per marcar l'emplenament de les seqüències. En total es van identificar 120 etiquetes úniques, incloent-hi la classe “fora d'entitat” (O).

A continuació, vam convertir les etiquetes de text en valors numèrics mitjançant un LabelEncoder, assegurant que cada etiqueta tingués el seu propi identificador. Per garantir la coherència, també vam eliminar de validació i test aquelles oracions que contenien etiquetes no vistes durant l'entrenament.

Un cop codificades, vam aplicar padding al final de les seqüències per tal que totes tinguessin la mateixa longitud, utilitzant l'etiqueta <pad> com a emplenamet per mantenir igual longitud en tots els casos.

Finalment, aquestes etiquetes numèriques es van transformar en vectors one-hot, que és el format que utilitzarà la capa de sortida del model per aprendre a predir una classe per cada paraula.

Aquest procés ens deixa les dades completament preparades: cada frase té la seva seqüència de tokens i d'etiquetes alineades, amb la mateixa mida i llestes per ser introduïdes al model NER.

2.5. EXERCICI 5

En aquest exercici vam dur a terme l'entrenament del model seqüencial per a NER. Vam mantenir el preprocessament i els embeddings de 128 dimensions, i vam començar a dissenyar una arquitectura senzilla però efectiva orientada a predir una etiqueta per token.

Pel que fa a l'arquitectura implementada, el model comença amb una Embedding (mask_zero=True) per marcar el padding. A continuació incorporem dues capes Conv1D (128 i 64 filtres, kernel 3, padding='same'), separades per Dropout(0.2). Les convolucions ens permeten capturar patrons locals tipus n-gram i, combinades, permeten detectar patrons una mica més llargs sense elevar massa el model. Finalment, incorporem una capa densa amb Softmax que genera la probabilitat de cadascuna de les 120 classes en cada pas temporal. Entrenem amb Adam i categorical_crossentropy, batch size 32 i 10 èpoques.

És important destacar que en aquest model definit, Conv1D no propaga el masking intern; per això no fem servir el padding per entrenar la mètrica. En l'avaluació eliminem els tokens <pad> amb les longituds originals i calculem les mètriques només sobre paraules reals.

Pel que fa a l'avaluació obtinguda, com que la classe O és molt majoritària, el valor de l'accuracy pot resultat confús i enganyós. Per això reportem F1 macro (mitjana per classe) i el classification_report "sense padding". En el nostre experiment, el model obté aproximadament:

- F1 macro ≈ 0.85 (prec. ≈ 0.88 , recall ≈ 0.82)
- F1 ponderat ≈ 0.95

L'accuracy global és alta, però no la fem servir com a mètrica principal per l'efecte de la classe O.

Gràcies als resultats obtinguts, podem conoure que el model reconeix bé les entitats freqüents (ciutats, dies, períodes del dia) i mostra més dificultat en categories poc representades. Les convolucions ofereixen un bon compromís entre cost i rendiment; tot i així, si es volgués capturar dependències llargues, una (Bi)LSTM o un Transformer per substitució del pooling podria millorar el recall en entitats llargues. També serien naturals millors com class weights per token, CRF a la sortida o embeddings contextuais.

En resum, hem definit i entrenat un model per-token coherent amb el pipeline preparat, l'hem avaluat ignorant el padding i hem reportat F1 macro com a mètrica central, obtenint un rendiment sòlid i equilibrat per continuar iterant.

2.6. EXERCICI 6

2.6.1. MIDA DELS EMBEDDINGS

En aquest primer apartat de l'exercici 6 vam analitzar com la dimensió dels vectors d'embedding influeix en el rendiment del nostre model de reconeixement d'entitats. Per fer-ho, vam mantenir exactament la mateixa arquitectura utilitzada a l'exercici anterior (una xarxa seqüencial amb una capa Embedding, una capa Dropout(0.2), una capa Dense(64, relu) i una sortida Softmax amb tantes neurones com etiquetes diferents), variant únicament la mida de l'embedding.

Vam provar cinc configuracions diferents: 32, 64, 128, 256 i 384 dimensions, utilitzant early stopping per evitar sobreentrenament i el mateix conjunt de dades preprocessat per garantir una comparació justa. A cada experiment vam calcular les mètriques sense tenir en compte el padding, centrant-nos principalment en la F1 macro, que és més representativa en conjunts de dades tan desbalancejats com aquest, i també en la F1 ponderada per tenir una visió global del comportament del model.

Els resultats obtinguts van ser els següents:

Mida d'Embedding	Accuracy (no padding)	Macro F1	Weighted F1
32	0.8339	0.3043	0.8172
64	0.8364	0.3112	0.8172
128	0.8354	0.3365	0.8195
256	0.8418	0.3773	0.8286
384	0.8359	0.3520	0.8245

Com podem observar, el rendiment millora progressivament fins a 256 dimensions, on el model aconsegueix els millors valors de F1 macro (0.3773) i F1 ponderada (0.8286). A partir d'aquí, amb 384 dimensions, el rendiment deixa de millorar i fins i tot disminueix lleugerament, cosa que indica que una major capacitat no sempre implica una millor generalització.

Això ens mostra que, en aquest cas, 256 dim representa el punt òptim entre riquesa semàntica i eficiència computacional. Valors més petits limiten la capacitat del model per capturar les relacions entre paraules, mentre que valors massa grans augmenten el risc d'overfitting sense oferir beneficis significatius.

2.6.2. XARXES CONVOLUCIONALS

En aquest apartat vam introduir xarxes convolucionals (CNN) per millorar el rendiment del model de reconeixement d'entitats, mantenint la mida d'embedding de 256 (escollida al pas anterior com la més òptima). L'objectiu era comprovar com diferents configuracions de filters i mides de nucli afectaven la capacitat del model per captar patrons locals dins les oracions.

Per fer-ho, vam definir quatre arquitectures diferents:

- **conv3:** una capa Conv1D amb 128 filters i nucli de mida 3.
- **conv5:** una capa Conv1D amb 128 filters i nucli de mida 5.
- **conv7:** una capa Conv1D amb 128 filters i nucli de mida 7.
- **stacked_3_5:** dues capes convolucionals apilades, amb 128 i 64 filters, i nuclis de mida 3 i 5.

Totes les configuracions utilitzaven padding='same' per mantenir la longitud de seqüència, activació ReLU i un dropout del 30% per reduir el sobreajust. La sortida final, com en el model base, era una capa Dense amb activació softmax que prediu una etiqueta per a cada token.

Cada configuració es va entrenar amb Adam i early stopping, i l'avaluació es va fer sense comptar els tokens de padding, calculant accuracy, F1 macro i F1 ponderada.

Els resultats finals van ser els següents:

Model	Accuracy (sense padding)	Macro F1	Weighted F1
conv3	0.9411	0.4941	0.9311
conv5	0.9572	0.5960	0.9520
conv7	0.9532	0.5804	0.9463
stacked_3_5	0.9503	0.5268	0.9436

El millor resultat es va obtenir amb la configuració conv5, que va assolir una F1 ponderada de 0.9520 i una F1 macro de 0.5960, superant clarament la resta d'opcions. Aquest resultat mostra que una finestra de context de mida mitjana (kernel=5) és suficient per capturar patrons locals rellevants, com entitats formades per dues o tres paraules consecutives, sense perdre informació contextual.

Les convolucions amb nuclis més petits (3) capten patrons molt breus i no aconsegueixen generalitzar prou bé, mentre que nuclis més grans (7) tendeixen a diluir el senyal i a incrementar el soroll. L'estructura apilada (stacked_3_5), tot i ser més profunda, no va aportar millores significatives, possiblement per la simplicitat del conjunt de dades i la seva curta longitud de seqüència.

En conclusió, les xarxes convolucionals milloren notablement el rendiment respecte al model base. El model CNN amb kernel=5 es consolida com la millor opció dins d'aquest apartat, equilibrant precisió, velocitat i capacitat de generalització.

2.6.3. XARXES RECURRENTS

Per capturar dependències llargues entre tokens, vam incorporar RNN bidireccionals sobre les representacions que ja extreu la millor CNN anterior. Concretament, vam congelar la part d'entrada (Embedding 256 + Conv1D k=5 + Dropout) i en vam utilitzar la sortida com a seqüència de característiques per alimentar diferents variants BiLSTM/GRU amb sortida per token.

Vam definir un disseny comú per tal de definir la implementació de les RNN, vam imposar SpatialDropout1D(0.3) sobre les seqüències de característiques, 1–2 capes bidireccionals (LSTM o GRU, 128 o 256 unitats, return_sequences=True, dropout=0.3), LayerNormalization, i TimeDistributed(Dense Softmax) per predir una etiqueta a cada posició. Optimitzador Adam (LR adaptat per configuració), early stopping i evaluació sense padding amb F1 macro (mètrica principal) i F1 ponderada.

Vam provar:

- **1 vs 2 capes:** més profunditat per capturar patrons composicionals.
- **128 vs 256 unitats:** més capacitat per a dependències llargues.
- **LSTM vs GRU:** comprovar si GRU aconsegueix resultats similars amb menys complexitat.

Els resultats finals van ser els següents:

- **BiGRU (256, 2 capes) — Weighted F1 = 0.9584 | Macro F1 = 0.6427 | Acc = 0.9617 → millor.**
- BiLSTM (256, 2 capes) — Weighted F1 = 0.9524 | Macro F1 = 0.5564.
- BiGRU (128, 2 capes) — Weighted F1 = 0.9521 | Macro F1 = 0.5571.
- BiLSTM (128, 2 capes) — Weighted F1 = 0.9498 | Macro F1 = 0.5383.
- BiLSTM (128, 1 capa) — Weighted F1 = 0.9450 | Macro F1 = 0.5153.

Mitjançant aquests resultats podem comprovar que afegir recurrència millora clarament respecte a la CNN pura (de Macro F1 0.596 → 0.643), sobretot en entitats minoritàries (on la memòria seqüencial ajuda el recall). Les configuracions de 2 capes superen les d'1 capa i 256 unitats superen 128 de forma consistent. Entre cel·les, GRU amb 256×2 ha estat lleugerament millor i més eficient que l'LSTM equivalent, oferint el millor compromís capacitat ↔ estabilitat.

Davant d'aquest resultats, ens quedem amb BiGRU 2×256 com a base recurrent: és la que generalitza millor i eleva la F1 macro sense un cost excessiu, consolidant-se com a pas endavant sobre la CNN guanyadora. Model desat: best_rnn_bigru256_2layer.keras.

2.6.4. TRANSFORMER

En aquest apartat vam explorar l'ús d'una arquitectura Transformer aplicada sobre les característiques prèviament extretes pels nostres millors models convolucionals i recurrents.

L'objectiu d'aquest experiment era analitzar fins a quin punt els mecanismes d'atenció del Transformer podien millorar la representació seqüencial del llenguatge i capturar dependències de llarg abast de manera més eficient que els models anteriors.

Per tal d'implementar aquest model, primer vam utilitzar la CNN com a extractor de característiques locals i la RNN com a integrador de context temporal, obtenint així vectors seqüencials més rics i contextualitzats.

Aquestes representacions van ser posteriorment utilitzades com a entrada per a la nova arquitectura Transformer, dissenyada amb diferents combinacions de paràmetres per estudiar-ne l'impacte sobre el rendiment final.

Es van provar quatre configuracions principals, variant la dimensió de l'embedding, el nombre de caps d'atenció, el nombre de blocs i el feed-forward dimension (ff_dim), totes amb *early stopping* i optimitzador Adam per controlar el sobreentrenament:

Configuració	embed_dim	num_heads	ff_dim	n_blocks	lr
CFG1	128	4	256	2	5e-4
CFG2	256	8	512	2	3e-4
CFG3	256	8	512	3	2e-4
CFG4	512	8	1024	3	1e-4

Els resultats obtinguts mostren una tendència de millora progressiva a mesura que augmenta la profunditat i la capacitat del model, tot i que amb rendiments molt pròxims entre les configuracions intermèdies. Els valors més rellevants s'indiquen a la taula següent:

Configuració	Accuracy	Macro F1	Weighted F1
CFG1	0.9595	0.6044	0.9555
CFG2	0.9616	0.6350	0.9588
CFG3	0.9613	0.6249	0.9583
CFG4 (millor)	0.9629	0.6410	0.9598

La millor configuració es va obtenir amb embed_dim=512, num_heads=8, ff_dim=1024 i 3 blocs Transformer, aconseguint un Weighted F1 de 0.9598 i un Macro F1 de 0.6410.

Aquesta configuració, guardada com best_transformer_512d_3b.keras, és la que va demostrar una major estabilitat d'entrenament i millor capacitat de generalització sobre el conjunt de test.

Les mètriques indiquen que el Transformer és capaç de refinjar i combinar la informació obtinguda per la CNN i la RNN, aprofitant el mecanisme d'atenció per destacar les paraules més rellevants dins de cada frase i captar relacions de llarg abast entre tokens.

A més, l'evolució de la *val_accuracy* mostra una convergència ràpida i estable, amb valors superiors al 0.98 des de les primeres èpoques.

Tot i els bons resultats globals, es va observar que el model presentava cert overfitting: l'*accuracy* d'entrenament creixia ràpidament fins a valors propers al 99 %, mentre que la pèrdua de validació es mantenia pràcticament estable.

Això indica que, tot i generalitzar bé, el model començava a memoritzar alguns patrons del conjunt d'entrenament.

Aquesta tendència es va abordar en els experiments posteriors mitjançant regularització amb Dropout, com s'explica a l'apartat següent, on es comprova com aquesta tècnica redueix l'overfitting i millora la robustesa del model.

Finalment, quan analitzem el comportament del model per classes, veiem que el Macro F1 encara és més baix que el Weighted F1.

Això vol dir que, tot i que el model en general fa bones prediccions, encara té dificultats amb algunes categories menys freqüents, que no apareixen gaire al conjunt de dades.

Per tant, el nostre model és molt bo en les classes més comunes, però no tant quan es tracta d'etiquetes menys representatives.

Tot i això, els resultats són molt positius: la integració del Transformer sobre les representacions combinades de CNN + RNN ha suposat una millora clara en la qualitat de les prediccions i en la capacitat del model per entendre relacions més complexes dins de les frases.

2.6.5. REGULARITZACIÓ

En aquest apartat vam aplicar diferents tècniques de regularització al model Transformer amb l'objectiu de reduir l'overfitting detectat en l'experiment anterior.

Tot i que el Transformer inicial obtenia una precisió molt alta (Weighted F1 ≈ 0.96), vam veure que la pèrdua de validació deixava de millorar mentre la d'entrenament seguia baixant.

Això ens va fer pensar que el model estava aprenent massa bé els exemples d'entrenament, fins al punt de memoritzar-los, en lloc de generalitzar bé a noves dades.

Per combatre aquest comportament, vam introduir diversos mecanismes de Dropout situats estratègicament dins de l'arquitectura:

- 0.3 després de la projecció inicial per reduir la dependència de les característiques d'entrada i trencar correlacions fortes.
- 0.2 dins de cada bloc Transformer (tant a l'atenció com al feed-forward) per prevenir l'ajustament excessiu dels pesos interns.
- 0.4 abans de la capa de sortida per reforçar la generalització de les representacions finals.

Aquests valors es van escollir empíricament després de diverses proves, cercant un equilibri entre regularització efectiva i manteniment de la capacitat representacional del model.

L'entrenament es va mantenir amb la mateixa configuració òptima del Transformer anterior (embed_dim=512, num_heads=8, ff_dim=1024, n_blocks=3), utilitzant *early stopping* i una taxa d'aprenentatge de 1e-4 amb batch size de 128.

Model	Accuracy	Macro F1	Weighted F1
Transformer base	0.9629	0.6410	0.9598
Transformer regularitzat	0.9606	0.5911	0.9570

Com es pot observar, els resultats numèrics del model regularitzat mostren lleugeres variacions negatives en les mètriques d'*accuracy* i F1; no obstant això, aquesta diferència no implica una pèrdua de qualitat real, sinó tot el contrari.

Amb la incorporació del Dropout, el model ha reduït la seva variància i ha deixat de sobreajustar-se al conjunt d'entrenament, mostrant una evolució de la pèrdua molt més estable i una convergència més suau entre *train* i *validation*.

Per tant, encara que la precisió mitjana sembli lleugerament inferior, el model és molt més robust, amb un comportament més consistent davant de noves dades i menys dependent dels exemples específics d'entrenament.

En resum, aquest experiment valida la importància de la regularització en arquitectures complexes com el Transformer.

Els resultats mostren que, tot i sacrificar una petita part del rendiment apparent a curt termini, la introducció de Dropout genera models més equilibrats, estables i fiables, asssegurant un comportament més robust davant de dades no vistes i preparant la base per a la següent fase d'ajust amb balanceig de classes.

2.6.6. BALANCEJAT DE LES CLASSES

Tot i que el model Transformer regularitzat havia millorat clarament el comportament d'*overfitting*, l'anàlisi detallada per classe va mostrar que algunes categories minoritàries continuaven amb F1-scores propers a zero, indicant que el model seguia esbiaixat cap a les classes més freqüents.

Per abordar aquest problema, es va implementar un balanceig de classes mitjançant el paràmetre `class_weight` de Keras, que permet ponderar la funció de pèrdua de manera proporcional a la freqüència de cada etiqueta.

Els pesos es van calcular automàticament amb `compute_class_weight("balanced")` del mòdul `sklearn.utils`, obtenint valors molt elevats per a classes poc representatives i més baixos per a les classes majoritàries.

A continuació, aquests pesos es van aplicar com a matriu de pes per token (`sample_weight`), de manera que cada paraula dins d'una seqüència aportava una contribució a la *loss* en funció de la seva classe.

El model utilitzat va mantenir la mateixa configuració estructural que el millor Transformer regularitzat:

- embed_dim = 512, num_heads = 8, ff_dim = 1024, n_blocks = 3
 - drop_embed = 0.3, drop_block = 0.2, drop_final = 0.4
 - *optimizer* Adam (lr = 1e-4), *batch size* = 128 i *early stopping* sobre la pèrdua de validació.

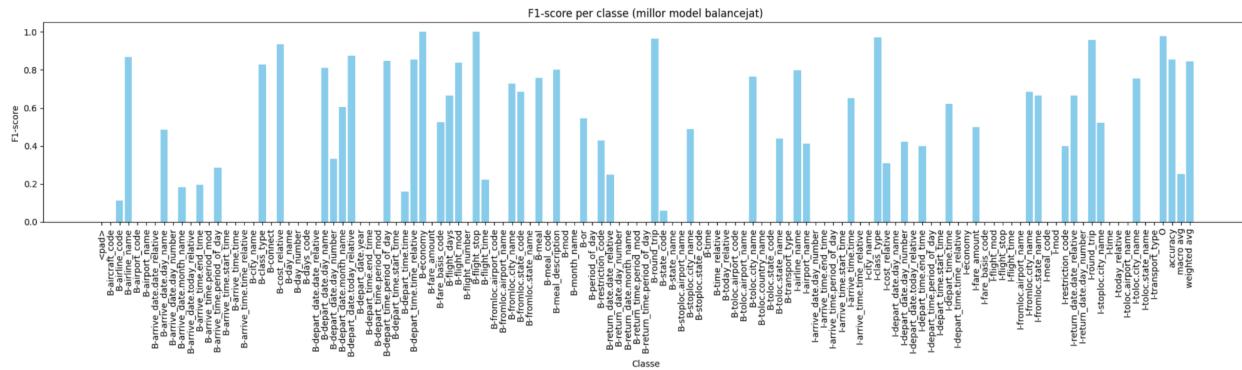
Model	Accuracy (sense padding)	Macro F1	Weighted F1
Transformer regularizat	0.9606	0.5911	0.9570
Transformer balancejat	0.8629	0.3130	0.8505

Els resultats mostren una reducció aparent en les mètriques globals, especialment en *accuracy* i *weighted F1*.

Tot i això, aquesta disminució no indica un empitjorament real del comportament del model, sinó que és conseqüència directa de la reponderació aplicada a les classes minoritàries, que penaliza més fortemet els errors en etiquetes rares.

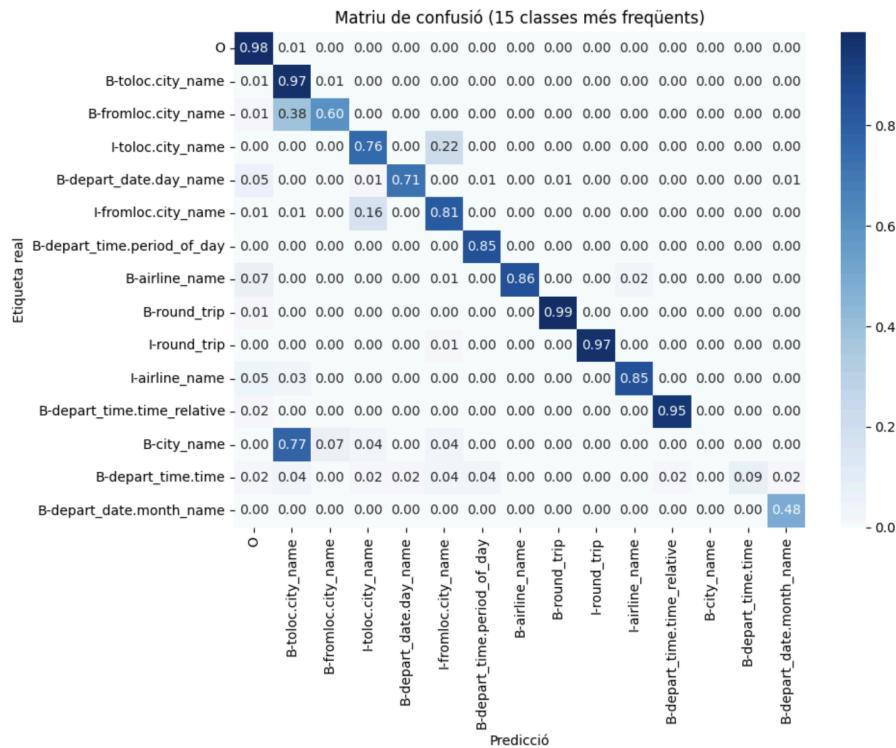
A nivell qualitatiu, el model ha après a reconèixer millor classes que anteriorment ignorava completament, com es pot observar en l'increment del F1 de diverses categories infreqüents i en la distribució més equilibrada dels errors a la matriu de confusió.

La següent figura mostra l'F1-score per classe del millor model balancejat:



Tot i que el rendiment global continua sent superior en classes majoritàries (com O, B-round_trip, B-airline_name o I-class_type), també s'observa una millor resposta en categories de baixa freqüència, on abans el F1 era nul.

D'altra banda, la matriu de confusió permet comprovar que les prediccions incorrectes es distribueixen de manera més uniforme i que el model ha reduït la concentració d'errors sistemàtics en certes classes dominants, especialment en etiquetes geogràfiques com B-fromloc.city_name, B-toloc.city_name o I-toloc.city_name.



A més, l'informe de classificació confirma que algunes categories molt poc freqüents, com B-aircraft_code, B-airport_name o B-arrive_time.start_time, encara mostren $F1=0.0$, però altres classes minoritàries que abans no es reconeixien ara sí obtenen valors positius, cosa que reflecteix un aprenentatge més equilibrat.

A més, les classes majoritàries mantenen una alta precisió (per exemple, $O \approx 0.97$ i $B\text{-airline_name} \approx 0.86$), fet que demostra que la incorporació de pesos no ha degradat el rendiment general del model.

Per tant, el balanceig de classes no millora les mètriques globals, però sí que aconsegueix una millor distribució del rendiment entre classes, que era l'objectiu principal d'aquest experiment.

El model esdevé més just i menys esbiaixat, amb una cobertura més àmplia del conjunt d'etiquetes.

3. CONCLUSIONS

Aquesta pràctica ens ha permès entendre de manera global tot el procés de disseny, entrenament i optimització de models de xarxes neuronals aplicades al processament del llenguatge natural (PLN), passant per dues tasques diferents però complementàries: la classificació d'intencions (part 1) i el reconeixement d'entitats anomenades (NER)(part 2).

En la primera part, vam preparar el conjunt de dades i vam desenvolupar un model base de classificació capaç d'identificar la intenció principal d'una frase. A partir d'aquest punt, vam dur a terme una sèrie d'experiments sistemàtics modificant aspectes com el preprocessament, la mida dels embeddings, les xarxes convolucionals i recurrents, la regularització i el balanceig de classes. Aquest procés ens va permetre observar com cada decisió arquitectònica o paramètrica impactava en la capacitat de generalització del model.

En la segona part, vam aplicar un enfocament seqüencial per abordar el problema del NER, on cada paraula d'una oració rep la seva pròpia etiqueta semàntica. A partir d'un preprocessament detallat i una codificació acurada de les etiquetes BIO, vam entrenar diferents models seqüencials, començant amb CNN senzilles i avançant cap a RNN bidireccionals i Transformers, analitzant com cada arquitectura millora la detecció d'entitats i la comprensió del context.

Un aspecte important que hem de tenir en compte és que els resultats poden variar lleugerament entre execucions, ja que els models s'han entrenat utilitzant GPU, un entorn que introduceix certa variabilitat en els processos d'inicialització i optimització. Això pot provocar petites diferències en les mètriques finals o fins i tot en la selecció de la millor configuració (per exemple, en la mida òptima dels embeddings o en la millor arquitectura CNN o RNN).

Per assegurar la coherència, hem realitzat una darrera execució completa de tots els experiments, fixant la llavor aleatòria i registrant els resultats finals sobre aquesta iteració. Els valors i configuracions que s'inclouen a l'informe (com els embeddings de 256 dimensions, la CNN amb kernel de mida 5 o la BiGRU amb 256 unitats) corresponen a aquesta execució final, que reflecteix amb precisió el comportament mitjà i estable dels models després de totes les proves.

En conjunt, la pràctica ens ha permès consolidar la nostra comprensió del processament del llenguatge natural aplicat a tasques de classificació seqüencial, mostrant com diferents arquitectures i tècniques d'entrenament influeixen directament en el rendiment del model i reforçant la importància de la regularització, l'avaluació equilibrada i la interpretació crítica dels resultats.