Universitat Autònoma de Barcelona
Facultat de Ciències

Matemàtica Computacional i Analítica de Dades

TREBALL FI DE GRAU

COMPARACIÓ DE MODELS BASATS EN TRANSFORMERS

Autora: Júlia Pumares Benaiges

Tutor: Antoni Lozano Bagen

Barcelona, Juny 2023

Índex

1 Introducció

1.1 Motivació

En els últims anys el processament de llenguatge natural (PLN) ha experimentat molts avenços gràcies al desenvolupament de nous models i tècniques. Entre aquests models destaquen els basats en l'arquitectura Transformer que han aconseguit resultats sorprenents en una amplia gama de tasques de PLN. Aquests models han revolucionat aquest camp al ser capaços de captura tota la informació contextual en un text. Una de les tasques de PLN és la classificació de textos que és útil per varies aplicacions, des de detecció de sentiments a la classificació de noticies o a la detecció de spam.

Aquest treball té com a objectiu principal realitzar una comparació de varis models basats en transformers, centrant-nos en la seva aplicabilitat i rendiment en la tasca específica de classificació de textos. S'estudiarà l'estat de l'art dels transformers i els diferents models. També s'avaluaran en comparació amb altres solucions més tradicionals d'aprenentatge automàtic i aprenentatge profund.

La motivació d'aquest treball recau en la necessitat d'entendre i comparar amb profunditat els diferents models basats en transformers, alhora que també es comparen amb altres models més tradicionals Al realitzar aquest treball s'espera obtenir una visió més clara sobre com utilitzar aquests models en la classificació de textos, així com comprendre el seu impacte en termes d'execució i precisió.

1.2 Objectius

Els objectius d'aquest treball són els següents:

- Estudiar l'estat de l'art dels transformers.
- Analitzar l'estructura dels transformers.
- Fer servir models basats en transformers per resoldre una tasca de processament de llenguatge natural (PLN), concretament la classificació de textos.
- Comparar diferents models basats en transformers (el temps d'execució i la precisió) en aquesta tasca.
- Comparar aquest models amb models més bàsics d'aprenentetge computacional i profund.
- Utilitzar l'entorn de treball de Google Colab per poder fer servir els models pre-entrenats.

2 Marc teòric

2.1 Processament de llenguatge natural

El Processament de Llenguatge Natural (PLN) és una branca de la intel·ligència artificial que es centra en l'estudi de com els ordinadors entenen, interpreten i processen el llenguatge humà. L'objectiu principal que té és desenvolupar algoritmes i models capaços de processar, analitzar i generar llenguatge humà en diferents formes com text, veu i escriptura a mà. Amb l'increment de la quantitat de dades de text disponibles a Internet, PLN és una eina essencial per varies aplicacions com ara resumir textos, traducció automàtica de textos, classificació de textos, generació de text, xatbots i assistents de veu. El PLN tracta l'anàlisi lèxica, l'anàlisi morfològica, l'anàlisi sintàctica, i la interpretació semàntica per tal d'aconseguir els seus objectius, encara que la majoria de les aplicacions se centren en algunes d'elles i no les tracten totes amb profunditat.

2.1.1 Reptes del PLN

El llenguatge natural és complex, divers, desorganitzat i ambigu, la qual cosa dificulta que els models el pugui processar. A més a més, el llenguatge natural depèn del context i pot ser sarcàstic, irònic i metafòric. Les principals dificultats que pots trobar una màquina per comprendre el llenguatge són les següents:

- **Ambigüitat lèxica:** una paraula pot tenir diferents significats.
- **Ambigüitat sintàctica:** fa referència a una seqüència de paraules que pot tenir més d'un significat.
- **Ambigüitat referencial:** una paraula o frase pot referir-se a una o més propietats.
- **Ambigüitat pragmàtica:** a vegades una frase no expressa literalment el que sembla dir. En la ironia o el sarcasme, les paraules poden tenir un significat positiu o negatiu per definició, però s'utilitzen per crear l'efecte contrari.

Aquests són alguns exemples dels reptes que el PLN ha d'afrontar en la comprensió i interpretació del llenguatge natural. Les màquines han de ser capaces de superar aquestes dificultats per aconseguir una comunicació més precisa i efectiva amb els éssers humans.

2.1.2 Mètodes per resoldre problemes PLN

Hi ha dues maneres principals de com abordar un problema d'aquest tipus:

Codificacions manuals: aquest enfocament implica codificar manualment les regles gramaticals i lingüístiques del llenguatge que serveixen per analitzar i comprendre el text. Aquest mètode es bastant complicat i tediós ja que es necessiten totes les regles per totes les ambigüitats i varietats del llenguatge.

Aprenentatge automàtic: aquest enfocament es basa en algoritmes d'aprenentatge automàtic que permeten a les màquines aprendre a partir d'exemples i dades d'entrenament.
Els models de PLN funcionen trobant relacions entre les parts del llenguatge, per exemple, paraules, lletres i frases contingudes en una base de dades de text. Abans de poder aplicar un model a les dades primer cal transformar el text en un format que les màquines puguin interpretar. I també cal extreure característiques dels textos que es pugui passar després a un model. Algunes de les tècniques que es poden aplicar per pre-processar un text són:

- **Segmentació de les frases:** Dividir un paràgraf en frases. En molts llenguatges és fàcil de fer a partir dels signes de puntuació però en altres llengües, com per exemple, Xinès antic no es tan fàcil perquè no tenen un delimitador que separi les frases.
- **Tokenització:** Descompondre les frases en paraules o *tokens*, és útil per entendre el context i les components individuals.
- ***Stemming*:** aquesta tècnica redueix les paraules a la seva arrel, traient els sufixos i prefixos, d'aquesta manera es redueix la complexitat del vocabulari. Per exemple les paraules *finally* i *final* és reduirien a *fina*.
- ***Lemmatitzation*:** similar al stemming però fa servir un diccionari per reduir les paraules a l'arrel. és més acurat que stemming però també és més lent. La diferència és que aquí obtenim una paraula que existeix. Per exemple *runs*, *running* i *ran* es reduirien a *run* .
- **Eliminació de *Stop words:*** són un conjunt de paraules que apareixen més freqüentment que les altres i que s'eliminen del text perquè no aporten gaire informació i així es pot focalitzar en les altres paraules. Generalment són articles, preposicions com ara: *a, and, the...*
- ***Dependency parsing:*** consisteix en identificar les relacions gramaticals entre les paraules d'una frase, quines paraules són subjectes, objectes o modificadors d'altres paraaules. Es pot crear una arbre que representa l'estructura de la frase.
- ***Part-of speech tagging (POS):*** consisteix en etiquetar cada paraula amb la seva part del discurs, com ara si es un nom, verb o adjectiu.

Algunes de les tècniques per extreure característiques són:

- **Models de sac de paraules (*Bag-of-Words*):** compta el nombre de vegades que surt cada paraula o n-gram (conjunt de n paraules) en el document. Cada document és representat com un vector de característiques basat en la freqüència de cada paraula.
- **TF-IDF:** a cada paraula se li aplica un pes segons la seva importància. El *Term Frequency* (TF) ens diu com d'important és la paraula al document i es calcula com el nombre d'ocurrències de la paraula al document entre el nombre total de paraules del document. El *Inverse Document*

Frequency (IDF) ens diu com d'important és el terme en tot el conjunt de dades. Es calcula com el logaritme del nombre de documents al corpus entre el nombre de documents que contenen la paraula. El TF-IDF és la multiplicació de TF i IDF.

- **Word2Vec:** és un model que utilitza una xarxa neuronal per aprendre associacions entre les paraules d'un corpus de text. Representa cada paraula com un vector. Aquests vectors contenen informació semàntica i sintàctica sobre la paraula. La funció matemàtica similitud del cosinus s'utilitza per indicar el nivell de similitud semàntica entre dues paraules.

- **GLoVE:** és un model similar a l'anterior ja que també aprèn relacions entre paraules, però ho fa utilitzant tècniques de factorització de matrius. Es construeix una matriu basada en el nombre de co-ocurrències entre les paraules.

Després d'haver pre-processat les dades, es passen a una arquitectura NLP que les modela per resoldre una tasca concreta. Les característiques extretes es poden fer servir per entrenar varis models d'aprenentatge computacional com ara *logistic regression*, *Naive Bayes*, *decision trees*, o *Support Vector Machines*. Amb els models d'aprenentatge profund normalment no cal utilitzar les característiques extretes sinó que es pot passar directament el text, tot i que també es poden utilitzar com a entrada del model.

2.1.3 Classificació de Textos

La classificació de textos és una tasca de NLP que consisteix en assignar una categoria a un text. Els classificadors de textos es poden utilitzar per organitzar, estructurar i categoritzar textos des de documents fins a registres mèdics. Les aplicacions principals de la classificació de textos són:

- **Anàlisi de sentiment:** consisteix en analitzar l'emoció que conte un text i classificar-lo com a positiu, neutre o negatiu. És útil en xarxes socials, ressenyes de productes, enquestes, feedback de consumidors, ja que els negocis poden obtenir *insigths* de com els consumidors valoren els productes.

- **Classificació per tema:** consisteix en identificar els temes o tòpics principals d'un text i assignar-los categories predefinides.

- **Detecció de la intenció:** consisteix en analitzar la intenció, propòsit o objectiu darrere d'un text. Com per exemple detecció de spam.

2.2 Aprenentatge computacional

En aquesta secció veurem per sobre alguns models d'aprenentatge computacional i com s'apliquen per una tasca de classificació.

- ***Decision Tree:*** És un model que utilitza una estructura d'arbre per prendre decisions basades en les característiques de les dades d'entrada. En cada node de l'arbre, es realitza una prova en una característica específica i es divideix el conjunt de dades en funció del resultat d'aquesta prova. Aquest procés es repeteix fins que s'arriba a una decisió final (a quina classe pertany l'entrada).

- ***Random Forest:*** És un conjunt d'arbres de decisió (*decision trees*). Cada arbre en el conjunt és entrenat amb una mostra aleatòria de les dades i després es realitza una votació per decidir la classe final de la predicció. Això ajuda a reduir la variància i a millorar el rendiment general del model.

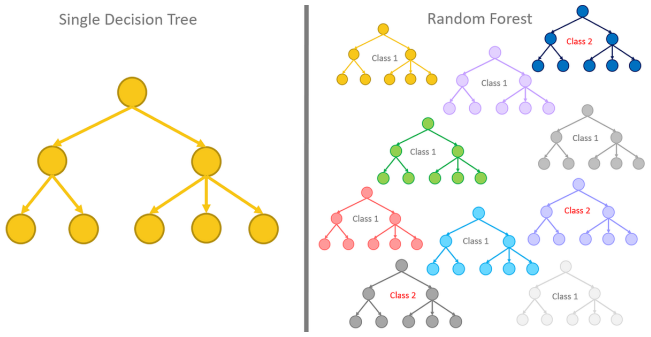


Figure 1: Decision Tree i Random Forest [4]

- ***Logistic Regression:*** aquest algorisme prediu les probabilitats de pertànyer a una classe específica. Utilitza una funció logística (sigmoide per classificació binària i Softmax per classificació multiclasse) per transformar una combinació lineal de les dades d'entrada en una probabilitat entre 0 i 1. La sortida del model serà un vector de probabilitats per a cada classe, i la classe amb la probabilitat més alta serà seleccionada com a predicció final.

- ***Linear SVC:*** *Support Vector Classifier* determina el millor hiperplà que divideix les dades en els grups predefinits. L'algoritme crea múltiples hiperplans en l'espai de característiques i s'intenta trobar el que millor separa les classes, que és el que té la màxima distància amb les punts de les diferents classes que es volen separar.

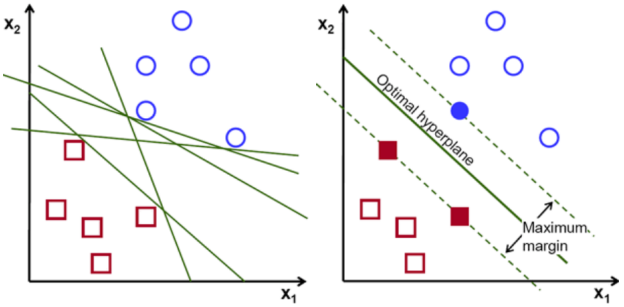


Figure 2: Hiperplans lineals entre dues classes [5]

2.3 Aprenentatge profund: Xarxes neuronals

Una xarxa neuronal artificial és un model computacional inspirat en les xarxes neuronals biològiques del cervell humà, intenta imitar la manera en que les neurones biològiques s'envien senyals entre elles. Aquestes xarxes artificials estan formades per capes de neurones: la capa d'entrada d'entrada, una o més capes ocultes i la capa de sortida. Cada neurona artificial calcula la seva sortida (y) fent la suma ponderada de les dades d'entrada (x_i) amb els pesos(w_i) que determinen la importància de cada entrada. A aquesta suma se li aplica una funció d'activació (α), si la suma dona un valor per sobre el llindar de la funció d'activació aquesta neurona s'activa i per tant envia informació a la següent neurona. És a dir, la funció d'activació decidirà si l'entrada de la neurona és important pel proces de predicció de la xarxa.

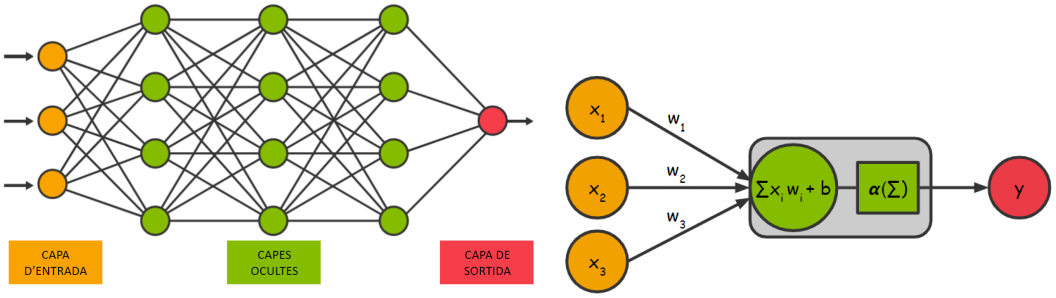


Figure 3: Xarxa Neuronal Artificial [7]

Hi ha molta varietat de funcions d'activació, algunes de les més comuns són:

- **Lineal:** L'activació és proporcional a l'entrada, retorna el valor que entra (no fa res).
- **ReLu:** Si l'entrada és més petita que 0 la neurona no s'activarà.
- **Sigmoide:** Porta l'entrada a un rang entre 0 i 1. Com més gran sigui el valor més s'aproparà a 1.
- **Tanh:** Similar a la sigmoide amb la diferència que el rang de sortida va de -1 a 1.

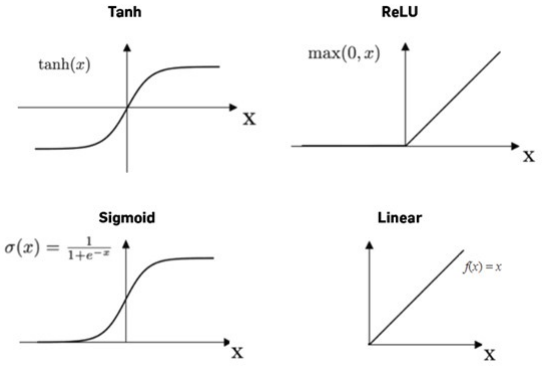


Figure 4: Funcions d'activació [6]

Les xarxes artificials es poden entrenar mitjançant un conjunt de dades format per entrades i les sortides esperades corresponents. Durant l'entrenament la xarxa ajusta els pesos i així pot generalitzar a partir de les dades rebudes. A partir d'aquest aprenentatge la xarxa es capaç d'extreure conclusions d'un conjunt d'informació complexa i aparentment no relacionada i pot fer prediccions o classificar noves dades no vistes anteriorment amb precisió. Existeixen moltes arquitectures i morfologies de xarxes neuronals diferents, i cadascuna s'especialitza en una tasca concreta. Per exemple les xarxes neuronal recurrents (RNN) per dades seqüencials o les xarxes neuronals convolucionals (CNN) per imatges.

2.3.1 Xarxes Neuronals Recurrents

Les Xarxes Neuronals Recurrents (RNN) són un tipus de xarxes neuronals que s'utilitzen per modelar dades seqüencials, com ara series temporals o llenguatge natural, en les quals l'ordre és important. S'anomenen recurrents ja que la informació es passa de manera iterativa d'un pas de la xarxa a l'altre. Les sortides anteriors s'utilitzen com a entrades de les següents neurones. D'aquesta manera permeten que la xarxa tingui una memòria artificial i pugui recordar i oblidar informació que ha vist anteriorment relacionant les noves entrades amb les entrades anteriors. La neurona recurrent rep en cada instant de temps una entrada (x_t) i la sortida del pas anterior.

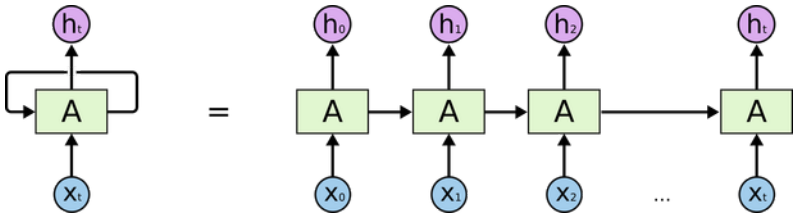


Figure 5: Xarxa Neuronal Recurrent [7]

En la figura de sobre veiem una representació d'una neurona recurrent i aquesta neurona desenvolupada en una xarxa completa. En la figura X_t representa l'entrada de la neurona a temps t i A és l'estat amagat al temps t (la memòria de la xarxa).

RNN bidireccionals

Les xarxes bidireccionals són les que processen el text d'inici a fi i al revés, de fi a inici En tasques de generació de text es útil tenir context al voltant de la paraula, no només les paraules anteriors.

Long-Short-term-Memory Recurrent Neural Network (LSTM)

Les RNN tenen el problema del *vanishing gradient descent*, els gradients que s'utilitzen per actualitzar els pesos durant el pas de *backpropagation* es van fent molt petits i això fa que no s'actualitzin els pesos de la xarxa correctament. Com a conseqüència la xarxa perd memòria, les paraules del principi tenen un pes molt petit, i per tant al final d'una frase la xarxa ja no es tenen en compte paraules del principi.

LSTM soluciona aquest problema introduint una arquitectura més complexa que permet recordar i oblidar de manera selectiva la informació dels passos anteriors. Una neurona LSTM està formada per tres portes que regulen el flux d'informació a la neurona i per dos estats: l'estat de la neurona que és la memòria a llarg termini (c_t) i l'estat ocult de la neurona que és la memòria a curt termini (h_t).

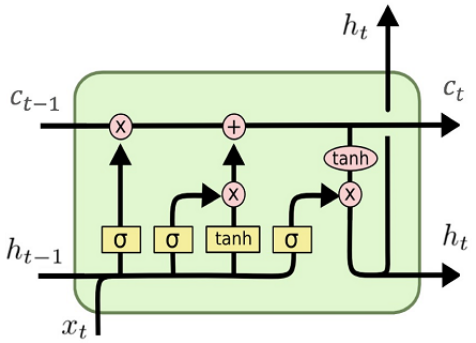


Figure 6: Neurona LSTM [10]

- **Porta d'oblit**: determina quina informació s'ha de retenir i quina oblidar del pas anterior. Rep com a entrada la informació de l'estat ocult anterior (h_{t-1}) i la informació de la neurona actual (x_t). Es passen per una funció sigmoide (σ) de manera que els valors propers a 0 es descarten i els altres continuen endavant.

- **Porta d'entrada** determina quina informació de la nova entrada guardarem a l'estat de la neurona (c_t). Les entrades són h_{t-1} i x_t i es passen per dues funcions d'activació: una sigmoide i una tanh. La tanh serveix per regular la xarxa i reduir el biax.

- **Porta de sortida**: determina la quantitat de l'estat de la neurona que hauria de ser part de la sortida. h_{t-1} i x_t es passen per una funció d'activació sigmoide. El nou estat de la neurona es passa per una tanh.

Gated Recurrent Unit (GRU)

GRU també resol el problema del *vanishing gradient*. La diferència amb LSTM és que combina la porta d'entrada i la d'oblit en un única porta d'actualització i ajunta l'estat de la neurona i l'estat ocult en un únic estat ocult. El resultat és una arquitectura més simple que ha de fer menys operacions amb tensors i això permet entrenar més ràpidament que la LSTM.

- **Porta d'actualització**: es responsable de la memòria a llarg termini, actua similar a la porta d'entrada i d'oblit de la LSTM. Controla quina quantitat de la nova entrada s'hauria d'afegir a l'estat ocult i quina quantitat de l'estat ocult anterior s'hauria de retenir. Aquesta porta està controlada per una funció d'activació sigmoide i retorna un valor entre 0 i 1 que determina el grau en que l'entrada i l'estat previ es combinen.

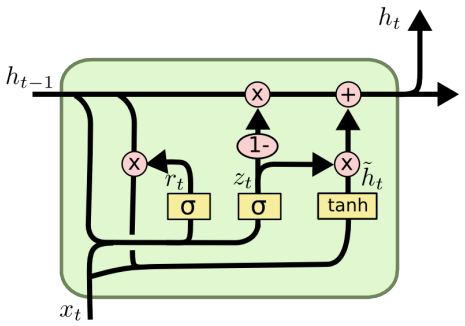


Figure 7: Neurona GRU [10]

- **Porta de reinici**: es responsable de la memòria a curt termini, determina quina quantitat de l'estat ocult anterior s'hauria d'oblidar, també té una sigmoide que indica el grau en que l'estat previ s'hauria d'oblidar.

2.4 Transformers

Els transformers van ser introduïts l'any 2017 en l'article *Attention Is All You Need* (L'atenció és tot el que necessites) per l'equip de Google Brain on proposaven aquesta nova arquitectura dissenyada per processar informació seqüencial com ara el llenguatge natural o imatges. Està basada en el mecanisme d'atenció, el qual permet modelar dependències entre totes les unitats d'una seqüència independentment de la distància a la que es troben entre elles dins de la seqüència.

Els transformers han permès solucionar les dues limitacions que presentaven les RNNs i CNNs, que són les dificultats a l'hora de paral·lelitzar-les i les dificultats per capturar dependències a llargues distàncies. Per una banda, els Transformers processen tot el contingut alhora (no seqüencialment) i això permet paral·lelitzar l'entrenament, cosa que no era possible amb les RNN ja que cada sortida depenia de l'anterior. Per altra banda, les RNN tenen memòria a curt termini: quan tenim una seqüència molt llarga RNN no pot accedir a les paraules generades a l'inici de la seqüència. LSTM i GRU tenen una memòria a més llarg termini cosa que fa que tinguin una finestra de referencia mes gran però tot i així segueixen sense funcionar bé per frases molt llargues. El mecanisme d'atenció del transformers, donats els recursos computacionals necessaris, té una finestra de temps de referència infinita, és a dir, és capaç d'utilitzar tot el context d'un document o text.

2.4.1 Arquitectura

El transformer té un tipus d'arquitectura de codificador-descodificador que podem observar en la següent imatge:

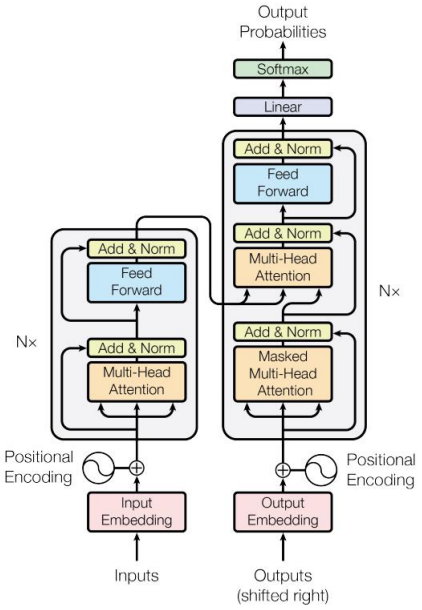


Figure 8: Arquitectura Transformer [1]

A la part esquerra tenim el codificador que rep una seqüència d'entrada i construeix una representació que conté tota la informació apresa d'aquella entrada. I a la dreta trobem el descodificador que fa servir aquesta representació i una altra entrada per, pas a pas, generar una sortida.
Anem a entrar en més detall en cada bloc del transformer:

Input embbedings / Output embbedings

La incrustació de paraules (*word embbeding*) consisteix en donada una seqüència d'entrada (un text) vectoritzar-la, és a dir, hem d'obtenir per cada paraula una representació com a vector amb valors continus. Aquest vector codifica el significat de la paraula, dos vectors que siguin propers en l'espai vectorial representaran dues paraules amb significats propers, per exemple, paraules de la mateixa família o del mateix camp semàntic.

Positional encoding

Aquest pas serveix per afegir informació posicional a les vectoritzacions de cada paraula. Les funcions que s'utilitzen per afegir aquesta informació són:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

- pos: és la posició de la paraula a la seqüència.
- i: és la dimensió de l'*embedding* del Transformer.

- d_{model} és la dimensió total de l'*embedding* del Transformer.

De manera que per les paraules en posicion senars es crea un vector utilitzant la funció sinus. I pels parells s'utilitza la funció cosinus. El vector que codifica la posició es suma a les vectoritzacions d'entrada. Amb aquestes funcions s'aconseguix aportar informació de la posició a la que es troba cada vector.

CODIFICADOR

Està composat per 2 submòduls: el mecanisme d'atenció (*Multi-head attenton*) seguit d'una xarxa neuronal d'avançada cap endavant.

Multihead attention

La *Multi-head attenton* és un mòdul del transformer que combina el resultat de varis mecanismes d'atenció. El mecanisme d'atenció associa cada paraula de l'entrada a la resta de paraules de la seqüència.

Per calcular l'atenció s'utilitzen tres vectors:

- **Vector Clau (Key - K):** representa l'element pel qual volem calcular els pesos d'atenció. Ens proporciona informació sobre l'element.
- **Vector Consulta (Query - Q):** es pot entendre com un vector cerca, per cada paraula indica una descripció del que busca.
- **Vector Valor (Value - V):** representa la informació asociada a cada element de la entrada.

La sortida es calcula com la suma ponderada dels valors on el pes assignat a cada valor (vector valor) es calcula mitjançant la compatibilitat de la cerca (vector consulta) amb la clau corresponent (vector clau). Això correspon a la següent formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

1. Multipliquem amb un producte vectorial la *query* i la *key* obtenint una matriu de puntuacions que determina quanta atenció es posa en cada paraula.
2. Escalem aquesta matriu dividint-la per l'arrel de la dimensió de la *query* i la *key*. Es fa per obtenir gradients més estables.
3. Apliquem una funció *Softmax* per obtenir els pesos d'atenció normalitzats amb valors entre 0 i 1.
4. Multipliquem la matriu de puntuacions normalitzades pel vector *value*.

D'aquesta manera hem aconseguit calcular els pesos d'atenció per la seqüència d'entrada i produir un vector amb la informació codificada de l'atenció que ha de posar cada paraula en cadascuna de la resta de paraules de la seqüència.

Aquest vector d'atenció es suma a la vectorització original d'entrada i posició. Això es el que s'anomena

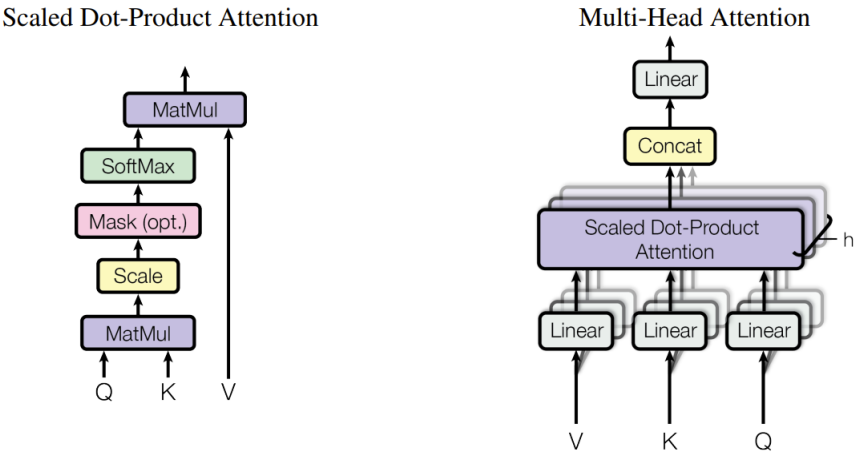


Figure 9: Estructura Multi-head attention [1]

conexió residual, que passa per una capa de normalització. El residu normalitzat passa per una xarxa neuronal propagació cap endavant formada per dues capes lineals amb una funció d'activació ReLu entre mig. Seguidament s'aplica una normalització. Les normalitzacions s'apliquen per estabilitzar la xarxa cosa que fa que es redueixi substancialment el temps d'entrenament.

DESCODIFICADOR

El descodificador té com a entrades un token d'inici, la llista de les entrades i sortides anteriors i també la sortida del codificador que contenia la informació d'atenció de l'entrada. El descodificador acaba de descodificar quan ha generat un token de sortida.

Masked Multi-head Attention

És un mecanisme d'atenció com el del codificador amb la diferència que al generar una paraula de la sortida només podem tenir accés a les paraules de sortida anteriors, per tant no podem calcular les puntuacions amb paraules del futur, i per això es fa servir un mètode anomenat *masking*. Aquesta màscara és una matriu de la mateixa mida que la matriu de puntuacions, formada per zeros a la diagonal i a sota, i per menys infinit a sobre de la diagonal. Aquesta màscara es suma a la matriu de puntuacions i quan més endavant s'apliqui la funció *softmax* on hi hagi un menys infinit obtindrem un zero.

Segona *Multi-head Attention*

Ara aplicarem una segona atenció *multi-head*, en aquest cas tenim com a *queries* i *keys* la sortida del codificador, i com a *values* la sortida de la *multi-head attention* anterior. Aquest proces relaciona la sortida del codificador i l'entrada del descodificador perquè es pugui decidir en quina part de l'entrada del codificador es important posar l'atenció.

Figure 10: Màscara Atenció *Multi-head* [11]

La sortida d'aquesta *multihead attention* es passa per una xarxa neuronal d'avançada cap endavant (*point-wise feed forward layer*).

Classificador Lineal i Softmax

La capa lineal actua de classificador, tan gran com classes hi hagi. Per exemple si hi ha 10.000 paraules (cada paraula es una classe), doncs la mida del classificador serà 10.000. A la sortida del classificador s'aplica una *softmax* per produir la probabilitat de cada paraula entre 0 i 1. Ens quedem amb l'index de la probabilitat més alta i això és la paraula predita.

El descodificador agafa la sortida i l'afegeix a a la llista d'entrades al descodificador, i continua descodificant així fins que prediu un token. La predicció que tingui una probabilitat més alta és la classe final assignada a l'últim token.

El descodificador també es pot apilar en N capes, cada capa agafa les sortides del codificador i de totes les capes de descodificador que té a sota. D'aquesta manera el model pot aprendre a extreure i focalitzar-se en diferents combinacions d'atenció per cada cap d'atenció.

2.5 Models basats en Transformers

La capacitat dels transformers a ser paral·lelitzats ha fet possible entrenar amb bases de dades molt grans i per tant desenvolupar models entrenats amb dades de text llargs com ara el corpus de la Viquipèdia. El pre-entrenament consisteix en entrenar un model des de zero inicialitzant els pesos aleatòriament. Tots els models de Transformers s'han entrenat de manera auto-supervisada, això vol dir que l'objectiu es calcula automàticament de les dades d'entrada del model, i no cal que es posin etiquetes a les dades manualment.

El pre-entrenamnet permet als models aprendre de forma automàtica a reconèixer patrons i estructures en el text. Un cop el model esta pre-entrenat es pot utilitzar per diverses tasques específiques de PLN ajustant els model amb noves dades d'entrenament del problema concret, aquest proces s'anomena *fine-tunning*. Aquest models són molt útils ja que tenen un coneixement general del llenguatge contingut en les dades d'entrenament i fan possible no haver d'entrenar models des de zero per cada problema específic, el qual seria molt costos.

Els diferents models de Transformers poden utilitzar les dues parts del Transformer de manera independent, depenent de la tasca:

- **Models amb només codificador:** van bé per tasques que requereixen una comprensió de l'entrada, com ara classificació de textos o reconeixement d'entitats.
- **Models amb només descodificador:** van bé per tasques generatives com ara generació de text.
- **Models codificador-descodificador o models seqüència-a-seqüència:** van bé per tasques generatives que requereixen una entrada, com ara traducció de textos o resum de textos.

Existeixen diversos models pre-entrenats que estan disponibles públicament i que han mostrat un rendiment molt bo en tasques de PLN. A continuació explicarem algunes de les arquitectures de models basats en Transformers i utilitzarem aquests models pre-entrenats en la part pràctica d'aquest treball.

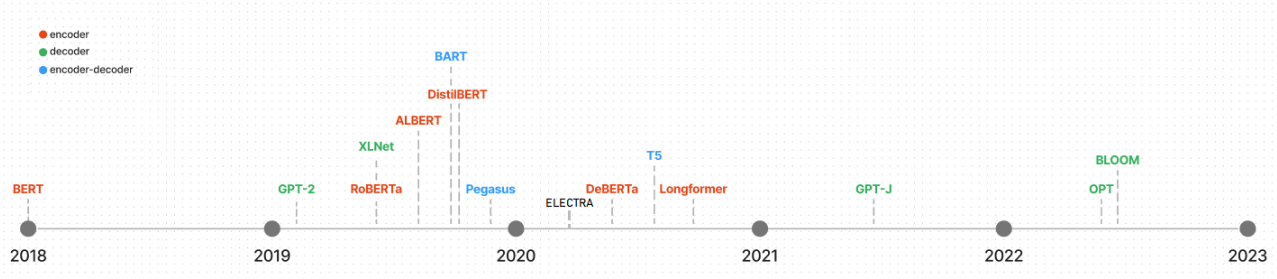


Figure 11: Història dels models de Transformers [19]

2.5.1 BERT

El *Bidireccional Encoder Representation from Transformer* és un model que va ser desenvolupat l'any 2018. Anteriorment les tasques de NLP es resolien amb models creats per cada tasca específica, però ara és possible crear models com BERT que és capaç de resoldre més de 11 tasques de NLP i amb millors resultats que els models previs. Ha sigut entrenat amb la Viquipèdia en anglès (unes 2.5B paraules) i el Google's BooksCorpus (unes 800M paraules). L'entrenament amb aquestes bases de dades tan grans ha sigut possible gràcies a l'arquitectura del transformer i l'augment de la velocitat aconseguit utilitzant TPUs (*Tensor Processing Units*).

Tal i com el seu nom indica BERT utilitza només la part del codificador de l'arquitectura del Transformer, i és bidireccional perquè té en compte les paraules de l'esquerra i de la dreta ja que els Transformers utilitzen tot el context d'una paraula per representar-la. D'aquesta manera una paraula polisèmica tindrà diferents representacions depenent de la frase en la que aparegui, per exemple, la paraula estrella en les següents frases: '*És una estrella del rock.*' i '*El Sol és una estrella.*' tindrà representacions diferents. En models anteriors com word2vec o GloVe s'hauria generat una sola representació per a la paraula estrella.

El model BERT es va entrenar en dues tasques per aprendre i reproduir representacions contextuals de les paraules:

- *Masked Language Modeling* (MLM): consisteix en reemplaçar algunes paraules per un token [MASK]. El model intenta predir el valor original de les paraules emmascarades basant-se en el context a partir de les altres paraules. Durant l'entrenament de BERT s'emmascaren el 15% de les paraules. Per exemple: '*I am [MASK] an ice cream.*' El model hauria de predir la paraula *eating*.
- *Next Sentence Prediction* (NSP), el model rep parelles de seqüències d'entrada i aprèn a predir si la segona seqüència va després de la primera en el text original o no. Durant l'entrenament de BERT el 50% de les entrades són dues seqüències que van seguides i la resta són parelles de seqüències aleatòries. Per exemple:
  - **Correcte:** '*Paul went shopping. He bought a new shirt.*'
  - **Incorrecte:** '*Ramona made coffee. Vanilla ice cream cones for sale.*'

Hi ha dos models originals de BERT, BERT *base* i BERT *large*. El primer està format per 12 blocs de codificadors i té una dimensió d'*embedding* de 768 i el segon està format per 24 codificadors i té dimensió d'*embedding* 1024. El model *base* és va entrenar amb 16 TPUs durant 4 dies i el *large* amb 64 durant 4 dies també.

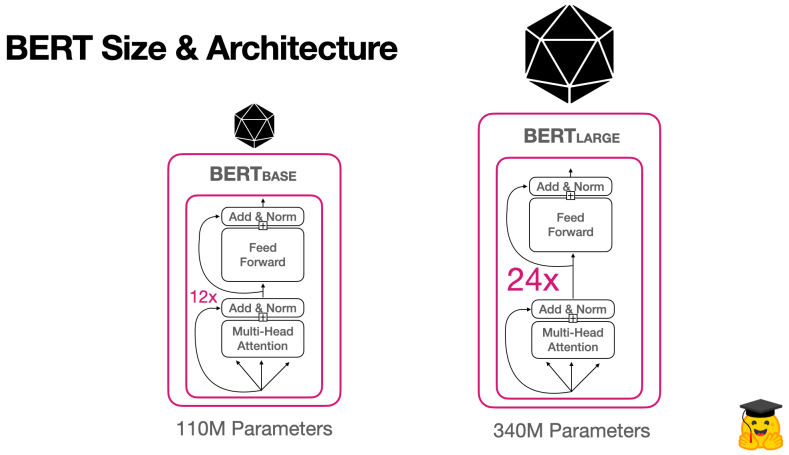


Figure 12: Arquitectura BERT [14]

Al fer *fine-tunning* per resoldre altres tasques, el model afegeix capes lineals a sobre dels codificadors apilats, aquestes capes extres serveixen per generar una sortida específica per la tasca que s'està resolent. D'aquesta manera es pot configurar BERT per resoldre gaire bé qualsevol tasca.

Com que aquests models han aconseguit resultats molt bons va créixer la demanda de models més petits per poder fer servir BERT en entorns computacionals més petits. Per això es van desenvolupar diferents models més petits, alguns dels quals explicarem a continuació.

2.5.2 RoBERTa

Robustly Optimized BERT Approach és una variant de BERT que va ser entrenada amb un conjunt de dades 10 vegades més gran que el de BERT. A més a més RoBERTa utilitza una tècnica d'emmascarament dinàmica per entrenar el model, que ajuda a aprendre representacions de paraules més robustes i generalitzades. Les modificacions que s'han fet respecte BERT són:

- **Treure l'objectiu de NSP:** S'ha comprovat que al treure la loss de NSP els resultats són iguals o una mica millors.
- **Entrenar amb una mida *batch* més gran i amb seqüències més llarges:** s'ha passa de una mida de *batch* de 256 a 8000 i la mida del vocabulari de 30k a 50k. Els avantatges que s'obtenen són que *batches* més grans milloren la perplexitat en l'objectiu MLM així com l'exactitud en les tasques finals. També fa que sigui més fàcil de paral·lelitzar.
- **Canviar dinàmicament el patró de màscara:** en l'arquitectura de BERT l'emmascarament es fa només una vegada durant el pre-processat. Per no fer servir una única màscara estàtica les dades d'entrenament es dupliquen i emmascaren 10 vegades, fent servir una màscara diferent cada cop que la seqüencia es passa al model.

2.5.3 DistilBERT

El model DistilBERT és un model més petit, més ràpid i més econòmic que BERT, es tracta d'una versió destil·lada del model BERT. Conté un 40% menys de paràmetres que BERT, s'executa un 60% més ràpid i preserva el 95% del seu rendiment.
Aquest model s'ha creat a partir de la tècnica *Knowledge distillation* que serveix per comprimir un model gran anomenat 'professor' en un model més petit, anomenat 'estudiant', s'entrena aquest model petit per reproduir el comportament del model 'professor', és a dir, per retornar les mateixes probabilitats que el model BERT *base*. Aquest model utilitza només 6 blocs de codificador (BERT *base* n'utilitzava 12), aquests blocs s'inicialitzen agafant 1 de cada 2 blocs pre-entrenats de BERT. I igual que RoBERTa només utilitza l'objectiu de MLM.
El model ha estat entrenat utilitzant una *loss* triple:

- La mateixa *loss* del model de llenguatge utilitzada per BERT.
- La *distillation loss* que mesura la similitud entre les sortides de DistilBERT i BERT.
- La *cosine-distance loss* que mesura com de similars són els estats amagats de DistilBERT i BERT.

La combinació d'aquestes funcions de *loss* és la que fa que hi hagi aquest aprenentatge estudiant-professor entre els dos models.

2.5.4 ALBERT

Albert busca els mateixos objecius que DistilBERT, i redueix la mida de BERT. Té millor rendiment amb la mateixa memòria però realment no acaba sent més ràpid. Albert està entrenat des de zero i es capaç d'obtenir bons resultats tot i tenir una arquitectura més petita ja que segueix les següents tècniques de reducció de paràmetres:

- ***Factorized Embedding Parameterization:*** dividir la matriu d'*embbeding* en dos trossos, així es pot augmentar la mida de la capa amagada sense modificar la dimensió de l'*embedding*.
- ***Cross-layer parameter sharing:*** Els blocs de codificadors comparteixen tots els paràmetres, d'aquesta manera es redueix la mida dels paràmetres i s'incrementa la regularització del model.
- **Treure les capes de *dropout*:** una capa de *dropout* ignora algunes neurones seleccionades a l'atzar, això vol dir que aquestes neurones no s'entrenen i són inútils durant un temps.

Albert s'entrena en la tasca MLM, amb una diferència que consisteix en emmascara n-grames, amb un màxim de 3 paraules, és a dir s'emmascararan grups de 1, 2 i 3 paraules, augmentant la dificultat de la tasca. També s'introdueix una nova tasca, en substitució de NSP, *Sentence Order Prediction* (SOP), s'intenta predir l'ordre correcte de dos fragments consecutius de text.

2.5.5 ELECTRA

Efficient Learning an Encoder that Classifies Token Replacements Accurately fa servir un nou enfocament de pre-entrenament que té com a objectiu igualar o superar el rendiment d'un model pre-entrenat en la tasca MLM utilitzant menys recursos computacionals. La tasca de pre-entrenament d'ELECTRA es basa en la detecció de tokens substituïts en la seqüència d'entrada. Aquesta configuració requereix dos models Transformers, un generador i un discriminador. Donada una seqüència d'entrada es reemplacen de manera aleatòria algunes paraules per [MASK]. El generador prediu els tokens originals que han sigut emmascarats. El discriminador té com entrada les seqüències amb els tokens [MASK] substituïts per les prediccions del generador. El discriminador prediu per cada token de la seqüència si es original o si ha sigut reemplaçat pel generador.

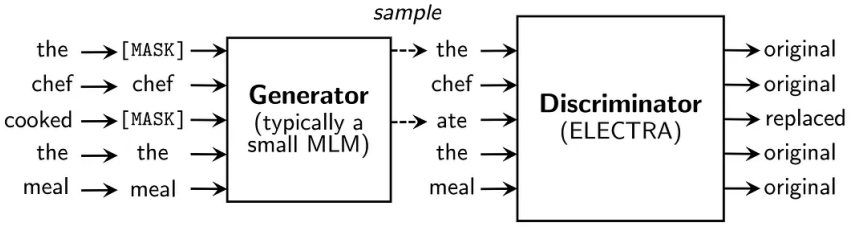


Figure 13: Pre-entrenamet ELECTRA [16]

La *loss* del discriminador es pot calcula sobre tots els tokens d'entrada, en MLM la *loss* només es calculava amb els tokens emmascarats que representaven un 15% del total. Aquesta diferència és la que fa que ELECTRA segui tan eficient ja que el model aprèn de tots els tokens d'entrada. A escala petita aconsegueix grans resultats fins i tot entrenada en una sola GPU.

2.6 Mètriques d'avaluació

- **Accuracy**: és la fracció de prediccions que el model ha fet correctament. $A = \frac{N.prediccionscorrectes}{N.totaldeprediccions}$
- **Loss**: la pèrdua és una mètrica que penalitza les males prediccions. Mesura l'error entre la sortida predita i la sortida desitjada. Si la predicció és perfecta valdrà 0 i per això els models intentaran minimitzar-la.
- **Matriu de confusió**: Les columnes representen el número de prediccions de cada classe i les files el nombre d'instàncies reals d'aquella classe. Permet veure les classes predites i les classes reals, de manera que és fàcil identificar, per exemple, si el model està confonent dues classes.
- **Cross-Validation**: és un mètode que utilitza diferents porcions de les dades per entrenar i testejar un model en diferents iteracions.

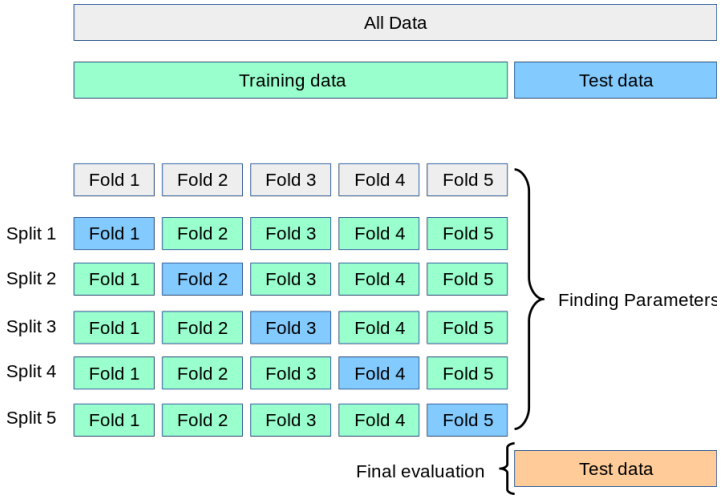


Figure 14: Cross-Validation [20]

3 Cas pràctic

El codi utilitzat en aquesta secció ha sigut desenvolupat en Python en llibretes de Google Colab, aquest codi es pot trobar en el següent repositori de GitHub: https://github.com/PumaresBenaiges/TFG.

3.1 Base de Dades

El conjunt de dades utilitzat és el TREC (*Text REtrieval Conference*), està format per un conjunt de preguntes en anglès que pertanyen a una de 6 categories principals i 50 subcategories. El conjunt està format per 5452 preguntes d'entrenament i 500 de test. Per tant es tracta d'un problema de classificació que intentarem resoldre fent servir els models esmentats a la part teòrica. La longitud mitjana de cada frase és de 50 caràcters i 10 paraules. El vocabulari està format per 8604 paraules diferents. Les dades han estat recollides de 4 fonts diferents: 4500 preguntes en anglès publicades per la USC (Hovy et al., 2001), unes 500 preguntes construïdes manualment per algunes classes rares, 894 de les versions de la base de dades TREC 8 i TREC 9, i 500 preguntes de TREC 10 que serveixen com a conjunt de test.
Les categories o classes a les quals pertanyen cadascuna de les preguntes són:

- ABBR (0): Abreviació.
- ENTY (1): Entitat.
- DESC (2): Descripció i conceptes abstractes.
- HUM (3): Ser humà.
- LOC (4): Localitat.
- NUM (5): Valor Numèric.

A continuació podem observar que les classes no estan balancejades, la classe *Abbrebiation* (0) té menys elements que les altres. Respecte les classes concretes tenim bastanta varietat en el nombre d'elements per classe.

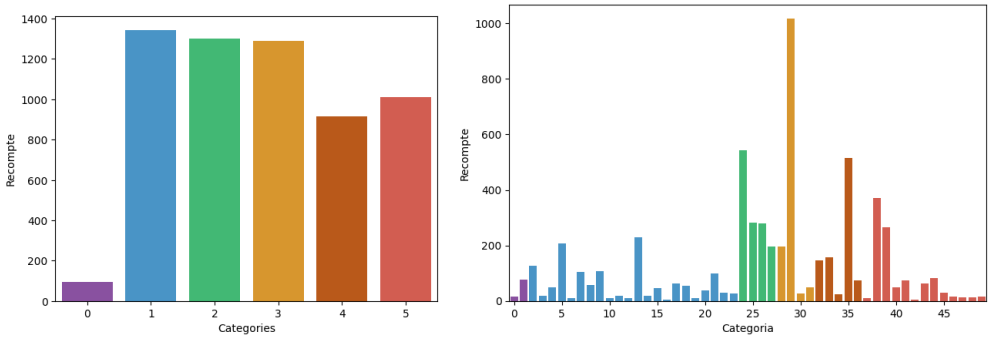


Figure 15: Distribució de les categories i subcategories

3.2 Models d'Aprenentatge Computacional

3.2.1 Metodologia

En aquest apartat hem fet servir la llibreria de *machine learning* de *sklearn* que té diversos models que podem fer servir. Per aplicar els models de Machine Learning hem seguit els passos següents:

1. **Netejar el text:** posar totes les lletres en minúscula i treure els signes de puntuació i altres elements que no siguin lletres. També treure els *stop words* per reduïr la mida del vocabulari.
   `What is Nitrox diving?` $\rightarrow$ `what is nitrox diving`
2. **Vectoritzar cada frase:** hem provat dues vectoritzacions diferents:
   (a) Obtenim una matriu on cada columna és una paraula del vocabulari. Per cada fila tenim el valor TF-IDF de la paraula si aquella frase conté la paraula, sinó tenim un 0. Obtenim una matriu 5452 x 1080. Vol dir que tenim 1080 paraules o característiques.
   (b) A partir de totes les paraules de les frases d'entrenament creem una llista que serà el vocabulari. A cada paraula del vocabulari li assignem un número. Codifiquem cada frase canviant la paraula pel número corresponent. Afegim un *padding* de zeros perquè tots els vectors siguin de la mateixa mida. Obtenim una matriu d'entrenament de 5452 x 18 ja que la frase més llarga conté 18 paraules.
3. Passem la matriu obtinguda a partir de les frases d'entrenament al model i l'entrenem.
4. Avaluem el model amb el conjunt de dades de test.

3.2.2 Resultats

	Vectorització 1		Vectorització 2	
Model	**Accuracy**	**Temps(s)**	**Accuracy**	**Temps(s)**
DecisonTree	0.31	0.97	0.72	0.04
RandomForestClassifier	0.31	0.97	0.69	0.33
GradientBoostingClassifier	0.63	61.3	0.68	5.27
KNeighborsClassifier	0.58	0.01	0.38	0.01
BaggingClassifier	0.68	0.60	0.60	0.04
Perceptron	0.53	0.80	0.19	0.04
SGDClassifier	0.72	1.33	0.19	0.95
LogisticRegression	0.71	4.86	0.23	1.49
LinearSVC	0.71	0.07	0.30	0.24

Podem observar que la majoria de models són relativament ràpids, i amb la segona vectorització, que utilitzava una matriu més reduida, l'entrenament és més ràpid. Els models *LinearSVC*, *Logistic Regression*

i *SGDClassifier* (que és un *K-Neighbors)* són els que obtenen millors resultats al utilitzar la primera vectorització, per contra *Decision Tree*, *Random Forest* i *Gradient Boosting Classifier* obtenen els millors resultats de la segona vectorització.

Hem realitzat una validació creuada (*Cross-Validation*) amb els models, aquest mètode produeix resultats lleugerament pitjors ja que dividim les dades d'entrenament en entrenament i validació, la qual cosa significa que disposem de menys dades per entrenar. Així doncs, obtenim els següents resultats:

	Vectorització 1		Vectorització 2	
Model	**Accuracy**	**Std. Dev.**	**Accuracy**	**Std. Dev.**
DecisonTree	0.31	0.01	0.65	0.01
RandomForestClassifier	0.40	0.02	0.66	0.02
GradientBoostingClassifier	0.55	0.01	0.64	0.02
KNeighborsClassifier	0.46	0.01	0.27	0.01
BaggingClassifier	0.54	0.02	0.22	0.02
Perceptron	0.48	0.05	0.19	0.04
SGDClassifier	0.63	0.02	0.23	0.02
LogisticRegression	0.63	0.01	0.20	0.01
LinearSVC	0.64	0.02	0.22	0.03

A continuació podem veure la matriu de confusió del model SGDClassifier, ens permet veure l'*accuracy* de cada classe per separat.

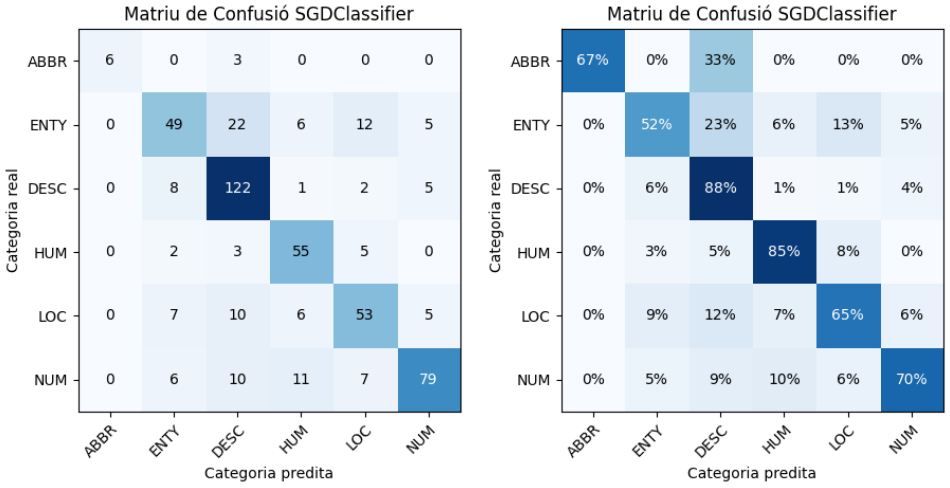


Figure 16: Matriu de confusió del model SGD

3.3 Xarxes Neuronals Recurrents

3.3.1 Metodologia

Per la part d'aprenentatge profund hem fet servir la llibreria *TensorFlow*, els models utilitzats consisteixen en una neurona seqüencial de *TensorFlow* (`tf.keras.Sequential`) amb diverses capes. Els passos que hem seguit són:

1. **Preprocessat de les categories:** escriure cada categoria com un vector de 6 posicions perquè el que retornarà el model serà la probabilitat que la frase d'entrada pertanyi a cada classe.
   `0` → `[1 0 0 0 0 0]`
   `1` → `[0 1 0 0 0 0]`
   `2` → `[0 0 1 0 0 0]`
   `3` → `[0 0 0 1 0 0]`
   `4` → `[0 0 0 0 1 0]`
   `5` → `[0 0 0 0 0 1]`

2. **La xarxa:**

   (a) La primera capa és una capa de vectorització de text `tf.keras.layers.TextVectorization`, que es responsabilitza de diverses tasques de pre-processament del text. Aquesta capa converteix el text en minúscules, elimina els signes de puntuació, segmenta el text en tokens que poden ser paraules o n-grams, indexa els tokens (assignant un número a cada token) i transforma cada seqüència d'entrada en un vector utilitzant els índexs. A continuació, mostrem un exemple de com es veuria una frase després de la vectorització:
   `What is a fear of money?` → `[3 4 7 61 5 189 0 0 0 0 0 0 0 0 0 0 0 0]`
   `From what cause does tuberculosis stem?` → `[30 3 3236 16 360 4190 0 0 0 0 0 0 0 0 0 0 0 0]`

   (b) Després tenim una capa d'*embedding*, que guarda un vector per a cada paraula. Aquesta capa converteix la seqüència d'índexos per a cada paraula en un vector específic per a cada paraula. Després d'entrenar amb moltes dades, les paraules amb significats semblants tindran vectors similars.

   (c) A continuació, tenim una capa `Bidirectional LSTM`, que propaga l'entrada en dues direccions: cap endavant i cap enrere. Aquesta capa permet que la informació del context passat i futur sigui considerada durant el processament de la seqüència.

   (d) Finalment, tenim dues capes `Dense` que realitzen un processament final i converteixen la representació vectorial en un logit únic, que és la sortida de classificació. La funció d'activació *relu* s'aplica a la primera capa `Dense` de 64 neurones, mentre que l'última capa `Dense` té 6 neurones i utilitza una funció d'activació *softmax* per generar les probabilitats de pertànyer a cada una de les 6 classes.

3.3.2 Resultats

Hem provat diverses variacions d'aquest model (canviar LSTM per GRU, afegir més capes bidireccionals, afegir *dropout*) i els resultats no han variat gaire d'un model a l'altre. Amb aquest model hem obtingut una *accuracy* del 82%. El model ha tardat aproximadament 1 minut en entrenar. Hem entrenat el model amb l'opció `EarlyStopping` que para d'entrenar el model quan no hi ha millora al cap d'un cert nombre d'èpoques, en aquest cas hem posat 5. En total hem posat un màxim de 30 èpoques però el model ha parat a la dinovena època, això vol dir que el nombre òptim d'èpoques és 14 (19-5). Si observem el gràfic de la *accuracy* i la *loss* al llarg de les èpoques podem veure que la època 14 té la màxima accuracy de test i el mínim valor de la *loss*.

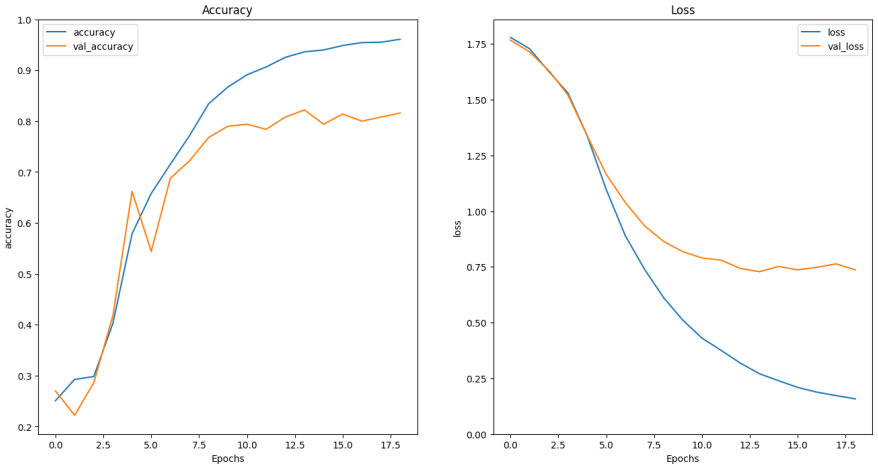
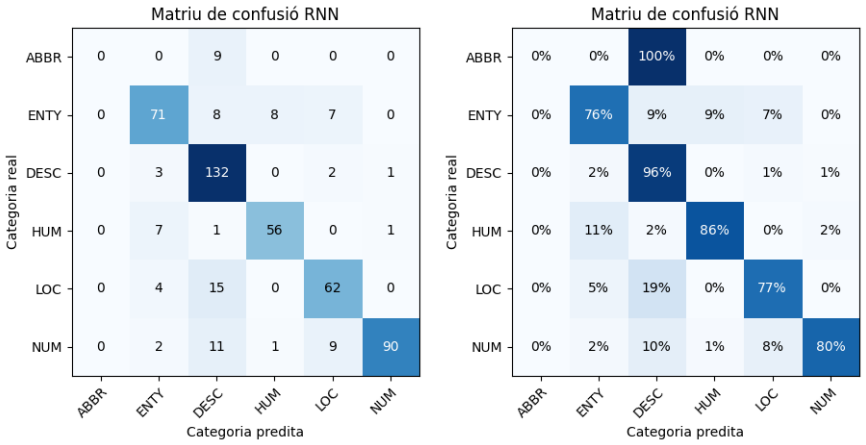


Figure 17: Evolució accuracy i loss amb les éspoques

A la matriu de confusió d'aquest model, veiem que el model no ha sigut capaç d'aprendre a distingir la classe 0, ja que no ha classificat cap mostra correctament en aquesta categoria. Segurament es deu al fet de que per aquesta classe hi ha molt poques mostres en comparació amb les altres classes. Respecte els models d'aprenentatge computacional hem millorat l'*accuracy* de totes les classes menys la 0.



3.4 Models basats en Transformers

3.4.1 Metodologia

Per poder entrenar la nostre base de dades amb models basats en transformers utilitzarem les següents llibreries:

- `Datasets`: aquesta llibreria conté mols conjunts de dades entre els quals hi ha el TREC.
- `Transformers`: en aquesta llibreria trobarem els models pre-entrenats i funcions per fer el *fine-tunning*.
- `Evaluate`: ens permetrà obtenir les mètriques per crear la funció d'avaluació.
- `nvidia-ml-py3`: servirà per monitoritzar la utilització de la memòria.
- `accelerate`: llibreria necesària per poder executar el codi que entrena el model.

Els models que hem utilitzat són:

- `bert-base-cased`: *cased* significa que distingeix entre majúscules i minúscules.
- `roberta-base`
- `xlm-roberta-base`: és una versió multilingüe de RoBERTa.
- `distilbert-base-uncased`: és *uncased* i per tant no distingirà entre majúscules i minúscules.
- `albert-base-v2`: versió 2 del model, s'ha entrenat amb més dades i durant més temps que el model original ALBERT.
- `google/electra-small-discriminator`: utilitza una versió petita del discriminador.

Els pasos que hem hagut de seguir per poder entrenar el model són els següents:

1. Pre-processat: aplicar un *tokenizer* per processar el text i obtenir una representació numèrica de les paraules.
2. Fem servir la classe `TrainningArguments` de la llibreria `Transformers` per definir els paràmetres d'entrenament.
3. Per l'avaluació fem servir una funció d'*accuracy* de la llibreria `Evaluate`.
4. Creem un objecte de la classe `Trainer`, que és una API optimitzada per entrenar models de Transformers de manera senzilla sense haver d'escriure manualment el bucle d'entrenament. Li passem els paràmetres d'entrenament que hem definit, les bases de dades d'entrenament i de test i la funció d'avaluació.

3.4.2 Resultats

Hem obtingut molt bons resultat i s'han millorat considerablement els resultats obtinguts amb mètodes anteriors. A continuació es pot observar una taula amb els resultats dels diferents models. Per cada model tenim l'*accuracy*, el temps que ha tardat, les mostres que processa per segon, la memòria utilitzada de la GPU en MB i el nombre de paràmetres del model. En tots els casos s'ha utilitzat una mida del *batch* de 8 i s'ha entrenat durant 3 èpoques, segurament es podria augmentar el nombre d'èpoques i millorarien els resultats però com que ja són molt bons només entrenarem durant 3 èpoques pel temps que tarda. Tots els models donen molt bons resultats, però és més lent que el que estàvem fent abans amb xarxes recurrents. Cal destacar el model ELECTRA, és el més ràpid ja que es el model més petit, i alhora no perd gaire precisió respecte els altres models. DistilBERT és el segon més ràpid i el que ha donat millor *accuracy*.

Model	**Accuracy** %	**Temps** min	**Mostres/ segons**	**M GPU** MB	**# Params**
BERT Base	97.4	24:46	11.00	8,463	108,314,886
RoBERTa	97.0	26:13	10.39	8,565	124,650,246
XLM-RoBERTa	96.6	28:36	09.53	11,309	278,048,262
DistilBERT	97.6	12:37	21.60	5,831	66,958,086
ALBERT	95.8	26:32	10.27	6,953	11,688,198
ELECTRA	95.0	05:23	52.07	3,445	13,550,342

A continuació tenim les *accuracy* i la *loss* del model ELECTRA al llarg de les èpoques. A partir de la època 5 l'*accuracy* comença a baixar i la *loss* te un comportament irregular. Per tant hauríem de fer servir menys de 5 èpoques per entrenar aquest model.

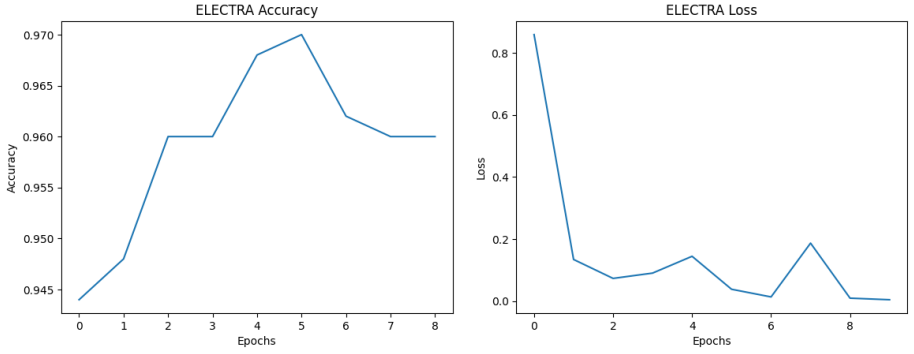
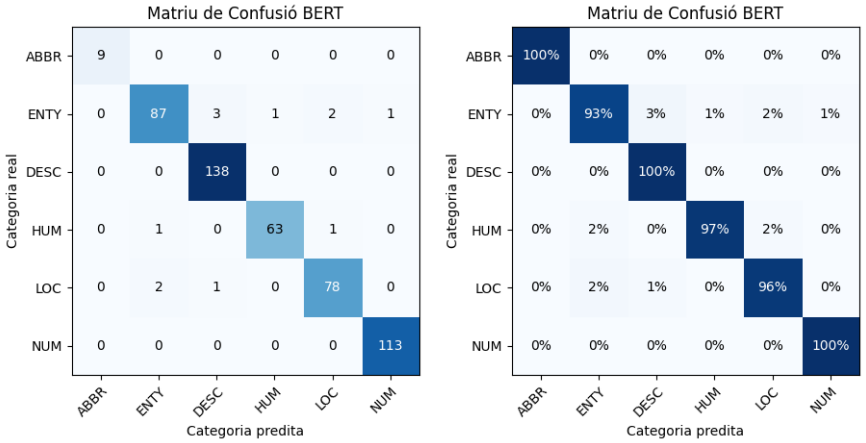


Figure 18: Electra Test *Accuracy* i *Loss*

Aquí tindríem la matriu de confusió dobtinguda amb el model BERT, observem que aquest model ha sigut capaç de predir bé la classe 0, que anteriorment no s'havia predit bé, amb una *accuracy* del 100%:



3.4.3 Millores eficiència entrenament

Per fer que l'entrenament sigui més ràpid provarem d'augmentar la mida del *batch*. Hem de tenir en compte que al fer-ho també s'utilitzarà més memòria, per tant farem les proves amb el model que utilitzava menys memòria, ELECTRA. Podem veure els resultats a la següent taula, el temps s'ha reduït una mica però a costa d'augmentar bastant la memòria. A partir de mida 128 ja es superava la memòria disponible. A més a més també ha disminuït la *accuracy* així que deixarem la mida *batch* que teníem a l'inici de 8.

Mida Batch	**Accuracy** %	**Temps** min	**Mostres/ segons**	**Memòria** MB
8	94.6	4:50	56.47	3,269
16	92.8	4:34	59.75	5,065
32	93.6	4:17	63.62	8,419
64	87.4	4:13	64.64	15,291

Provarem alguns altres mètodes per poder millorar l'eficiència de l'entrenament, els dos primers són per disminuir la memòria utilitzada i el tercer per augmentar la velocitat:

- ***Gradient Accumulation (GA):*** Consisteix en calcular el gradient fent passos petits enlloc de calcular els gradients per tot el *batch* de cop. Es calculen els gradients de manera iterativa en *batches* més petits fent passos cap endavant i cap enrere a través del model i s'acumulen els gradients durant el procés. Quan s'han acumulat suficients gradients s'executa el pas d'optimització del model. Això permet augmentar la mida del *batch* però també pot allargar una mica el temps d'execució. Per poder aplicar l'acumulació del gradient només cal afegir el paràmetre `gradient_accumulation_steps` a `TrainingArguments`.

- ***Gradient checkpoiting (GC):*** Quan es calculen els gradients durant el pas enrere es guarden totes les activacions del pas cap endavant, i això pot ocupar bastanta memòria. De manera alternativa es podrien oblidar aquestes activacions i re-calcular-les quan calgui. Tot i que això

alentiria l'entrenament. Aquest mètode fa servir una aproximació intermitja: guarda algunes activacions seleccionades, així només s'hauran de re-calcular unes quantes. Cal passar el paràmetre `gradient_checkpointing` com a `True` als arguments d'entrenament.

- **Tipus de dades decimals**: no cal guardar totes les variables amb una precisió de 32 bits. Si reduïm la precisió els càlculs seran més ràpids. Les activacions es poden guardar amb la meitat de precisió, en 16 bits. Els gradients per altra banda s'han de calcular amb precisió 32 bits en el pas d'optimització. Fer els diferents càlculs a diferents precisions s'anomena *mixed precision trainning* (MP), és a dir, entrenament amb precisió mixta. Cal canviar el paràmetre `fp16` de `TrainingArguments` a `True`.

En la següent taula podem veure les millores aplicades per separat i en conjunt al model ELECTRA. Observem que quan millorem la memòria també augmentem el temps d'entrenament. Quan reduïm la velocitat (MP) en canvi també millorem l'ús de memòria (recordem que estem guardant les dades en un format més petit) tot i que no tant com abans. La combinació de totes les millores dona un bon resultat, obtenint un *speed up* de 1.25 o un augment en la velocitat del 20% i una reducció de la memòria del 43%.

	Accuracy %	**Temps** min	**Mostres/ segons**	**Memòria** MB
Inicial	95.0	5:02	54.12	3269
GA	96.4	5:35	48.80	3247
GA + GC	96.2	6:40	40.86	1841
MP	95.6	3:52	70.42	2931
GA + GC + MP	94.4%	4:00	68.73	1861

A continuació podem veure les millores aplicades a tots els models provats i el guany obtingut. Hem aplicat totes les millores a tots els models excepte el model DistilBERT i el model Albert als quals no s'ha pogut aplicar el *Gradient Checkpointing*. I per això veiem que han millorat molt en velocitat i poc en memòria.

Model	**Accuracy** %	**Temps** min	**Mostres/ segons**	**M GPU** MB	**Millora Temps** %	**Millora Memòria** %
BERT	97.6	13:08	20.75	3981	47	53
RoBERTa	97.0	13:05	20.80	4303	50	50
XLM-RoBERTa	95.6	14:13	19.18	7339	50	35
DistilBERT	97.2	04:55	55.4	4759	61	18
Albert	96.4	12:09	22.43	6753	54	3
ELECTRA	95.0	04:00	68.73	1861	24	46

4 Conclusions

En aquest treball s'ha comprovat que els models basats en transformers són els que donen millors resultats en tasques de classificació de textos. Aquests models superen significativament els enfocaments tradicionals d'aprenentatge automàtic i aprenentatge profund en termes de precisió i exactitud.
També s'ha comprovat que l'ús de models basats en transformers requereixen un temps d'execució més gran i un major ús de memòria en comparació amb els altres mètodes. Tot i així aquest cost es veu compensat pels resultats obtinguts que són molt més precisos.

Per altra banda s'ha vist que es poden utilitzar models de transformers més petits i obtenir igualment bons resultats. BERT és considera l'estat de l'art en moltes aplicacions però models com DistilBERT o ELECTRA poden ser una bona alternativa quan calgui una inferència ràpida. Per tant depenent del problema escollirem un model o altre basat-nos el temps que tinguem, la precisió que vulguem obtenir i la memòria de la qual disposem.

A més a més hem vist els avantatges de poder utilitzar models pre-entrenats sense els quals no seria possible realitzar aquest tipus d'entrenamnet, ja que entrenar un model des de zero requereix una quantitat molt gran de dades, i això fa que sigui molt costos a nivell de temps i recursos computacionals, fins i tot té un impacte mediambiental important.
Finalment aquest treball ha permès comprendre millor els models basats en transformers i la seva aplicabilitat en la classificació de textos.

Bibliografia

[1] *Attention is all you need.* Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, 2017. Link

[2] *TREC dataset.* Link

[3] *Text Classification: What it is And Why it Matters.* Monkey Learn Link

[4] *From a Single Decision Tree to a Random Forest.* Rosaria Silipo, 2019. Link

[5] *SVM: Feature Selection and Kernels.* Pier Paolo Ippolito, 2019. Link

[6] *Activation Function.* AI Wiki, 2020. Link

[7] *Basic Architectur of RNN and LSTM.* Vivek Singh Bawa, 2017. Link

[8] *A Brief Introduction to Recurrent Neural Networks.* Jonte Dancker, 2022. Link

[9] *LSTM for Text Classification in Python.* Shraddha Shekhar, 2021. Link

[10] *RNN, LSTM and GRU.* dProgrammer lopez, 2019. Link

[11] *Illustrated Guide to Transformers - Step by Step Explanation.* Michael Phi, 2020. Link

[12] *Transformers.* Ria Kulshrestha, 2020. Link

[13] *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, 2018. Link

[14] *BERT 101: State Of The Art NLP Model Explained.* Britney Muller, 2022. Link

[15] *Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT.* Victor Sanh, 2019 Link

[16] *ELECTRA: Pre-trainning text encoders as disriminators rather than generators.* Kevin Clark, Minh-Thang Luong, Quoc V. Le, Christopher D. Manning, 2020. Link

[17] *Overview of ROBERTa model.* pawangfg, 2023. Link

[18] *A review of pre-trained language models: from BERT, RoBERTa, to ELECTRA, DeBERTa, BigBird, and more.* Tung.M.Phung, 2021. Link

[19] *The Transformer model family.* Hugging Face, 2023. Link

[20] *Cross-validation: evaluating estimator performance.* Scikit-learn developers. Link