Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Sistemes basats en Frame 2, Name Entity Recognition

Tractament de la Veu i el Diàleg

Autors:

Zhihao Chen; Zhiqian Zhou

13 d'octubre de 2024

Índex

1 Introducció

Aquesta pràctica és una continuació de l'anterior, en la qual vam desenvolupar un sistema basat en Frames per a la classificació d'intencions mitjançant xarxes neuronals. En aquesta nova fase, afegirem un altre component essencial d'un sistema basat en Frames: un model de reconeixement d'entitats. L'objectiu d'aquest model és identificar amb precisió què demana l'usuari un cop determinat el domini de la conversa gràcies a la classificació d'intencions prèviament realitzada.

1.1 Conjunt de dades

Treballarem amb el mateix conjunt de dades utilitzat en la pràctica anterior, que conté un conjunt d'oracions en anglès relacionades amb demandes reals sobre vols, com ara consultar horaris, fer reserves o cancel·lar serveis. A diferència de l'anterior enfocament, en aquesta pràctica les etiquetes amb les quals treballarem per a l'aprenentatge supervisat de classificació són una cadena d'etiquetes BIO (Begin - Inside - Outside).

Cada cadena d'etiquetes correspon a una oració de demanda i té una longitud (quantitat d'etiquetes) igual al nombre de paraules de l'oració. Cada element de la llista representa una etiqueta assignada al token corresponent, utilitzant la codificació BIO (Begin - Inside - Outside). Aquesta codificació indica si un token és el començament d'una entitat (B), una part interna d'una entitat (I) sense ser-ne l'inici, o si està fora de qualsevol entitat (O). Aquesta estructura ens permet identificar amb precisió les entitats, fins i tot quan es troben adjacents les unes a les altres.

Tal com feiem anteriorment, abans de poder utilitzar aquest conjunt de dades en un model, realitzarem un procés de neteja i transformació per adequar-lo als requeriments d'un sistema de classificació basat en xarxes neuronals.

1.2 Desenvolupament de la pràctica

El desenvolupament d'aquesta pràctica segueix el mateix patró que l'anterior, basant-se en la resolució d'exercicis. Cada exercici representa un pas endavant en la construcció del sistema, des del preprocessament de les dades fins al disseny i l'entrenament dels models. Aquest enfocament progressiu permet desenvolupar el sistema de manera estructurada i gradual.

2 Exercici 1

En el primer exercici, se'ns demana carregar les dades d'entrenament, de validació i de test. No obstant això, en el conjunt de dades proporcionat només disposem de dades d'entrenament i de test. Per tant, per obtenir un conjunt de dades de validació que permeti avaluar el model durant el seu entrenament, es decideix dividir una part de les dades d'entrenament. Com que el conjunt d'entrenament té un volum considerable, extreure una petita part d'aquestes dades no afectarà significativament el seu rendiment.

Aquest procés de partició es realitza seleccionant les últimes 900 instàncies del conjunt d'entrenament per utilitzar-les com a dades de validació. Si imprimem el tamany de cada conjunt de dades veurem que els tamanys són correctes.

Aquesta partició de dades no és del tot robusta, ja que podria haver-hi dependències entre les instàncies

segons la seva posició en el conjunt de dades. Per tant, seleccionar les últimes 900 oracions podria provocar que aquestes comparteixin certes propietats que no es troben en la resta del conjunt. Això podria introduir biaixos en el model durant l'entrenament.

Una possible millora en futures iteracions seria utilitzar tècniques de partició més eficients, com ara *Random Sampling* o *Stratified Sampling*. Aquestes tècniques permetrien una selecció més representativa i aleatòria de les dades de validació, reduint el risc de biaixos i millorant la generalització del model.

3 Exercici 2

Com que utilitzem la mateixa base de dades que en la pràctica anterior, però amb etiquetes diferents, se'ns demana seleccionar les columnes (*features*) adequades, és a dir, la columna d'oracions i la columna d'etiquetes BIO. Si inspeccionem les dades, veurem que aquestes corresponen a les dues primeres columnes.

Les etiquetes, igual que en la pràctica anterior, es presenten com una cadena de caràcters que inclou explícitament els símbols de cometes (*"*). Python interpreta aquests símbols com a caràcters addicionals, cosa que augmenta la longitud de la cadena d'etiquetes i provoca un desajust (*mismatch*) entre la paraula i la seva etiqueta corresponent.

Per solucionar aquest problema, quan separem el target (etiquetes) de les variables explicatives (oracions), substituïm els caràcters de cometes per espais en blanc, eliminant-los així del text. Aquesta operació assegura que la longitud de les cadenes coincideixi amb la de les oracions corresponents, evitant errors durant el processament posterior.

4 Exercici 3

El preprocessament de les oracions segueix el mateix procediment que en la pràctica anterior.

- Primer, construïm un diccionari de vocabulari a partir de les paraules presents en les oracions. Posteriorment, filtrem aquest diccionari per quedar-nos només amb les 500 paraules més freqüents (tot i que el nombre de paraules és un paràmetre configurable).
- A continuació, convertim les oracions, que són seqüències de paraules, en seqüències d'enters que corresponen a les claus del diccionari.
- A diferència de la pràctica anterior, aquí el nombre de paraules en cada oració pot ser rellevant, ja que les etiquetes tenen la mateixa longitud que les oracions. Per aquest motiu, guardem en una llista la quantitat de paraules de cada oració.
- Finalment, apliquem la tècnica de padding per assegurar-nos que totes les seqüències d'enters tinguin la mateixa longitud (la longitud de la seqüència més llarga en les dades d'entrenament), de manera que puguin ser processades posteriorment pels models d'aprenentatge.

Apliquem el mateix processament amb les dades de validació i de test.

Cal tenir en compte que, en convertir les oracions a representacions numèriques amb el *tokenizer*, es filtren les paraules que tenen una clau inferior al nombre de paraules definit en el diccionari, és a dir, aquelles amb

una baixa freqüència d'aparició. Això pot causar problemes de desajust (*mismatch*) entre les oracions i les seves etiquetes. Per evitar aquest problema, configurarem una mida de diccionari més gran que el nombre de paraules presents en les oracions.

La manera òptima de solucionar-ho seria eliminar els tokens que tinguin una clau superior al nombre de paraules en el vocabulari, juntament amb les seves etiquetes corresponents. Tanmateix, en la pràctica, considerem que els nombres propis (que apareixen com a entitats d'interès) sovint tenen una freqüència més baixa que els noms comuns. Per això, configurar una mida de diccionari més ampli ens permet conservar aquestes entitats, cosa que és avantatjosa per al nostre model.

5 Exercici 4

Igual que a la primera part, aquí també necessitem convertir les diferents classes d'entitats en vectors *one-hot*. No obstant això, ara les etiquetes no són simplement una sola classe, sinó un conjunt de classes codificades amb el format BIO (Begin, Inside, Outside).

En primer lloc, cal determinar el nombre de classes existents. Ho fem mitjançant la funció *count_unique_entities*, que, donada una llista de cadenes d'etiquetes, retorna el nombre total de classes i quines són aquestes classes. Les classes segueixen el format ¡codificació B-I-O¿-¡Entitat¿ si es tracta d'una entitat, i simplement l'etiqueta "O"si no ho és.

El segon pas és codificar aquestes classes en etiquetes numèriques. Per fer-ho, utilitzem un model de *LabelEncoder* de la llibreria *scikit-learn*, que aprèn un conjunt d'etiquetes i, en inferir sobre una etiqueta, li assigna un valor numèric enter basat en les etiquetes apreses. Si apliquem aquest procés sobre totes les cadenes d'etiquetes, obtindrem una llista d'etiquetes numèriques per a cada oració, on cada etiqueta està alineada amb la paraula corresponent en l'oració. Posteriorment, apliquem la tècnica de *padding*, com ja vam fer abans, perquè totes les llistes d'etiquetes tinguin la mateixa longitud.

A continuació, convertim cada etiqueta de la llista d'etiquetes en un vector *one-hot*, amb un únic valor 1 en la posició que correspon al valor numèric assignat per l'*LabelEncoder*.

La mateixa transformació s'aplica a les dades de validació i de test. No obstant això, com a conseqüència d'una mala partició de dades, poden aparèixer etiquetes no vistes en les dades d'entrenament, fet que complica l'entrenament del model de classificació. Per solucionar aquest problema, iterem sobre les llistes d'etiquetes de les dades de validació i de test i, si trobem una etiqueta desconeguda (no vista durant l'entrenament), eliminem directament l'oració i la seva llista d'etiquetes corresponent.

6 Exercici 5

Després d'un preprocessament simple i d'una anàlisi general de la base de dades, passem a definir models d'aprenentatge profund per a la tasca de classificació d'entitats.

L'estructura de la xarxa neuronal proposada conté 4 capes:

- **Capa de `Embedding`:** Converteix les paraules en vectors numèrics que capturen relacions semàntiques, millorant la comprensió del context per part del model.
- **Capa de `LSTM`:** Utilitzem una xarxa LSTM per la seva capacitat de gestionar seqüències de dades i

capturar dependències a llarg termini. Aquesta capa és ideal per a tasques de NER perquè permet retenir informació rellevant de paraules anteriors en una oració.

- **Capa de `Dense Fully Connected`:** Extreu característiques més complexes, transformant les sortides de la LSTM en representacions més abstractes.
- **Capa de `Dense` amb funció d'activació `Softmax`:** Transforma les sortides en probabilitats per a cada classe d'entitats, i la classe amb la probabilitat més alta és la predicció final del model.

Aquí tens una versió millorada i més clara de la redacció:

Després de 5 èpoques d'entrenament, el model aconsegueix una exactitud de 0,98 tant en les dades d'entrenament com en les dades de validació i test. No obstant això, a causa de l'ús de la codificació BIO, utilitzar l'exactitud com a mètrica d'avaluació no és gaire eficaç. Les classes estan fortament desbalancejades (de les 9000 etiquetes hi ha més de 6000 "O"), la qual cosa introdueix un biaix en la inferència, fent que el model tendeixi a predir la classe més freqüent. En aquest cas, podria augmentar l'exactitud si el model prediu sistemàticament l'etiqueta "O", però això implicaria una incapacitat per reconèixer entitats reals.

Per abordar aquest problema, proposem utilitzar la mètrica de *macro-average F1-score*, que és més adequada per a conjunts de dades desbalancejats. Aquesta mètrica penalitza més severament els errors en classes menys freqüents, proporcionant una visió més justa del rendiment del model en la predicció de les entitats. Quan avaluem el model amb la *macro-average F1-score*, obtenim un resultat de només 0,32, cosa que indica que, tot i la seva alta exactitud general, el model té un rendiment pobre en la identificació d'entitats específiques.

7 Exercici 6

Un cop acabat la definició i l'entrenament d'un model senzill, que és ple de problemes, en aquest nou apartat ens demana aplicar diverses millores per corregir alguns errors presents i pujar el rendiment del model canviant l'arquitectura, canviant els hiperparàmetres o provant altres tècniques de processament.

7.1 Model Baseline

En aquest model *baseline* es busca definir una xarxa la més senzilla possible, amb la intenció que estigui plena de marges de millora. A l'exercici 5 ja vam dissenyar un model *baseline*; tanmateix, aquell no era avaluable de forma precisa, ja que utilitzava l'*accuracy* com a mètrica d'avaluació durant l'entrenament. Aquest enfocament no té en compte el desbalanceig de classes i, per tant, no reflecteix el rendiment real del model en la tasca de reconeixement d'entitats.

7.1.1 Ajustament del nombre d'èpoques

Abans de continuar, és essencial ajustar el nombre d'èpoques d'entrenament del model. Aquest hiperparàmetre és crucial, ja que pot influir en problemes com la sobreajustació (*overfitting*) o la manca d'ajustament (*underfitting*). Un nombre adequat d'èpoques és fonamental en l'entrenament de xarxes neuronals: a mesura que augmenta, es millora el rendiment del model i es redueix l'efecte d'*underfitting*, però també augmenta el risc d'*overfitting*. A més, un entrenament excessivament llarg consumeix molts recursos computacionals.

Per optimitzar aquest procés, aplicarem la tècnica d'*early stopping*. Aquesta tècnica consisteix a interrompre l'entrenament del model quan es detecta sobreajustació, basant-se en una mètrica de validació. En concret, si el rendiment en la partició de validació no millora durant un nombre determinat d'iteracions consecutives (definit pel paràmetre *patience*), l'entrenament es para automàticament. Això permet evitar la sobreajustació i optimitzar l'ús dels recursos.

7.1.2 Canvi de la mètrica d'avaluació

En aquest apartat, plantejarem un model amb la mateixa configuració (mateixes capes i neurones), però utilitzant una mètrica d'avaluació més eficaç per a la tasca de reconeixement d'entitats: la *macro-average F1-score*. En ser *macro-average*, aquesta mètrica no és sensible al desbalanceig de les etiquetes, ja que calcula la *F1-score* per a cada classe de manera independent i en fa una mitjana aritmètica, donant el mateix pes a cada classe.

Hem definit una funció `my_f1_score` que s'encarrega de calcular la puntuació F1 donat el model entrenat i les dades de predicció i etiquetes reals. A més, hem implementat la classe `F1ScoreCallback`, que calcula tant la *F1-score* per a les dades d'entrenament com per a les de validació en cada època. Aquesta classe, a més de calcular la mètrica, també s'encarrega de la funcionalitat d'*early stopping* descrita anteriorment, aturant l'entrenament quan detecta que la *macro-average F1-score* en la validació no millora significativament durant diverses iteracions consecutives.

Després d'efectuar aquests canvis, podem visualitzar les gràfiques que mostren l'evolució de la *loss* i la *macro-average F1-score* en les particions d'entrenament i validació durant el procés d'aprenentatge:

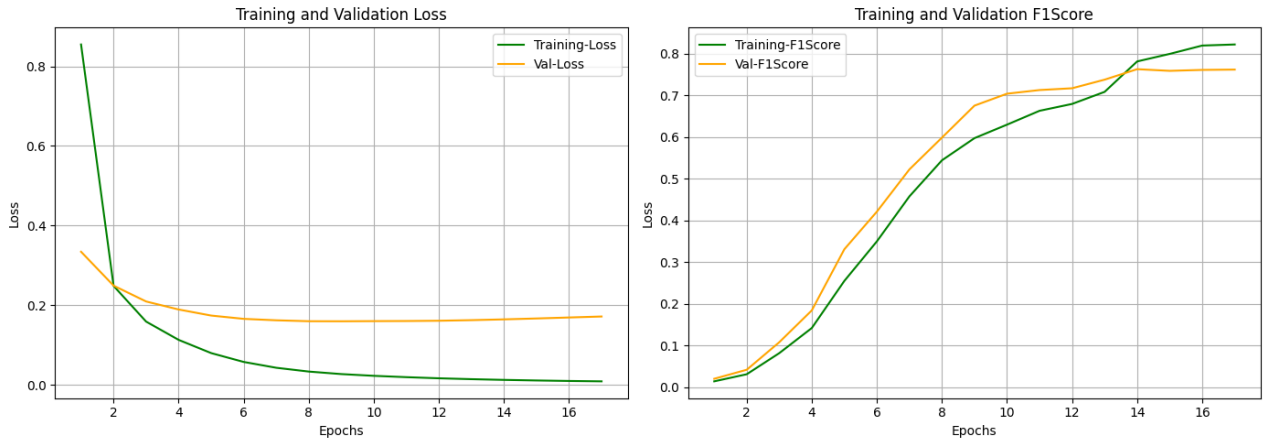

Figura 1: Evolució de la Loss i Macro-Average F1-Score del model baseline

Tal com s'observa, la *F1-score* en la partició de validació augmenta progressivament fins a arribar a 0.75 al voltant de les 24 èpoques, mentre que en la partició d'entrenament continua millorant fins a superar el 0.8 a les 25 èpoques, indicant una lleugera sobreajustació (*overfitting*).

En resum, en aquest model *baseline* hem aplicat dues millores: la incorporació de *early stopping* per evitar un entrenament ineficient i el canvi de mètrica d'avaluació a *macro-average F1-score*, la qual cosa permet tenir en compte el desequilibri de classes. Amb aquests ajustos, tot i mantenir una arquitectura senzilla, el model ja és capaç d'assolir un rendiment relativament alt, demostrant la importància d'utilitzar mètriques d'avaluació adequades.

7.2 Mida d'Embeddings

Com vam fer a la pràctica anterior, també aquí observarem l'impacte de la mida dels *word embeddings* sobre el rendiment del model. Els *word embeddings* són una representació numèrica de les paraules que permet al model capturar relacions semàntiques entre elles. La mida dels embeddings és un dels hiperparàmetres més importants, ja que determina la quantitat d'informació que cada vector pot emmagatzemar per representar una paraula.

Per avaluar com la mida dels word embeddings afecta el rendiment del model, hem dissenyat un banc de proves (*benchmark*) en què provem diverses mides d'embeddings. Concretament, canviem la dimensió dels vectors d'embeddings per valors de 50, 100, 150, 200, 250, 300 i 350. La idea és observar com augmentar la mida dels embeddings influeix en l'eficàcia del model, en termes de precisió i capacitat de generalització, i identificar la mida òptima que equilibri la qualitat de la representació i els recursos computacionals.

A continuació, presentem els resultats obtinguts. La primera columna indica la mida dels *word embeddings* utilitzada, mentre que la segona i tercera columnes mostren la *macro-average F1-score* assolida en les particions d'entrenament i de validació, respectivament:

Embedding Dim	**F1 Score (Train)**	**F1 Score (Val)**
50	0.954405	0.757534
100	0.888235	0.753935
150	0.871017	0.744615
200	0.873520	0.754563
250	0.907332	0.746873
300	0.871781	0.756162
350	0.893974	0.756113

Taula 1: F1 Scores per diferents valors d'Embedding Dimensional

Com podem observar, independentment de la mida dels *embeddings*, la *F1-score* obtinguda en la partició de validació és força consistent, amb valors al voltant de 0.75. Això indica que el model és relativament robust respecte a la mida dels *embeddings*, ja que tendeix a convergir cap a un rendiment similar. Aquesta estabilitat suggereix que altres factors del model (com la seva arquitectura) poden tenir un impacte més gran en el rendiment que no pas la mida dels *embeddings*.

Pel que fa a la partició d'entrenament, observem que una mida d'embeddings de 50 genera els millors resultats, amb una *F1-score* superior en uns 0.05 punts respecte a altres dimensions provades. Això pot ser degut al fet que una representació més compacta facilita l'aprenentatge, evitant la sobreajustació en un conjunt de dades limitat.

Per tant, en experiments futurs, utilitzarem aquesta mida d'embeddings de 50 per seguir optimitzant el model amb diferents arquitectures, ja que sembla oferir un bon equilibri entre simplicitat i rendiment.

7.3 Xarxes Convolucionals

A partir d'aquesta secció començarem a provar diferentes arquitectures que ens podria resultar benificiosa per a la tasca de reconeixement d'entitats.

La primera és utilitzar les capes convolucionals, que permeten aprendre relacions seqüencials locals entre les paraules d'una oració. Es podria afegir capes de Pooling per reduir la dimensionalitat i la complexitat, però en realitzar-lo es provoca un error en la dimensió de les dades. Per no complicar el treball, decidim utilitzar

només les capes convolucionals i els experiments consisteixen en modificar els seus hiperparàmetres com ara el nombre de filtres, el nombre de capes i la mida dels filtres.

Comencem pels nombres de filtres. El model convolucional bàsic en el qual partim els experiments té 256 filtres per capa, un nombre força elevat i amb la qual obtenim una f1-score de 0.73 en l'entrenament i 0.67 en la validació, mostrant que el seu rendiment és pitjor que el model baseline que tenim, que utilitza una LSTM. Per tal de millorar el seu rendiment, provem d'afegir i reduir filtres per veure l'impacte que té el nombre de filtres. Tanmateix, no hem vist cap millora, sinó que el rendiment ha pitjorat força en les dades d'entrenament, encara que a validació es manté igual.

També afegim capes, la qual cosa permet ampliar el context local après pel model i, probablement, millora significament el rendimet. Afortunadament, assolim una f1-score en la validació amb tan sols d'afegir dues capes més amb 256 neurones, mentre qe a l'entrenament puja fins a 0.9.

Finalment, la modificació del tamany dels kernels en base del model anterior també podria ampliar la capturació de les relacions seqüencials del model. No obstant això, sembla que fins al punt anterior ja era el límit dels models convolucionals sobre els nostres dades: el rendiment per a un tamany de kernel 5 i 7 és quasi invariable.

Experiment	**F1 Score (Train)**	**F1 Score (Val)**
basic	0.731246	0.676018
more_filters_512	0.579052	0.666804
less_filters_128	0.629499	0.663865
more_layers_1	0.871662	0.700237
more_layers_2	0.919996	0.782655
kernel_size_5	0.926330	0.764104
kernel_size_7	0.911162	0.744866

Taula 2: F1 Scores per diferents models CNN

7.4 Xarxes Recurrents

Les xarxes recurrents clàssiques sovint obliden informació rellevant en seqüències llargues. Per evitar aquest problema, utilitzem xarxes més avançades com LSTM (*Long Short-Term Memory*) i GRU (*Gated Recurrent Unit*), que poden retenir informació durant més temps gràcies als seus mecanismes de portes.

Els experiments en aquest apartat no inclouen models convolucionals, ja que no garanteixen una millora per a tasques de seqüències, i el nostre objectiu és comparar diferents models de *Natural Language Processing* (NLP).

El primer experiment consisteix a variar el nombre de neurones en les capes LSTM, passant de 64 unitats a 128 i 256. Els resultats de *macro-average F1-score* en la validació van ser 0.83 i 0.81, respectivament, mentre que en entrenament superaven 0.9, mostrant lleugera sobreajustació. Atès que aquest rendiment és prou alt, vam decidir no afegir més capes per evitar complicar l'arquitectura.

També vam provar una LSTM bidireccional, que va obtenir un rendiment de 0.99 en entrenament i al voltant de 0.8 en validació, indicant que la bidireccionalitat ajuda a captar patrons més profunds però augmenta el risc de sobreajustació.

Pel que fa a les xarxes GRU, amb 128 i 256 neurones, no es va observar millora significativa en la validació, mantenint-se al voltant de 0.75. L'ús d'una GRU bidireccional va resultar en *underfitting*, amb una caiguda de la precisió en entrenament a 0.72, possiblement degut a una pèrdua de capacitat per gestionar la complexitat

de seqüències quan es fan bidireccionals, requerint un ajustament més fi.

En concret, aquestes són els resultats obtinguts:

Experiment	F1 Score (Train)	F1 Score (Val)
basic	0.949319	0.819496
lstm_more_units_128	0.920266	0.830580
lstm_more_units_256	0.936398	0.807753
gru_more_units_128	0.927529	0.804012
gru_more_units_256	0.963936	0.839430
lstm_bidirec	0.987085	0.823044
gru_bidirect	0.732276	0.820886

Taula 3: F1 Scores per diferents models RNN

7.5 Transformers

Els Transformers són models basats completament en els seus mecanismes d'Atenció, la qual cosa permet capturar totes les relacions seqüencials a la vegada mitjançant productes matricials. Aquesta capacitat d'atenció evita que el model oblidi informació crítica. A més, gràcies a aquesta estructura basada en atenció, els Transformers poden processar tots els token d'una seqüència alhora, fen que l'entrenament sigui molt més ràpid.

Observant els resultats, però, veiem que el rendiment no és la desitjada. Això pot ser degut a la naturalesa de les nostres dades. Els Transformers solen beneficiar-se de contextos llargs per capturar patrons complexos de relacions entre paraules. En textos curts, el model té menys informació per establir relacions i limita el seu potencial. A més, els Transformers necessiten prou exemples de cada tipus d'entitat per aprendre a reconèixer patrons específics, però al dataset hi ha etiquetes amb pocs exemplars, per tant, el model podria no generalitzar bé.

Analitzant els resultats dels experiments veiem que en general no hi han greus overfittings. Això pot ser degut a l'ús de les capes Dropout. A més, el seu *loss* obtingut és el més baix entre tots els models, cosa que maximitza *accuracy* pero no necessàriament *f1-score macro*.

Experiment	F1 Score (Train)	F1 Score (Val)
basic	0.949319	0.819496
lstm_more_units_128	0.920266	0.830580
lstm_more_units_256	0.936398	0.807753
gru_more_units_128	0.927529	0.804012
gru_more_units_256	0.963936	0.839430
lstm_bidirec	0.987085	0.823044
gru_bidirect	0.732276	0.820886

Taula 4: F1 Scores per diferents models Transformers

7.6 Regularització

S'ha utilitzat la tècnica de regularització amb dropout per reduir l'overfitting en un model LSTM bidireccional amb 256 unitats. L'objectiu, doncs, és observar l'impacte d'usar Dropout al model amb més overfitting obtingut en proves anteriors.

Per provar l'impacte de diferents nivells de dropout, hem avaluat tres valors de probabilitat: 0.1, 0.2 i 0.3, col·locant-lo just després de la capa d'embeddings per veure com afecta el rendiment general. Addicionalment,

hem realitzat un experiment final aplicant dropout a dos punts diferents de l'arquitectura: tant després de la capa d'embeddings com després de la capa LSTM per tallar algunes connexions ja que aquesta capa és la decisiva del comportament del model.

Experiment	**F1 Score (Train)**	**F1 Score (Val)**
dropout_0.1	0.905985	0.825029
dropout_0.2	0.926984	0.833023
dropout_0.3	0.784107	0.791167
more_layer	0.855630	0.801430

Taula 5: F1 Scores per diferents models amb DropOut

Observant els resultats anteriors, veiem que la configuració amb dropout de 0.2 aplicada després de la capa d'embeddings ofereix el millor equilibri entre rendiment i generalització. Aquest pot reduir l'overfitting sense perjudicar gaire l'aprenentatge del model.

7.7 Balancejat de classes

Per abordar el desbalanceig de classes en el problema de reconeixement d'entitats (NER), es pot penalitzar més les prediccions incorrectes de les classes minoritàries assignant-hi ponderacions més altes. D'aquesta manera, un error en una classe majoritària tindrà menys impacte en la funció de pèrdua que un error en una classe minoritària.

En problemes de classificació estàndard, aquest ajust es pot fer fàcilment amb el paràmetre `class_weight` en el mètode `fit`. Però en problemes seqüencials com el NER, on cada paraula té la seva pròpia etiqueta, `class_weight` no es pot aplicar directament.

Com a solució, hem definit una funció de pèrdua personalitzada, `weighted_categorical_crossentropy`, que incorpora les ponderacions de cada classe en el càlcul de la pèrdua. Així, només cal canviar la funció de pèrdua del model per aquesta nova definició.

Tot i això, durant l'entrenament, vam observar que la *macro-average F1-score* tant en la partició d'entrenament com en la de validació va ser significativament inferior en comparació amb els models anteriors, tot i que estàvem utilitzant el millor model base vist fins ara. Això es deu al fet que les classes minoritàries tenen molt poques instàncies en el conjunt de dades. Encara que els assignem més pes, el model no és capaç de predir-les correctament per la seva escassa presència. Per tant, es veu que el *loss* en la partició de validació es convergeix en un valor molt gran respecte els models anteriors.

Si representem gràficament la relació entre les ponderacions de les classes i les seves *F1-scores*, veurem que, per a moltes etiquetes amb un pes elevat (barres grogues), la *F1-score* associada (barres blaves) és baixa. Això indica que assignar un pes alt a classes poc freqüents no millora el rendiment del model, sinó que redueix la *macro-average F1-score* global.

Dit d'una altra manera, per aconseguir una macro-average f1-score alta, és important veure barres blaves especialment altes en aquells casos on les barres grogues també són altes (que se solapen una sobre l'altra), ja que això indica que el model ha aconseguit una bona classificació de les classes menys freqüents, les quals tenen una ponderació elevada.

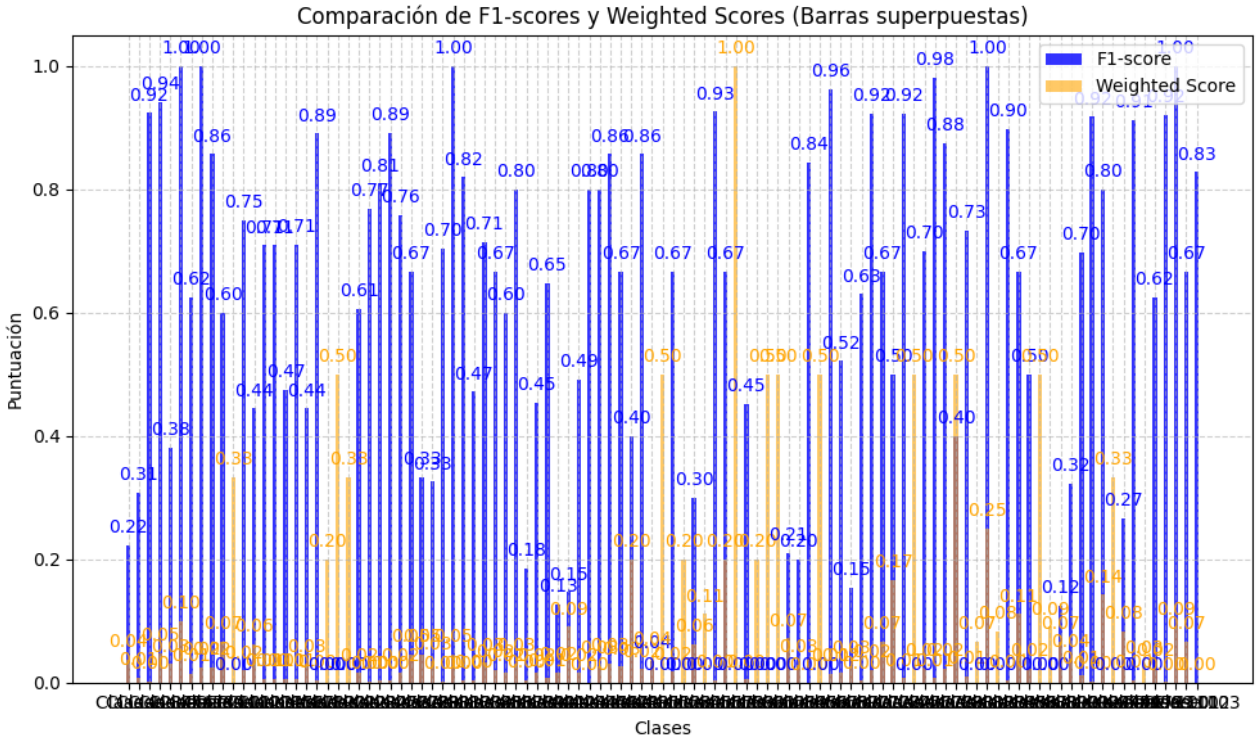

Figura 2: Relació entre f1-score d'etiquetes i ponderacions assignades

8 Conclusió

En conclusió, el punt clau que va portar a una millora significativa del model en aquesta pràctica va ser l'ajustació del nombre d'èpoques. Aquesta modificació va permetre millorar el rendiment del model, elevant la seva *f1-score* de 0.5 a 0.8, mantenint una *accuracy* alta, la qual cosa indica que el model fa prediccions correctes en general. Un altre clau és la modificació de la mètrica d'avaluació, ja que en una tasca de NER les classes són fortament desbalancejades.

Pel que fa al rendiment dels diferents models (CNN, RNN i Transformer), no es va observar una diferència significativa entre ells, tots van aconseguir resultats propers a una *f1-score* de 0.8. Per aquesta raó, no s'ha posat moltes gràfiques d'evolució en aquest informe, ja que totes tenen una forma similar. Encara així, amb l'arquitectura LSTM destacant per aconseguir un bon f1-score més fàcilment i amb Tranformers un *loss* més baix. Els models van mostrar un aprenentatge estable i progressiu fins a les 30 èpoques, amb algunes mostres de sobreajustament. En aquells casos, vam aplicar tècniques de *Dropout* per solucionar el problema, mantenint la bona avaluació en la partició de validació.

La limitació per aconseguir una *f1-score* superior a 0.8 probablement es deu al desbalanceig de les classes, ja que les classes minoritàries tenen massa poques instàncies, fet que fa difícil que el model les classifiqui correctament tot i assignar-los una ponderació més alta.

Finalment, cal destacar que durant tota la pràctica es va definir una *seed* per garantir la reproductibilitat dels experiments. Tot i així, com que cada experiment es va realitzar una sola vegada, no es pot determinar la variabilitat dels resultats. Per obtenir una millor comprensió de la variabilitat del rendiment, seria necessari executar els experiments diverses vegades, però això requeriria un hardware més potent del que tenim disponible.