TVD-GIA (FIB) Laboratory Assignment

Lab 2.2: Named Entity Recognition amb deep learning

Gerard Gómez Izquierdo
Víctor Molina Díez

November 19, 2023

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Contents

1 Introducció

En aquest treball ens centrarem a desenvolupar un model per dur a terme la tasca de Reconeixement d'Entitats Nominals (sigles en anglès: NER). Per executar aquesta tasca hem utilitzat un corpus amb 33394 frases (versió mini) amb les seves entitats nominals etiquetades amb el format B, I, L, O, U. Les diferents tipologies d'entitat que es tenen en compte en aquest corpus són: audiovisual_person, tipoContenido, entity.genero, audiovisual_content_title, entity.proveedor, entity.conQuien, entity.cast, entity.keyword, entity.productora, entity.favoritos.

A continuació, explicarem com hem preparat les dades per poder entrenar el model. Posteriorment, realitzarem diverses hipòtesis i experiments provant diferents arquitectures i diferents paràmetres per tal de determinar quina ens proporciona els millors resultats. És fonamental utilitzar les dades de validació per ajustar l'arquitectura i els paràmetres del model, reservant les dades de prova únicament per avaluar el comportament dels models ja ajustats en condicions totalment noves. Finalment, portarem a terme la predicció del conjunt de prova utilitzant els models que hagin obtingut millors resultats en el conjunt de validació. A més, analitzarem i compararem els resultats entre arquitectures que utilitzen diferents capes neuronals.

Convé destacar que tots els experiments realitzats junt amb els seus resultats (F1-score macro, gràfica amb losses i model.summary) es troben registrats a la fulla d'Excel dins de la carpeta .zip de l'entrega, a més al notebook es poden trobar els codis de tot el preprocessament de dades i dels millors experiment per les xarxes CNN, RNN, Transformers, Dropout i Balanceig de classes.

2 Results

2.1 Preprocessament de dades

Per poder començar a crear i entrenar els nostres models és necessari preprocessar les dades perquè aquestes siguin interpretables per les nostres arquitectures. Per fer-ho, hem seguit els següents passos:

1. **Corregim sentences del dataset:** Abans de la creació del vocabulari hem creat la funció *process_strings*, aquesta funció s'encarrega de detectar les paraules de les frases del nostre dataset que pertanyen al llistat de caràcters especials del tokenizer per defecte i les substitueix per el token OOV. Aquest pas es molt important per dues raons. En primer lloc, no volem que el nostre model utilitzi aquests caràcters perquè afegeixen complexitat i no aporten informació rellevant. I en segon lloc, perquè si per solucionar això utilitzem el paràmetre *filter* per defecte al crear el tokenizer el que estarem fent és eliminar els caràcters estranys al crear les seqüències d'enters que representen les paraules, i això faria que es desquadressin les paraules amb les etiquetes de les oracions.

   Exemple:

   Oració: 'alguna que sea similar a anti matter en movistar +'

   Etiquetes: O O O O O B-tef.audiovisual_content_title L-tef.audiovisual_content_title O B-tef.entity.proveedor L-tef.entity.proveedor

   Si passem a seqüencia d'enters amb paràmetre filter per defecte al tokenizer obtindrem: [8, 5, 49, 53, 11, 1843, 1844, 12, 28], la qual te mida 9 i no coincideix amb la mida de la seqüència d'etiquetes (10), en canvi utilitzant el token OOV introduït amb la funció *process_strings* obtindrem la secuencia: [8, 5, 49, 53, 11, 1843, 1844, 12, 28, 26], la qual, sí té mida 10 i totes les etiquetes correspondran a la paraula original.

2. **Construcció del Vocabulari:** El segon pas que hem realitzat consisteix en la construcció del vocabulari a partir de les paraules presents a les oracions d'entrenament. Per a aconseguir-ho, hem creat una instància del tokenitzador *Tokenizer* de Keras i l'hem entrenat utilitzant la funció *fit_on_texts* aplicada a les frases d'entrenament. Hem considerat totes les paraules del vocabulari perquè no tenim un vocabulari massa extens, i hem inclòs l'opció *oov_token*='OOV' i *filters*=''. L'ús de *oov_token* en tasques de Reconeixement d'Entitats Nominals (NER) és fonamental, ja que permet representar qualsevol paraula que no es trobi al vocabulari amb el token que desitgem. En aquest cas, aquesta funcionalitat és essencial perquè, en cas contrari, en convertir les seqüències de paraules en seqüències d'enters, podria succeir que les etiquetes corresponents a cada paraula es desalineessin.

   Exemple: Suposem que fem servir num_words = 1500 i volem transformar la següent oració en una seqüència d'enters usant el *Tokenizer*.

Oració: 'scott gold le odio' $->$ **Seqüència d'enters**: [1856, 1327, 198, 241].

Donat que estem fent servir un vocabulari de 1500 paraules, la seqüència d'enters amb padding quedaria de la següent manera: [1327, 198, 241, 0],podem veure com desapareix la primera paraula, en canvi, les etiquetes corresponents a la seqüència es veurien així: [B-tef.audiovisual_person, L-tef.audiovisual_person, O, O]. Com es pot observar, totes les etiquetes ara s'han desalineat, ja que es troben en una posició diferent de l'índex de la paraula a la qual fan referència. Això pot generar problemes a l'hora d'entrenar i avaluar el model, perquè també afecta les seqüències d'enters de les frases de validació i prova, on trobarem paraules que no s'hagin vist prèviament en el corpus d'entrenament.

El resultat d'utilitzar *oov_token* seria el següent: **Seqüència d'enters**: [1, 1327, 198, 241], on l'1 representa totes aquelles paraules desconegudes.

Internament el Tokenizer conté un diccionari que té les paraules com a claus i un índex associat a cada paraula com a valors. **Vocabulari**: {'OOV': 1, 'de': 2, 'me': 3, 'que': 4, ...}

En el nostre cas serà necessari fer això perquè les etiquetes coincideixin en la part d'avaluació (validació i test), ja que en la part d'entrenament al utilitzar tot el vocabulari no hi haurà casos on no coneixem la paraula.

Per altra banda, utilitzar el paràmetre *filters* = '' al tokenizer es necessari perquè hi ha casos on, per exemple, trobem paraules compostes amb guió, les quals tenen associada una única etiqueta. Si utilitzem el paràmetre *filters* per defecte eliminarem el guió i ens quedarem amb dues paraules diferents. Això fa que al passar les frases a seqüències d'enters es desquadrin les etiquetes de les paraules, ja que a la segona paraula se li assignaria una etiqueta que no li correspon .

Exemple:

Oració: la show de joo ye-bin

Seqüència d'enters amb paràmetre filters per defecte: [3, 73, 2, 3185, 3186, 3187]

Seqüència d'enters amb paràmetre filters buit: [3, 73, 2, 3185, 3186]

Etiquetes: O U-tef.entity.tipoContenido O B-tef.audiovisual_person L-tef.audiovisual_person

Podem veure com al utilitzar *filters* per defecte tenim una seqüencia de 5 etiquetes assignada a una seqüència de 6 paraules.

3. **Seqüències d'enters:** Seguidament, tal i com hem esmentat abans, hem convertit les oracions de cada partició del dataset en seqüències de nombres enters utilitzant la funció *tokenizer.texts_to_sequences*, la qual substitueix cada paraula per el seu corresponent index al vocabulari.

   Oració: 'scott gold le odio' $->$ **Seqüència d'enters**: [1856, 1327, 198, 241].

4. **Longitud de les Frases:** El tercer pas ha estat guardar la longitud de totes les seqüències de cada partició. Això ho fem perquè en passos futurs puguem avaluar els models que creem sense tenir en compte les prediccions de les etiquetes de padding. A més a més, ens guardem la mida de la seqüència més llarga en el conjunt de training.

5. **Padding de les Seqüències:** Per aconseguir que totes les seqüències tinguin la mateixa longitud, hem fixat la mida de totes les oracions de les tres particions amb la longitud de la frase més llarga trobada en el pas anterior. Hem afegit zeros (padding) per la dreta a les oracions amb una longitud menor a la longitud establerta. Per aquest últim pas hem fet ús de la funció *pad_sequences*.

   Seqüències d'enters amb padding (opció "post"): [1855, 1326, 197, 240, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

6. **Codificació de les Etiquetes** Per aconseguir que les diferents classes de les entitats siguin interpretades correctament per la nostra arquitectura, necessitem passar les etiquetes a format One-Hot, per això hem realitzat els següents passos:

   (a) **Comptatge d'Entitats Úniques:** Hem implementat la funció *count_unique_entities*, la qual rep una llista d'etiquetes, on cada etiqueta correspon a una frase,i retorna quantes i quines són les classes úniques que hi trobem.

Seguidament, li hem passat la llista d'etiquetes de cada frase del conjunt d'entrenament a la funció i hem trobat 38 etiquetes úniques. Aquestes etiquetes estàn compostes pel format d'etiquetatge BILOU i les classes enumerades a l'apartat d'introducció.

(b) **Afegim Padding com a Entitat Única:** En aquest següent pas hem afegit en la posició 0 de la llista d'entitats úniques la classe '$< pad >$' i hem sumat 1 al nombre d'entitats úniques. Això ens servirà per poder codificar l'etiqueta d'aquelles posicions de les seqüències de paraules on hem afegit padding.

(c) **Eliminació d'Etiquetes desconegudes:** Donat que la partició de test conté una oració amb entitats etiquetades amb 'U-tef.entity.review' i a la partició de train no apareix cap seqüència amb aquesta etiqueta, procedim a eliminar en la partició de test l'oració que conté aquesta etiqueta junt amb l'etiqueta corresponent a l'oració.

(d) **Codificació Numèrica de les Classes:** Després, pel conjunt de labels únics (incloent el padding) de la partició d'entrenament, hem creat una instància de *LabelEncoder()* que hem entrenat perquè assigni etiquetes numèriques a les diferents etiquetes de cada frase. L'etiqueta '$< pad >$' se li ha assignat el valor 0.

(e) **Creació de Llistes d'Etiquetes Numèriques:** Un cop entrenat el *LabelEncoder()*, hem creat les llistes d'etiquetes numèriques de cada partició per a cada frase. Aquestes llistes reben el nom de *part_numerical_labels* i contenen una llista de subllistes on cada element de les subllistes és el valor numèric de l'etiqueta assignada al valor de la paraula corresponent en aquella posició.
part_numerical_labels: [[2, 20, 28, 28],...]
part_labels: ['B-tef.audiovisual_person L-tef.audiovisual_person O O', ...]

(f) **Afegim Padding com a Etiqueta:** A continuació, per cada partició del dataset hem afegit zeros (padding) per la dreta a les seqüències numèriques d'etiquetes amb una longitud menor a la longitud establerta. D'aquesta manera als tokens padding de les seqüències de paraules els hi correspondrà la codificació de l'etiqueta assignada al padding (0).

(g) **Conversió a Format One-Hot:** Finalment, l'últim pas ha estat convertir aquestes etiquetes numèriques en format one-hot utilitzant la funció *to_categorical*. Això significa que per a cada paraula de cada frase tindrem un vector one-hot que representa la classe a la qual pertany.
Classe '$< pad >$' en format one-hot: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

2.2 Raonament i tasques prèvies a l'entrenament

Prèviament a començar l'entrenament de models, és necessari tenir clar certs aspectes que influiran en com nosaltres realitzarem i avaluarem els experiments. En primer lloc, tal com hem esmentat a la introducció d'aquest informe, tots els nostres experiments es faran sobre la partició de validació, ja que són dades que el nostre model no veu durant l'entrenament i així podrem veure el seu comportament quan rep dades desconegudes.

En segon lloc, per poder avaluar el nostre model utilitzarem la funció *preds_to_index*, la qual ens permet transformar les prediccions i les etiquetes a llistes d'un sol nivell i no tenir en compte les prediccions de l'etiqueta padding a l'hora d'avaluar el nostre model. Posteriorment, l'output d'aquesta funció l'usarem en la funció *classification_report* que ens proporcionarà les diferents mètriques sobre les prediccions del model.

En tercer lloc, és essencial decidir quina mètrica farem servir per avaluar els nostres models. En aquest cas utilitzarem la mètrica F1-score macro avg. La F1-score és una mètrica que combina la precisió i el recall en una sola mesura. En concret, F1-score macro calcula la F1-score per cada classe i després pren la mitjana aritmètica simple de tots aquests valors. Això significa que cada classe té el mateix pes en el càlcul de la puntuació global, independentment de la seva grandària, fet que ens permet avaluar el model sense tenir en compte el desequilibri de classes.

En quart lloc, per intentar fer els experiments més reproduïbles hem fixat les seeds de NumPy i Keras, però malgrat això segueix havent-hi una component d'aleatorietat als experiments que fa que els resultats variïn una mica en cada execució. Per aquesta raó, hem tingut en compte més d'una execució a l'hora

de prendre decisions durant el període d'experimentació.

Finalment, el pas previ que hem realitzat abans de començar la fase d'experimentació, ha estat implementar la funció *plot_training_curve*, la qual ens permet visualitzar l'evolució de les corbes de loss de train i de validació durant l'entrenament, fet que ens ajudarà a veure si el nostre model està ben ajustat, o si, per contra, estem fent overfitting o underfitting.

2.3 Disseny del Baseline

Per començar la fase d'experimentació, el primer que hem fet ha estat crear un model baseline semblant al que vam crear a la part 1 d'aquesta practica, amb la diferencia que ara, enlloc d'utilitzar GlobalMax-Pooling1D per capturar les dependències seqüencials, farem us d'un bloc de transformer. Per tant, la nostra arquitectura inicial consta d'una capa de TokenAndPositionEmbedding (embeddings + potistional encoding) amb embeddings de mida 64, un block de transformer amb 2 heads i una feed forward amb 32 neurones, una capa densa de 64 neurones i una capa densa de sortida amb 39 neurones. Cal destacar que tots els valors escollits els anirem modificant en funció dels resultats que anem obtinguem en futurs experiments.

Un cop definida aquesta primera arquitectura, hem realitzat experiments per escollir quins valors per els paràmetres de batch_size i nombre d'epochs usarem al llarg dels nostres experiments. Aquests van ser els resultats de la primera execució:

Figure 1: Bsize = 64, epochs = 10, F1-score = 0.79

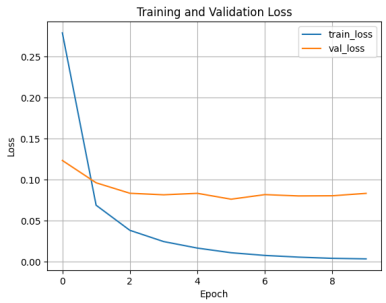

Figure 2: Bsize = 128, epochs = 10, F1-score = 0.80

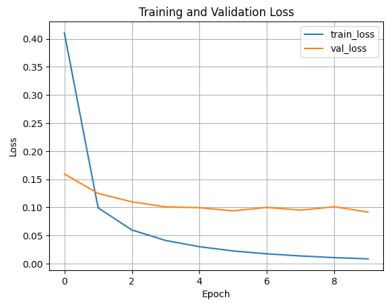

Figure 3: Bsize = 256, epochs = 10, F1-score = 0.76

En aquests experiments podem veure com amb batch size 64 fem overfitting des de l'època 3, ja que la corba de validació comença a tenir soroll i a augmentar. En canvi, quan utilitzem batch size 128 veiem com la corba de validació té molt menys soroll i no apreciem un clar overfitting, encara que la corba de validació cap al final de l'entrenament sembla tenir una petita tendència a pujar i, per tant, si afegim més èpoques podríem generar overfitting. Finalment, amb batch size 256 apreciem com obtenim resultats molt similars als anteriors, amb la diferència que en aquest cas la F1-score que obtenim és més baix i les losses de la corba de validació són més altes, per aquesta raó usarem els valors del segon experiment (batch_sixe = 128 i epochs = 10). Destaquem que no ha calgut experimentar amb epochs perquè en el segon experiment ja ens hem quedat al llindar de l'overfitting.

2.4 Task 1: Mida embeddings

Un cop hem definit el nostre model baseline, començarem experimentant amb la mida dels embeddings. L'experimentació amb la mida dels embeddings és essencial perquè aquesta pot influir en la capacitat de captar relacions semàntiques entre les paraules del context. Aquest fet té un impacte significatiu en una tasca com el reconeixement d'entitats nominals, ja que el context juga un paper essencial en la identificació d'aquestes. Utilitzar embeddings massa petits pot portar el model a perdre informació important sobre les relacions entre paraules, mentre que una mida massa gran pot afegir complexitat innecessària al nostre model i conduir a fer overfitting i a la ineficiencia computacional. El nostre objectiu és trobar una mida que ens aporti un equilibri, permetent al model generalitzar bé sobre les dades. Volem destacar que, per aquesta experimentació, és molt important utilitzar una capa com el bloc de transformer, ja que aquesta és més sensible a la mida dels embeddings i ens proporcionarà més coneixement a l'hora de seleccionar la mida òptima.

Per assolir el nostre objectiu hem realitzat els experiments amb les mides 32, 64, 128 i 256. Aquests han sigut els resultats:

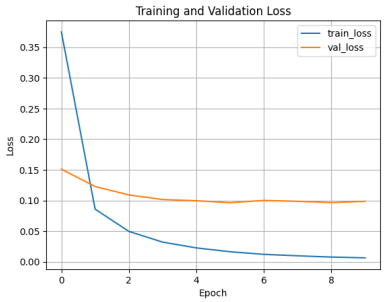

Figure 4: Mida Embedding = 32, F1-score = 0.78

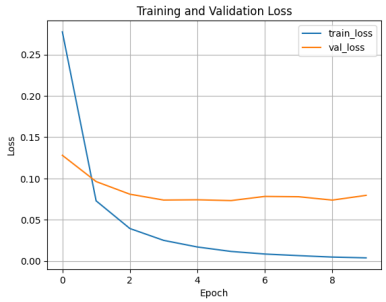

Figure 5: Mida Embedding = 64, F1-score = 0.79

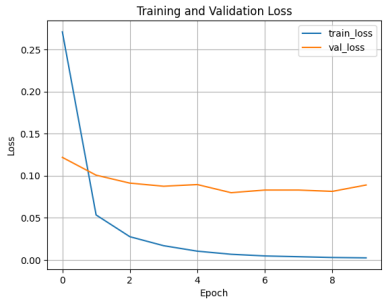

Figure 6: Mida Embedding = 128, F1-score = 0.80

En aquests experiments, podem observar com totes les corbes de loss són bastant estables i similars. No obstant això, és destacable que amb els embeddings de mida 32 obtenim una loss de validació més elevada i un F1-score més baix; per tant, no és la mida que seleccionarem. Pel que fa als embeddings de mida 64 i 128, notem com la funció de loss a la validació arriba a valors més baixos amb els de mida 64. En canvi, el valor del F1-score és més alt en el cas dels embeddings de mida 128. Això succeeix perquè, al tenir un desequilibri significatiu de classes, la loss de validació es veu més afectada per la millora de prediccions de la classe majoritària, mentre que el F1-score és més sensible a la millora en les classes minoritàries.

Donat que la nostra mètrica de referència és el valor F1-score, hem decidit seleccionar la mida 128, ja que aquesta mitiga millor el desequilibri de les classes. També notem com al final de la corba de loss de validació dels embeddings de mida 128, la loss comença a augmentar. Això podria ser degut a la complexitat de 1,159,367 paràmetres, i podríem estar començant a patir overfitting. És important tenir això en compte en les futures modificacions de l'arquitectura del model.

En el cas dels embeddings de mida 256, els quals no hem afegit per qüestió d'espai, la corba de validació presenta molta variabilitat, començant a mostrar signes d'overfitting des de l'inici de l'entrenament. Malgrat duplicar la complexitat, aconsegueix arribar al mateix valor de F1-score que en el cas dels embeddings de mida 128. Per tant, la mida de referència que utilitzarem a partir d'ara serà la de 128. Tot i això, en futurs experiments tornarem a explorar aquests valors, juntament amb el paràmetre 'num_heads' utilitzat en els blocs de transformers.

2.5 Task 2: Xarxes Convolucionals i Pooling

En aquesta part explicarem el camí que ha seguit la nostra experimentació amb xarxes convolucionals. Treballar amb xarxes convolucionals es molt interessant en la tasca de NER perquè aquestes tenen la capacitat de capturar inorfmació del context local d'una paraula. Això es molt important a l'hora d'etiquetar una entitat nominal, ja que el seu significat pot dependre fortament de les paraules que l'envolten. En el nostre model, hem utilitzat capes convolucionals per processar els embeddings de les paraules i capturar patrons locals significatius. Això ens ha donat la capacitat de detectar característiques rellevants en el context immediat de cada paraula, millorant així el F1-score del reconeixement d'entitats nominals. En particular, hem utilitzat múltiples capes convolucionals amb diferents mides de finestra per explorar diferents nivells de context. Això ens ha permès capturar informació a diferents nivells i veure quina arquitectura funciona millor en el nostre cas.

L'arquitectura que utilitzarem en l'experimentació que seguidament comentarem consisteix en una capa de embeddings de la mida marcada anteriorment, a més afegirem el paràmetre 'mas_zero' com a True perquè d'aquesta manera no considerarem el padding durant l'entrenament del model. Seguidament, afegirem les nostres capes de convolució i pooling amb padding = same perquè les dimensions a l'entrenar quadrin i finalment, com en l'arquitectura anterior, una densa de 64 neurones i una densa d'output amb 39 neurones.

El primer experiment ha estat utilitzar una sola capa de convolució i hem provat amb valors de 16, 32 i 64 neurones amb mides de kernel de 3 i 4. Per temes d'espai en el document només afegirem els 3 experiments més rellevants, els demés es poden trobar al fitxer d'experiments dins del .zip.

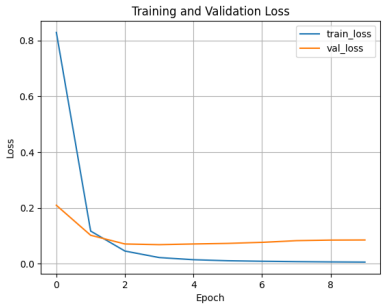

Figure 7: 16 neurones, kernel size = 3, F1-score = 0.78

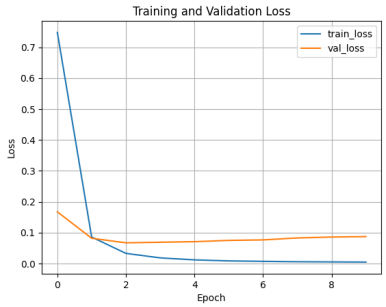

Figure 8: 32 neurones, kernel size = 3, F1-score = 0.81

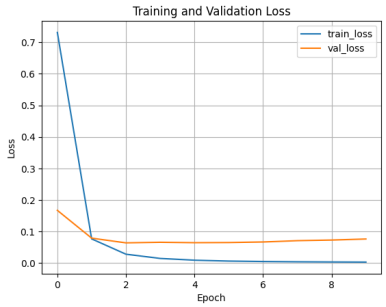

Figure 9: 32 neurones, kernel size = 4, F1-score = 0.82

En els nostres experiments inicials, incorporar una capa convolucional amb 16 neurones i un kernel de mida 3 va revelar signes d'overfitting, ja que la val_loss va augmentar a partir de l'època 3, però només vam arribar a un F1-score de 0.78 . Malgrat això, com hem destacat anteriorment, la val_loss no ofereix la mateixa informació que el F1-score, el qual és més sensible a les millores en les classes minoritàries. Per aquest motiu, vam continuar incrementant la complexitat del model per veure si, tot i que la val_loss pugés, podríem millorar els resultats del F1-score. A més, vam explorar l'opció d'augmentar la mida del kernel per captar més informació del context immediat de cada paraula.

Els resultats mostren que, amb 32 neurones i un kernel de mida 4, vam aconseguir un F1-score de 0.82. Això es va repetir amb 64 neurones, pel que vam prendre la decisió de mantenir les mides de la figura 9 per evitar afegir complexitat innecessària.

Després d'haver decidit les mides de la primer capa, anem a afegir una segona per veure si d'aquesta manera el model es capaç d'aprendre relacions més complexes entre les paraules que l'ajudin a etiquetar-les millor. Aquests han estat els resultats:

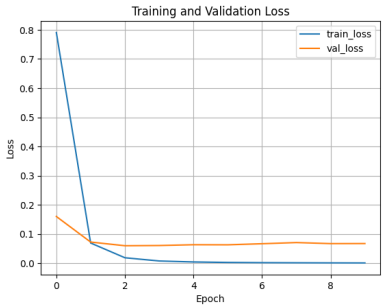

Figure 10: 16 neurones, kernel size = 3, F1-score = 0.84

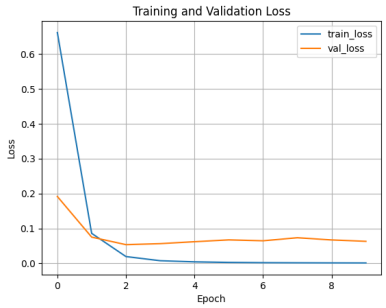

Figure 11: 32 neurones, kernel size = 3, F1-score = 0.84

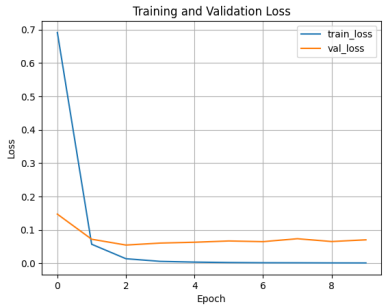

Figure 12: 64 neurones, kernel size = 3, F1-score = 0.84

Com es pot veure hem començat afegint una convolucional amb 16 neurones, hem anat afegint-ne més fins a 64 però al veure que el valor de F1-score ja no millorava al afegir més complexitat hem decidit quedar-nos amb la capa de 16 neurones per no afegir complexitat innecessària. Com podem veure, hem arribat a un F1-score de 0.84, per tant, afegir una segona capa efectivament a permès al model captar relacions més complexes entre les dades que han millorat el seu rendiment.

Donada la millora a l'afegir-ne una segona capa, procedirem a incorporar una tercera per veure si seguim millorant. Després de provar afegint una capa més de 16 neurones i una de 32, hem vist que arribem a F1-scores més baixos, així que hem decidit no afegir-la. Es poden veure els resultats al fitxer d'experiments.

A continuació, hem provat d'utilitzar diferents capes de pooling amb diferents tamanys de kernel però sempre amb stride 1 perquè les dimensions segueixin quadrant durant l'entrenament. Les avantatges que ens dona utilitzar capes de pooling són que podem reduir la dimensionalitat i capturar només aquelles característiques més importants del text. En el nostre cas al no poder reduir la dimensionalitat perquè utilitzem stride = 1 no ens aportarà grans beneficis. Durant l'experimentació hem provat amb les capes MaxPooling1D i AveragePooling1D amb diferents mides de pool size i en tots els casos hem empitjorat els resultats previs, de manera que no afegirem capa de pooling.

2.6 Task 3: Xarxes Recurrents

En aquest apartat del treball experimentarem amb l'ús de diferents tipus de xarxes recurrents i amb els seus paràmetres. Les xarxes recurrents són molt utilitzades en tasques de NER, ja que tenen propietats molt interessants. Una d'aquestes propietats és que tenen la capacitat de condensar la informació prèvia del context d'una paraula en un sol vector i utilitzar-lo per predir l'etiqueta de la paraula actual.

En concret en aquest apartat farem us de xarxes de tipus LSTM i GRU, les quals estan dissenyades per capturar dependències entre paraules més properes i també més llunyanes en el text. Això pot aportar-nos informació que les xarxes convolucionals no ens aportaven perquè només capturaven el context més local de les paraules. Per aquesta raó creiem que experimentar amb aquestes xarxes ens aportarà bons resultats sobretot en aquelles seqüències més llargues.

A continuació començarem el procés d'experimentació. Per començar, partirem d'una arquitectura similar als següents apartats, on utilitzarem una capa d'embeddings amb les dimensions marcades anteriorment, una capa amb una xarxa recurrent, una capa densa de 64 neurones i una de sortida amb 39 neurones. Cal destacar que per utilitzar aquestes capes a Keras ens ha calgut posar el paràmetre 'return_sequences' com a True perquè el model retorni una etiqueta per cada paraula de la frase d'entrada.

El primer experiment que hem realitzat ha estat afegir una única capa LSTM amb diferents nombres d'unitats LSTM. El nombre d'aquestes unitats influeix en la capacitat del model a l'hora d'aprendre patrons més o menys complexos en les frases d'entrada. Si en les nostres dades existeixen relacions molt complexes, el nostre model necessitarà més complexitat, sinó estarem afegint complexitat de més. En aquest cas nosaltres hem experimentat amb els valors 32, 64 i 128 i aquests han estat els resultats:

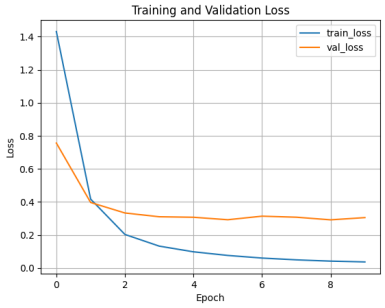

Figure 13: 32 neurones, F1-score = 0.75

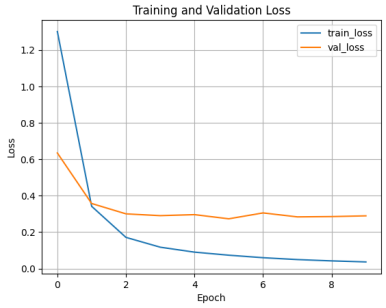

Figure 14: 64 neurones, F1-score = 0.75

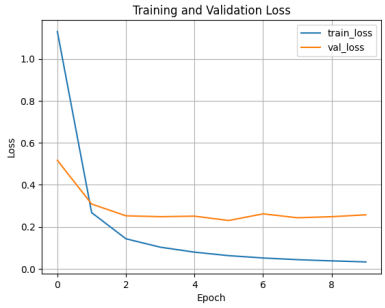

Figure 15: 128 neurones, F1-score = 0.74

Tal i com podem apreciar a mesura que afegim complexitat no estem millorant el valor de F1-score del model, per tant, hem decidit que utilitzarem únicament 32 unitats LSTM en la primera capa donat que en els altres casos estem afegint complexitat de més.

El segon experiment que hem realitzat ha estat fer el mateix que abans però amb xarxes GRU per veure si amb arquitectures menys complexes obtenim millors resultats.

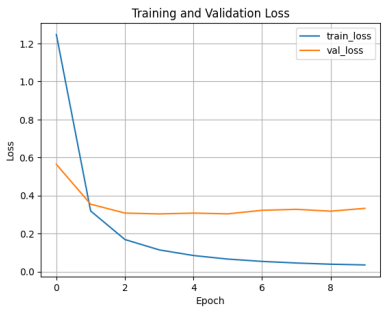

Figure 16: 32 neurones, F1-score = 0.73

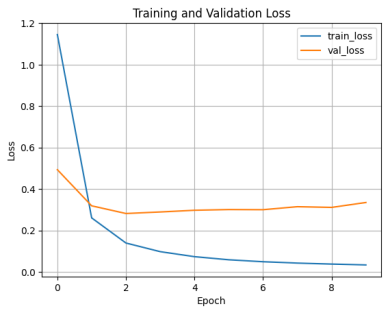

Figure 17: 64 neurones, F1-score = 0.75

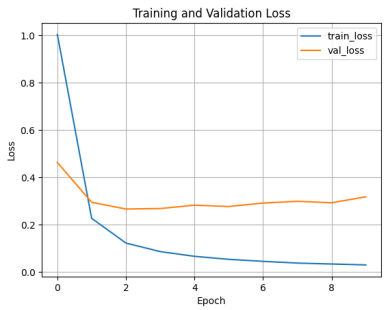

Figure 18: 128 neurones, F1-score = 0.75

En aquest experiment obtenim uns resultats molt similars als de les LSTM, en aquest cas arribem a 0.75 de F1-score si utilitzem 64 unitats de GRU. Donat que utilitzar 64 unitats de GRU fa que el nostre model tingui una complexitat de 1049255 paràmetres i utilitzar 32 unitats de LSTM fa que tinguem

1030567, aquesta primera serà una capa LSTM amb 32 unitats, ja que arribem als resultats més alts pel que fa el F1-score amb una complexitat menor.

A continuació provarem d'afegir una segona capa de LSTM, utilitzarem LSTMs perquè ens han funcionat millor en el primer experiment. El fet d'afegir més capes de xarxes recurrents pot ajudar al model a capturar patrons més abstractes i complexes en les dades. Per aquesta raó hem provat d'afegir una segona capa LSTM amb 16 i 32 unitats LSTM. Com a resultat d'aquesta experimentació hem obtingut un F1-score de 0.69 en el cas de 16 unitats i 0.70 en el cas de 32. Veiem com els resultats empitjoren molt respecte els de una capa així que hem decidit no afegir-ne cap més.

D'aquests 3 primers experiments podem destacar que els valors de F1-score són força baixos respecte a models entrenats prèviament. Creiem que això pot venir donat perquè les capes recurrents funcionen processant les oracions d'esquerra a dreta de manera seqüencial, això fa que a l'hora de predir una paraula només tenim accés al vector de context que conté informació de les paraules que han aparegut prèviament a la frase. En la tasca de NER és molt important tenir accés a tot el context, no només a aquell previ a la paraula. Per exemple, si nosaltres volem etiquetar l'entitat 'New York' en l'oració 'Yesterday I woke up in New York', per la xarxa, conèixer que després de la paraula 'New' ve 'York' aporta un coneixement molt important al model a l'hora de decidir com etiquetar l'entitat. Per aquesta raó, experimentarem amb xarxes bidireccionals, les quals processen les oracions d'entrada d'esquerra a dreta i a la inversa, solucionant el problema recent esmentat. A continuació, mostrarem els resultats d'utilitzar una capa bidireccional de LSTM amb 32,64 i 128 unitats. Cal destacar que també hem experimentat amb mida 16 però els resultats eren pitjors i per qüestió d'espai no ho hem afegit.

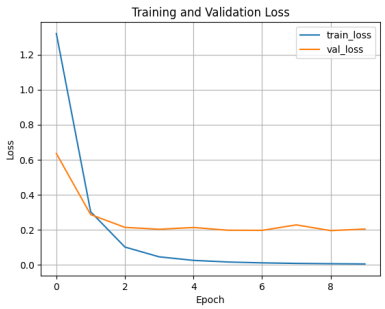

Figure 19: LSTM Bidireccional, 32 unitats, F1-score = 0.85

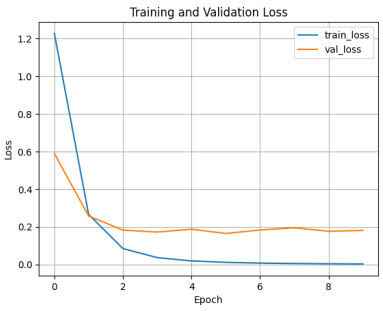

Figure 20: LSTM Bidireccional, 64 unitats, F1-score = 0.86

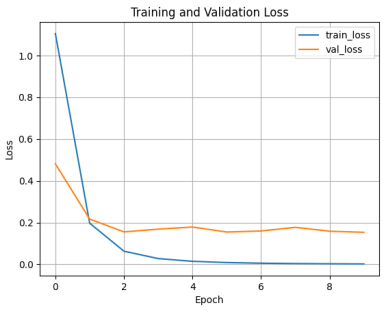

Figure 21: LSTM Bidireccional, 128 unitats, F1-score = 0.86

En aquest cas podem veure com arribem a F1-scores molt més alts, el fet de processar les oracions de dreta a esquerra ha permès al model capturar tot el context de les paraules i això s'ha vist reflectit en una millora en els resultats. Pel que fa les corbes de loss, en els 3 casos són molt similars i en cap d'ells fem overfitting. El millor resultat ha estat el segon, on usem 64 unitats dins de la capa bidireccional (64 + 64) i arribem a un F1-score de 0.86 amb menor complexitat que en el cas de 128 unitats.

Ara realitzarem el mateix experiment amb les capes GRU per veure si aquesta arquitectura funciona millor sobre les nostres dades ara que les pot processar en ambdós sentits. Aquests han estat els resultats.

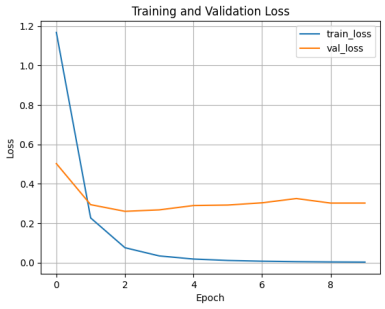

Figure 22: GRU Bidireccional, 32 unitats, F1-score = 0.87

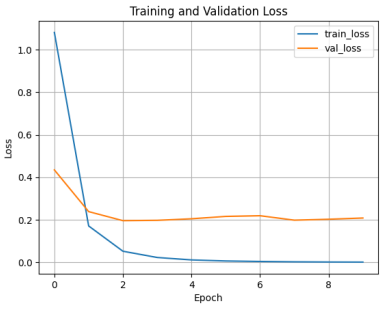

Figure 23: GRU Bidireccional, 64 unitats, F1-score = 0.88

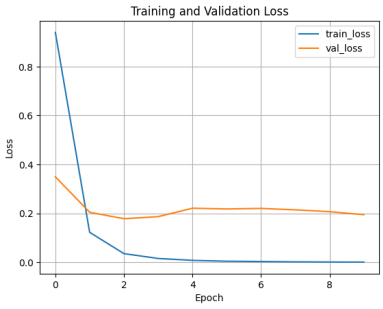

Figure 24: GRU Bidireccional, 128 unitats, F1-score = 0.87

Com podem veure, els resultats han estat molt positius arribant fins i tot a un F1-score de 0.88. Es interessant destacar com amb una loss de 0.2 arribem a un F1-score tan alt, això corrobora el que hem

estat veient al llarg dels experiments, una val_loss més baixa no sempre implica un millor resultat en la mètrica F1. Per tant, concluïm l'experimentació amb xarxes recurrents veient com el millor resultat s'ha donat amb una capa bidireccional GRU amb 64 unitats.

Volem destacar que donat que a les xarxes convolucionals arribàvem a una val_loss més baixa, hem volgut provar de fer un model híbrid on combinem el millor resultat de les convolucionals i el de les xarxes recurrents. El resultat d'aquesta experimentació no ha tingut èxit ja que hem arribat a un F1-score de 0.86, menor que el que obteníem únicament amb recurrents.

2.7 Task 4: Transformers

Un cop hem explorat com les xarxes convolucionals i recurrents funcionen per a la nostra tasca de NER, ara procedirem a experimentar amb transformers. Aquesta arquitectura es basa en el mecanisme d'atenció ('attention'), que és un mètode molt potent que ens permet comprendre el context global de les paraules d'una frase. Mitjançant un procés de "multi-head self attention", els transformers permeten determinar quines paraules en una oració d'entrada tenen més influència en altres quan es prediu la seva etiqueta.

En aquest procés d'experimentació partirem de l'arquitectura definida al baseline, on utilitzàvem una capa de TokenAndPositionEmbedding, un bloc de Transformer, una capa densa de 64 neurones i una densa d'output amb 39 neurones. Cal destacar que donada la complexitat d'aquesta arquitectura, reduirem el learning rate de l'Adam a 0.0005 per evitar fer overfitting immediatament. Seguidament, el que farem serà experimentar amb el nombre de heads i les dimensions de la xarxa feed forward que trobem dins del bloc de transformer. Tenir més nombre de heads pot ajudar al nostre model a focalitzar-se en diferents característiques importants del text d'entrada i per aquesta raó creiem que es interessant provar diferents valors. En el cas de les dimensions de la feed forward, es necessari provar diferents valors per veure quina complexitat s'adequa més a la nostra tasca. En el primer experiment hem provat les següents combinacions de num_heads i ff_dim: 2 i 32, 2 i 64, 2 i 128, 4 i 32, 4 i 64, 4 i 128. A continuació mostrarem els 3 resultats més interessants.

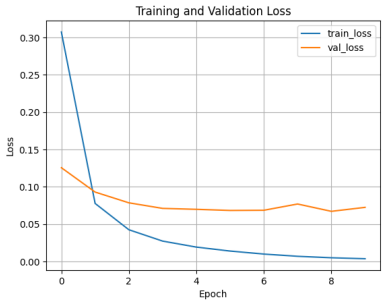

Figure 25: num_heads = 2 i ff_dim = 32, F1-score = 0.80

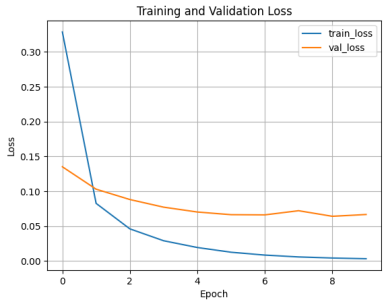

Figure 26: num_heads = 2 i ff_dim = 64, F1-score = 0.79

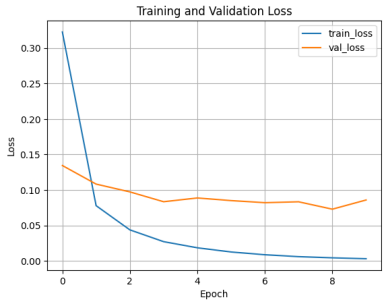

Figure 27: num_heads = 4 i ff_dim = 64, F1-score = 0.80

En les imatges es pot apreciar com gràcies a utilitzar transformers la loss baixa a menys de 0.1, encara que amb els paràmetres provats no hem pujat el F1-score de 0.80. El resultat que amb menor complexitat ha donat el valor més alt de F1-score, ha estat el primer de tots, el més simple, on hem utilitzat 2 heads i 32 neurones a la capa feed forward. Cal destacar que al final de la corba de loss comencem a fer overfitting així que caldrà tenir-ho en compte quan seguim afegint més complexitat.

A continuació, provarem d'afegir més blocs de transformers per veure si d'aquesta manera el model és capaç d'aprendre patrons més complexos de les dades. Aquesta millora de capacitat pot ser beneficiosa per capturar relacions i context intrínsec en el text, fet molt important a l'hora de reconèixer acuradament entitats nominals. En primer lloc, hem provat 3 combinacions diferents dels 2 primers blocs amb diferents ff_dim, 32 + 32, 32 + 64 i 64 + 64. En tots els blocs hem utilitzat num_heads = 2 perquè es el que millor ens ha funcionat al primer experiment. Aquests han estat els resultats:

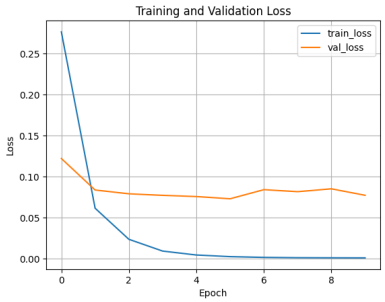

Figure 28: num_heads = 2, ff_dim1 = 32, ff_dim2 = 32 ,F1-score = 0.81

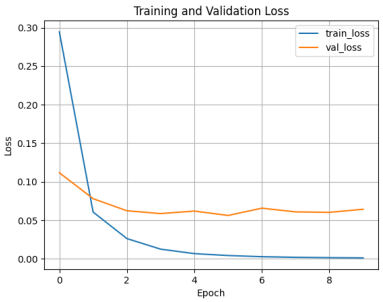

Figure 29: num_heads = 2, ff_dim1 = 32, ff_dim2 = 64 ,F1-score = 0.82

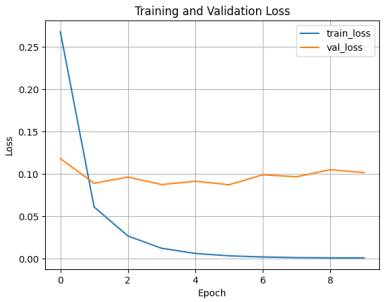

Figure 30: num_heads = 2, ff_dim1 = 64, ff_dim2 = 64 ,F1-score = 0.82

En els gràfics podem veure com millorem més en el segon experiment, quan afegim un segon bloc amb 64 neurones a la capa feed forward. A més, podem veure com en el tercer experiment, al fer que els dos blocs tinguin una feed forward amb 64 neurones apareix overfitting.

Com hem millorat, provarem d'afegir un tercer bloc sobre l'experiment anterior on hem obtingut els millors resultats. Provarem tant valors de 32 com de 64 neurones de la feed forward en aquest tercer bloc. Aquests han estat els resultats.

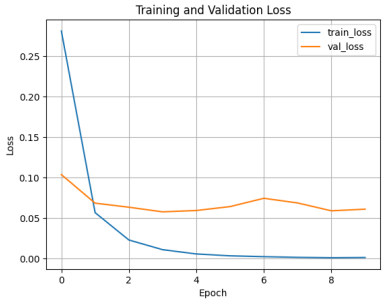

Figure 31: ff_dim3 = 32 ,F1-score = 0.82

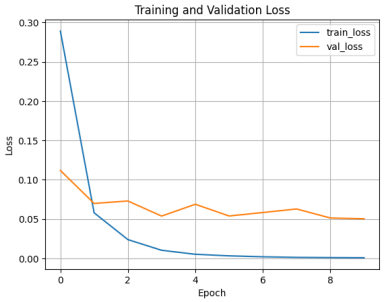

Figure 32: ff_dim3 = 64 ,F1-score = 0.86

Podem apreciar com en el segon experiment arribem a una val_loss molt baixa (0.05) sense fer overfitting i a un F1-score de 0.86. Posteriorment, hem provat d'afegir un quart bloc de transformers però els resultats empitjoraven. Com a conclusió, utilitzar num_head = 2 i tres blocs amb 32, 64 i 64 dimensions a la feed forward ens a donat un F1-score de 0.86.

Finalment, donat que ara hem fet més complexa l'arquitectura, provarem d'incrementar la mida dels embeddings i el nombre de heads per veure si amb una representació més extensa de la informació i una arquitectura més complexa podem millorar els resultats vists fins ara. Aquests han estat els resultats:

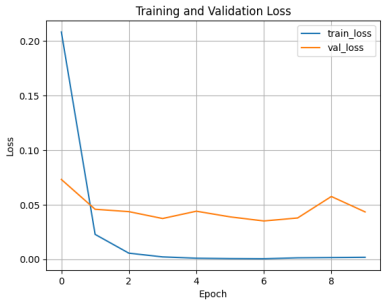

Figure 33: num_heads = 4, embedding_dim = 256 ,F1-score = 0.88

Figure 34: num_heads = 8, embedding_dim = 512, F1-score = 0.87

Podem veure com en el primer experiment, efectivament hem millorat el resultat anterior i hem arribat a un F1-score de 0.88. De la corba del primer experiment podem destacar que aconseguim arribar a losses per sota de 0.05 i que en les últimes epochs comencem a fer overfitting. En el segon experiment

arribem a pitjors resultats. Per tant, podem concloure que el nostre model amb 3 blocs de tranformers amb embedding size de 256 i 4 heads es el millor que hem trobat fins ara junt a les GRUs bidireccionals.

2.8 Task 5: Afegim Dropout

Tal i com hem pogut veure en l'últim experiment, el nostre millor model fa overfitting en les ultimes èpoques així que per evitar això, experimentarem sobre ell afegint tècniques de regularització com el Dropout que ens proporciona la llibreria de Keras. El Dropout desactiva de manera aleatòria algunes neurones durant l'entrenament, evitant que el model depengui excessivament d'un subconjunt particular de neurones. Amb la incorporació d'aquestes tècniques de regularització, es porta al model a generalitzar millor sobre dades no vistes, i per tant, esperem que també a millorar el F1-score sobre les dades de validació.

En el procés d'experimentació començarem provant valors de dropout més petits entre les primeres capes del model i anirem afegint més capes de dropout i augmentant o no els valors en funció de com reaccioni el nostre model. En el primer experiment hem provat afegint dropouts de 0.1, 0.2 i 0.3 després de la capa de TokenAndPositionEmbedding i aquests han estat els resultats.

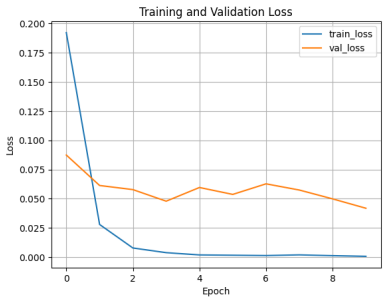

Figure 35: Dropout = 0.1, F1-score = 0.86

Figure 36: Dropout = 0.2, F1-score = 0.88

Figure 37: Dropout = 0.3, F1-score = 0.89

Podem veure com amb dropout 0.1 empitjorem el resultat, amb 0.2 la corba de validació té molt de soroll i seguim fent overfitting, però a l'utilitzar dropout 0.3 el nostre model sembla que aprèn a generalitzar millor i s'incrementa el F1-score a 0.89. Donat que al afegir més dropout milloren els resultats anem a realitzar un altre experiment afegint-ne dropout també després del primer bloc de transformer.

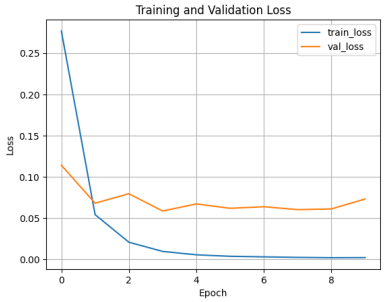

Figure 38: Dropout = 0.3 + 0.1, F1-score = 0.86

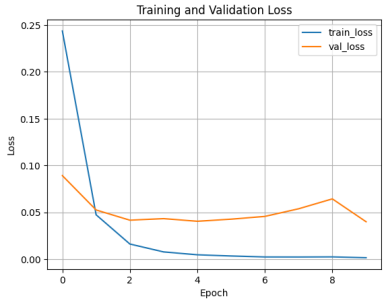

Figure 39: Dropout = 0.2 + 0.2, F1-score = 0.88

Podem veure com al repartir més els dropouts les corbes tenen menys soroll, això ho atribuïm a que estem generalitzant millor. Malgrat així, no millorem els resultats i en les últimes èpoques segueix havent overfitting així que provarem d'afegir més dropouts després del segon bloc i tercer bloc de transformer. Aquests han estat els experiments i els resultats.

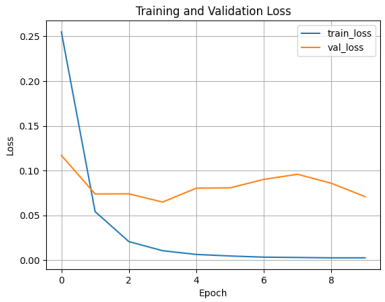

Figure 40: Dropout = 0.2 + 0.2 + 0.1, F1-score = 0.90

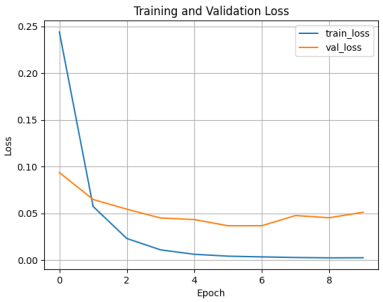

Figure 41: Dropout = 0.2 + 0.2 + 0.1 + 0.1, F1-score = 0.90

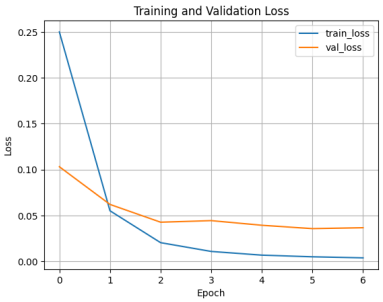

Figure 42: Dropout = 0.2 + 0.2 + 0.1 + 0.1, F1-score = 0.90, 7 èpoques

En aquests experiments hem començat afegint una tercera cap de dropout i hem pujat el F1-score a 0.9, posteriorment, al veure que seguien millorant els resultats hem afegit una quarta capa i ens hem trobat que el F1-score no pujava però teníem una corba de val_loss molt més estable i que arribava a una val_loss propera a 0.03 a la època 6 i després feia una mica d'overfitting. Per solucionar això, vam decidir fer l'experiment definitiu de la figura 42, on utilitzem 4 capes de dropout i limitem l'entrenament a 7 èpoques. Com a resultat, es pot apreciar que no fem overfitting i aconseguim ajustar el nostre model fins a una val_loss de 0.0366, malgrat això el nostre F1-score no millora i es queda a 0.90. Considerem aquest últim experiment el millor que hem vist fins ara perquè l'entrenament es estable i a més arribem a resultats molt positius.

2.9 Task 6: Balancegem les classes

En aquest apartat farem un petit anàlisi de la distribució de les classes en les nostres dades, posteriorment calcularem els pesos que li volem atribuir a cadascuna i finalment li passarem aquests pesos al nostre model com a paràmetre class_weights.

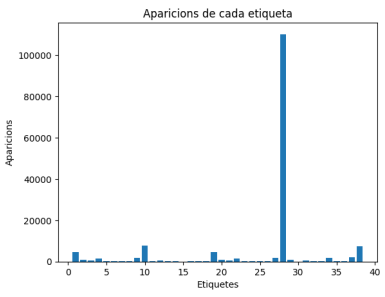

Figure 43: Train label count

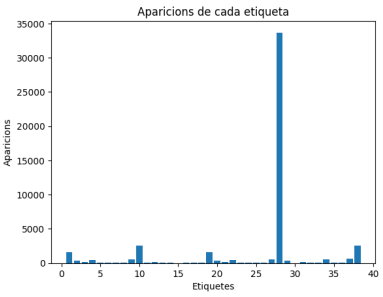

Figure 44: Val label count

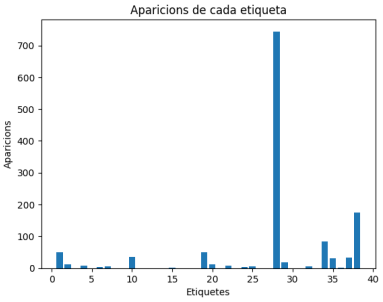

Figure 45: Test label count

Podem veure un clar desbalanç en les 3 particions encara que nosaltres de moment només ens fixarem en el del train. A continuació hem implementat una funció que hem anomenat 'compute_class_weights' i que calcula la freqüència inversa de cada classe, d'aquesta manera el model donarà menys importància a aquelles classes que apareguin més i més a aquelles que apareguin menys. Per tant, les classes 0 (padding) o 28 ('O') se'ls hi assignarà valors molt petits.

Després de calcular els pesos hem procedit a entrenar el nostre millor model amb aquests paràmetres i aquests ha estat el resultats.

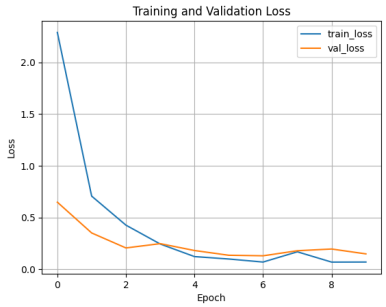

Figure 46: F1-score = 0.73. Resultat amb class_weights

Podem veure com el nostre millor model no millora a l'afegir aquest paràmetre. Això segurament estigui succeint perquè tots els canvis fets durant el procés d'elaboració han estat pensats per mitigar el desbalanç de classes. Ara si volguéssim millorar els resultats podríem construir un altre model començant a experimentar des del principi amb el paràmetre class_weights.

3 Prediccions a test

Per comprovar el comportament dels millors models trobats amb dades completament desconegudes hem realitzat les prediccions de test i em obtingut els següents F1-scores.

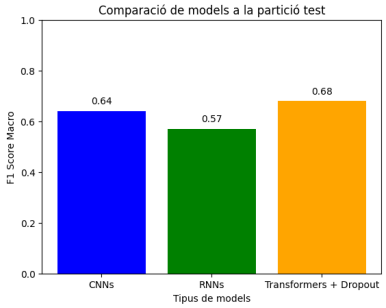

Figure 47: Imatge per comparar resultats al test dels nostres millors models en cada apartat

Com podem veure són valors molt baixos en comparació amb els de validació. Això ho atribuïm a que la distribució de classes en la partició de test difereix força de les trobades a validació, com podem veure en la figura 45.

4 Conclusions

La conclusió principal que podem extreure'n d'aquest treball és que la forma en la que capturem el context de les seqüencies de paraules és molt important. Els transformers han demostrat ser els models que millor resultats han donat en aquesta tasca gràcies a l'ús d'attention per captar el context. També hem pogut veure grans millores a les xarxes recurrents a l'afegir bidireccionalitat i poder conèixer el context total de les oracions. Finalment, pel que fa les xarxes convolucionals hem vist que en tasques de NER amb oracions curtes funcionen prou bé utilitzant kernels de mida 3 i 4, això en altres tasques on és més important el context més global no obtindríem tan bons resultats.

Per acabar, aquesta és l'arquitectura final del nostre millor model, corresponent a l'experiment de la figura 42.

```
Embedding Dim = 256
num_heads = 4
ff_dim1 = 32
ff_dim2 = 64
ff_dim3 = 64
ff_dim4 = 64
Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 token_and_position_embeddi  (None, 19, 256)           2015488
 ng_10 (TokenAndPositionEmb
 edding)
 dropout_54 (Dropout)        (None, 19, 256)           0
 transformer_block_22 (Tran  (None, 19, 256)           1069600
```

```
 sformerBlock)
 dropout_57 (Dropout)        (None, 19, 256)           0
 transformer_block_23 (Tran  (None, 19, 256)           1086016
 sformerBlock)
 dropout_60 (Dropout)        (None, 19, 256)           0
 transformer_block_24 (Tran  (None, 19, 256)           1086016
 sformerBlock)
 dense_70 (Dense)            (None, 19, 64)            16448
 dense_71 (Dense)            (None, 19, 39)            2535
=================================================================
Total params: 5276103 (20.13 MB)
Trainable params: 5276103 (20.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```