Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC6205 Procesamiento de Lenguaje Natural

Reconocimiento de Entidades Nombradas

Competencia #2

Estudiantes:
Mario Vicuña
Miguel Videla

Equipo:
TeamChalla

Profesor:
Felipe Bravo

Auxiliares:
Pablo Badilla
Gabriel Chaperón
Cristián Tamblay

Fecha:
2 de Agosto de 2020

Índice

1. Introducción

El reconocimiento de entidades nombradas o *named-entity recognition* (NER) es un problema fundamental de análisis de texto no estructurado, el cual permite extraer información relevante de un texto mediante la detección y clasificación de entidades de una secuencia discreta correspondiente a las palabras que lo componen, tales como nombres de personas, organizaciones, lugares, valores, etc.

El presente problema consiste en la detección y clasificación de palabras correspondientes a las entidades correspondientes a organizaciones (`ORG`), lugares (`LOC`), personas (`PER`), misceláneos (`MIS`) y palabras no correspondientes a entidades (`O`) de un conjunto de oraciones en idioma español, además de detectar si las palabras corresponden a una entidad inicial (`B`), en el caso de corresponder a la palabra inicial de una entidad compuesta por múltiples palabras, o intermedia (`I`) en caso contrario. Por lo tanto, el problema propuesto equivale a un problema de clasificación de palabras en 9 clases, el cual es realizado en el contexto de una competencia de reconocimiento de entidades nombradas, donde se provee de un conjunto de entrenamiento y validación para el diseño y evaluación de distintas soluciones con el objetivo de obtener el mejor desempeño de clasificación de acuerdo a las métricas *F1-score*, *precision* y *recall* sobre un conjunto de prueba no etiquetado.

Se exploraron soluciones basadas en *deep learning*,consistentes principalmente en distintas configuraciones de redes neuronales recurrentes, realizando un análisis exhaustivo de desempeño frente la variación de distintos hiperparámetros y arquitecturas del modelo mencionado, además de probar la incorporación de *embeddings* pre-entrenados basados en la técnica *fastText*. Adicionalmente, se decidieron explorar métodos del estado del arte basados en los modelos *transformers*, centrado en el *fine-tuning* del modelo BERT debido a que, de acuerdo a nuestro conocimiento, es el único modelo del estado del arte disponible pre-entrenado sobre un amplio corpus en idioma español [1]. El mejor desempeño fue obtenido mediante la utilización del modelo BERT, obteniendo el primer lugar de la competencia en las métricas *F1-score* y *precision*, con un valor de 0.76 y 0.81, respectivamente, y el segundo lugar en la métrica *recall*, alcanzando un valor de 0.72.

Se evidencia la eficacia de los algoritmos de *deep learning* en la tarea de reconocimiento de entidades nombradas, atribuida principalmente a la capacidad de representación de cualquier función parametrizada por los pesos y sesgos del modelo en virtud del *teorema de aproximación universal de redes neuronales*, además de la efectiva capacidad de modelos especializados en el procesamiento de series de tiempo, tales como las redes neuronales recurrentes y los *transformers*, de representar información contextual y modelar dependencias temporales extensas en una secuencia temporal discreta correspondientes a las palabras de una oración, permitiendo extraer y procesar características altamente informativas de los datos en función de la tarea de clasificación a resolver, evitando una selección manual completamente no trivial de estas.

El documento se estructura como sigue: En la sección 2 se describen las métricas de evaluación utilizadas para comparar el desempeño de los distintos modelos propuestos. En la sección 3 se presenta una breve descripción de distintos los algoritmos utilizados para la resolución de la tarea NER. En la sección 4 se presentan en detalle los distintos experimentos realizados, reportando y analizando los resultados obtenidos y en la sección 5 se presenta un resumen del trabajo realizado, una discusión general de los resultados analizados en la sección experimental y las principales conclusiones obtenidas junto a una propuesta de trabajo futuro.

2. Métricas de Evaluación

En la presente sección se describen brevemente las métricas tres de evaluación consideradas para la comparación de desempeño de los distintos modelos utilizados para la resolución de la tarea *named entity recognition* en lenguaje español.

- **Precision**: Corresponde a la fracción de clasificaciones correctas entre los datos clasificados como positivos. Matemáticamente Precision corresponde a:

$$P = \frac{TP}{TP + FP} \tag{1}$$

  siendo TP la cantidad de clasificaciones positivas correctas y FP incorrectas. Conceptualmente, optimizar sobre la Precision corresponde a minimizar el Error de Tipo I, es decir que se busca una baja tasa de falsas detecciones.

- **Recall**: Corresponde a la fracción de clasificaciones correctas entre los datos que verdaderamente corresponden a la clase positiva. Matemáticamente Recall corresponde a:

$$R = \frac{TP}{TP + FN} \tag{2}$$

  siendo FN los falsos negativos o datos de clase positiva incorrectamente clasificados. Conceptualmente, optimizar sobre el Recall corresponde a minimizar el Error de Tipo II, es decir que se busca una baja tasa de no-detecciones.

Típicamente existe un trade-off entre las métricas de Precision y Recall. Por ejemplo, para maximizar el Recall un algoritmo podría determinar que todas sus entradas son postivas (con lo que $FN = 0$). Evidentemente con esto se dispara la tasa de falsos positivos, perjudicando evidentemente la Precision del modelo y, más aún, es un comportamiento que claramente no se desea en un clasificador.

No obstante, se pueden combinar ambas métricas, por ejemplo mediante el F1-Score, la cual es la tercera medida de desempeño en el grupo de las que no se evaluarán en la competencia.

- **F1-Score**: El F1-Score surge como una métrica que combina ambas con la finalidad de aproximarse a un punto óptimo, por medio de una media harmónica ponderada mediante un parámetro de balance α. Esta métrica, en términos de P y R se define matemáticamente como:

$$F_\alpha = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} \tag{3}$$

  También se puede escribir esta expresión en términos del factor de balance β de la siguiente forma:

$$F_\beta = (1 + \beta^2)\frac{PR}{\beta^2 P + R} \tag{4}$$

Cuando $\beta = 1$, o equivalentemente $\alpha = \frac{1}{2}$, $F_\beta = F_1$ toma precisamente la forma de la media harmónica (balanceada) entre P y R, de ahí que esta métrica se conozca como F1-Score. En términos generales se la denomina simplemente F-Score, pudiendo considerar cualquier valor para α o β.

Cabe destacar que a lo largo del presente trabajo las métricas reportadas para los modelos no contemplan la clase `O`, debido a que la mayoría de etiquetas en el corpus de entrenamiento corresponden a dicha clase (87.6 %) y el cálculo de las métricas puede verse dominado por el buen desempeño para esta clase y esto podría llevar a interpretar mal los resultados y desempeño del modelo siendo poco representativa respecto a la tarea de reconocimiento de entidades a resolver.

3. Descripción de Algoritmos

En la presente sección se presenta una breve descripción de los distintos algoritmos de *Machine Learning* utilizados.

3.1. Perceptrón Multicapa

El perceptrón multicapa o *multilayer perceptron* (MLP) [2] es un tipo de red neural artificial que procesa un vector de entrada x a través de múltiples capas l, las cuales multiplican el vector de entrada por una matriz de pesos W_l correspondiente a cada capa, adicionando un respectivo término de sesgo b_l. La salida de cada capa es procesada por una función no lineal conocida como función de activación (ej: Sigmoide (σ), tanh, ReLU, etc.) cuyo resultado es sucesivamente utilizado como entrada de la capa siguiente hasta alcanzar la última capa, llamada capa de salida. De este modo, la salida $\hat{y}$ del perceptrón multicapa puede ser expresado como:

$$\hat{y} = \sigma(\cdots\sigma(\sigma(xW_0 + b_0)W_1 + b_1)\cdots)W_l + b_l \tag{5}$$

con W_i y b_i la matriz de pesos y el término de sesgo correspondiente a la i-ésima capa y σ la función de activación sigmoide.

El perceptrón multicapa utiliza una técnica de aprendizaje supervisado llamada *back-propagation* en su etapa de entrenamiento, es decir, dado un conjunto de datos de entrenamiento x con su respectiva salida deseada y, el modelo modifica sus parámetros mediante la propagación del error de predicción determinado por una función de pérdida L entre la salida $\hat{y}$ predicha por el modelo y la salida deseada y.

Dependiendo de la función de pérdida escogida, el perceptrón multicapa es utilizado típicamente para tareas de clasificación o regresión, y en virtud del *Teorema de Aproximación Universal*, las funciones de activación no lineales aplicadas entre capas y el gran número de parámetros del modelo permiten representar una amplia variedad de funciones que se ajusten a los datos con precisión arbitraria.

3.2. Embeddings

En términos simples, se puede entender como *embedding* a una representación vectorial densa de las palabras en un corpus, usualmente diseñados para una tarea específica. Para efectos del presente trabajo se consideran *embeddings* obtenidos [3] por medio de `FastText` para la tarea de *skipgram*, puesto que estos son los *embeddings* pre-entrenados que se utilizarán en los experimentos.

Skipgram consiste en la predicción del contexto en una ventana de tamaño k a partir de una palabra central w_i. De esta manera se entrenan dos capas de neuronas: una que codifica la palabra w_i (en formato *one-hot vector*) en su *embedding* d-dimensional c_i, y una segunda que decodifica dicho *embedding* para la predicción del contexto.

`Fastext` [4] consiste en una extensión del modelo *skipgram* en el que los *embeddings* de una palabra no son obtenidos directamente en una matriz de pesos, sino que se construyen a partir de los *embeddings* de sus n-gramas. Es decir, cada palabra w_i es descompuesta en sus n-gramas y estos son alimentados al modelo *skipgram* para realizar la predicción.

3.3. Red Neuronal Recurrente

La red neuronal recurrentes o *recurrent neural network* (RNN) es un tipo de red neuronal cuya arquitectura está especialmente diseñada para procesar datos de estructura de series de tiempo gracias a su capacidad de almacenar memoria a través de propagación de estados en el tiempo.

Sean θ los parámetros de la red neuronal, x_t y h_t los datos de entrada y el estado en el tiempo t, respectivamente, la red neuronal recurrente puede expresarse mediante la siguiente función recurrente:

$$h_t = f(x_t, h_{t-1}, \theta) \tag{6}$$

Una de las arquitecturas más simples de redes neuronales recurrentes es la Elman RNN [5], la cual simplemente multiplica matrices de pesos a las entradas y a los estados adicionando un término de sesgo:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{7}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{8}$$

donde W_h, U_h y b_h corresponden a las matrices de pesos relativas a la entrada, el estado anterior y el término de sesgo para cada tiempo t, W_y y b_y corresponden a la matriz de pesos y el término de sesgo de la salida, y σ_h y σ_y corresponden a funciones de activación arbitrarias.

Una de las principales limitaciones de esta arquitectura es su incapacidad de procesar largas dependencias temporales entre los datos siendo, en este escenario, inviable su entrenamiento debido a fenómenos de desvanecimiento y explosión del gradiente del error. Para lidiar con este problema se han propuesto múltiples arquitecturas las cuales añaden compuertas que facilitan el flujo directo de información a través del tiempo. Una de las arquitecturas más famosas es la llamada *Long Short-Term Memory* (LSTM) [6], la cual se compone de múltiples compuertas con matrices de pesos W y sesgos b independientes, las cuales determinan la información debe ser utilizada para actualizar los estados de la red y la información que debe ser desechada en el proceso. Matemáticamente, la red neuronal recurrente LSTM queda determinada por las siguientes compuertas:

$$i_t = \sigma\big(W_i x_t + U_i h_{t-1} + b_i\big) \tag{9}$$

$$f_t = \sigma\big(W_f x_t + U_f h_{t-1} + b_f\big) \tag{10}$$

$$o_t = \sigma\big(W_o x_t + U_o h_{t-1} + b_o\big) \tag{11}$$

$$\tilde{C}_t = \tanh\big(W_g x_t + U_g h_{t-1} + b_c\big) \tag{12}$$

$$C_t = \sigma\big(f_t * C_{t-1} + i_t * \tilde{C}_t\big) \tag{13}$$

$$h_t = \tanh(C_t) * o_t \tag{14}$$

con h_{t-1}, C_{t-1}, los estados ocultos y de celda en el tiempo $t-1$, x_t la entrada en el tiempo t, i_t, f_t y o_t las compuertas de entrada, olvido y salida, respectivamente, y $\tilde{C}_t$, C_t y h_t, el estado de celda propuesto, el estado de celda final y la estado oculto final en el tiempo t.

Otra arquitectura común de red recurrente que emplea compuertas es la denominada *Gated Recurrent Unit* (GRU) [7], la cual se suele entender como una simplificación de la LSTM, debido a que incorpora una menor cantidad de parámetros y compuertas. A grandes rasgos se cuenta dos

compuertas: r que controla el acceso a memoria (el estado anterior h_{t-1}) y a partir de esta se genera un estado candidato $\tilde{h}_t$. Este estado candidato se interpola con el estado anterior por medio de la compuerta z que es la que finalmente determina la información que se traspasa al nuevo estado h_t proveniente tanto del estado anterior como del estado candidato. Matemáticamente esta arquitectura se define mediante las siguientes ecuaciones:

$$h_t = (1 - z) * h_{t-1} + z * \tilde{h}_t \tag{15}$$

$$z = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{16}$$

$$r = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{17}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r * h_{t-1}) + b_h) \tag{18}$$

Una modificación o componente adicional (y muy común) que se emplea en redes recurrentes es la bidireccionalidad. La idea que sustenta esto es que para una palabra x_i una red recurrente computa el estado h_i con la información relevante contenida en $x_1, \ldots, x_{i-1}$, pero para la tarea deseada el resto de la secuencia $x_{i+1}, \ldots, x_n$ podría también contener información importante (por ejemplo el rol semántico de $x_i =' can'$ en *the trash can ...* podría estar mejor definido con el contexto $x_{i+1}, \ldots, x_n$ que si se considera tan solo $x_1, \ldots, x_{i-1}$). Una red recurrente bidireccional se compone de dos redes recurrentes: la primera construye los estados $h_1^f, \ldots, h_n^f$ al alimentarse con las entradas en su orden secuencial original $x_1, \ldots, x_n$, mientras que la segunda red se alimenta con la secuencia invertida $x_n, \ldots, x_1$ para obtener los estados $h_n^b, \ldots, h_1^b$. Con esto, la predicción final para cada *input* x_i se hace tomando como entrada la concatenación de los estados obtenidos $[h_i^f; h_i^b]$.

Las redes neuronales recurrentes también se pueden apilar, de modo tal que la secuencia de salida de una se utiliza como secuencia de entrada para la siguiente. A esto se le denomina una *stacked RNN*.

3.4. Transformer

El modelo *transformer* [8] propone utilizar únicamente mecanismos de atención para modelar la dependencia temporal de una secuencia, prescindiendo completamente del modelo neuronal recurrente, permitiendo la paralelización del mismo, disminuyendo considerablemente los tiempos computacionales de procesamiento y mejorando, a la vez, su desempeño frente a distintas tareas.

El modelo propuesto sigue la estructura encoder-decoder, donde el encoder mapea una secuencia de entrada $(x_1, ..., x_n)$ a una representación continua $z = (z_1, ..., z_n)$, mientras que el decoder, dada la representación intermedia z, genera una secuencia de salida $(y_1, ..., y_m)$ de los símbolos de manera autoregresiva, es decir, utiliza todos los símbolos previamente generados para generar el siguiente, donde el encoder y decoder del modelo *transformer* se compone de un apilamiento de mecanismos de auto-atención, atención punto a punto, y capas *fully-connected*.

La función de atención corresponde a un mapeo de una *query* y con conjunto de valores pares *key-value* a una salida. Dado los vectores *query*, *key* y *value*, la salida corresponde a la suma ponderada de los elementos del vector *value*, cuyos pesos son computados mediante una función de compatibilidad entre el vector *query* y el vector *key*.

Sean Q, K, V matrices correspondientes a un conjunto de *queries*, *keys* y *values* de dimensión d_k, d_k y d_v, respectivamente, se define la función de atención producto punto escalda como:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_h}}\right)V \tag{19}$$

En vez de ejecutar una sola función de atención sobre *queries*, *keys* y *values* de dimensión d_{model}, se propone proyectar dichos vectores h veces mediante proyecciones lineales de dimensión d_k, d_k y d_v, respectivamente, donde cada proyección corresponde a aplicación de la función de atención en paralelo, retornando una salida de dimensión d_{model}, las cuales son concatenadas y proyectadas nuevamente hasta obtener los valores finales. Esta modificación de la función de atención se denomina *multi-head attention*, y queda definida como:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{20}$$

con $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$ y $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ las matrices de proyección.

De este modo, el modelo *transformer* aplica las función *multi-head attention* en capas de atención encoder-decoder, donde las *queries* corresponden a la salida de la capa previa del decoder, mientras que los valores *keys* y *values* corresponden a la salida del encoder, permitiendo que cada posición en el decoder sea capaz de atender todas las posiciones de la secuencia de entrada.

Adicionalmente, cada capa del enconder computa la función *multi-head attention* cuyas *queries*, *keys* y *values* vienen de un mismo lugar (auto-atención) correspondiente a la salida de la capa previa del encoder, permitiendo que a cada posición del encoder atender a todas las posiciones de la capa previa del mismo. De modo similar, el decoder aplica el mismo mecanismo de auto-atención restringiendo el flujo de información hacia la izquierda mediante el enmascaramiento de los valores de entrada correspondientes a conexiones que violen la propiedad auto-regresiva.

Finalmente, la salida de capa de atención en el encoder y decoder pasa por una red *fully-connected* de dos transformaciones lineales con función de activación *ReLU* aplicado de manera separada e idéntica a cada posición de la secuencia y los tokens de entrada del modelo son transformados a un *embedding* de dimensión d_{model} fija, donde la salida del modelo corresponden a una transformación lineal con función $softmax$ de la última capa del decoder.

En la figura 1 se presenta un diagrama de la arquitectura del modelo *transformer* anteriormente descrita, donde N_x corresponde a una capa del encoder o decoder.

3.4.1. BERT

El modelo BERT (*Bidirectional Encoder Representations from Transformers*) [9] corresponde a un modelo *transformer* de codificación bi-direccional de arquitectura multipropósito.

La restricción de unidireccionalidad del modelo *transformer* (propiedad auto-regresiva) se relaja mediante un entrenamiento de lenguaje enmascarado del modelo (*masked-language model*), en el cual se enmascara aleatoriamente algún token de la secuencia de entrada, con el objetivo de predecir la id del token oculto. Adicionalmente al *masked-language model*, el modelo BERT se

entrena bajo la tarea de predicción de oración posterior (*next sentence prediction*), el cual utiliza conjuntamente representaciones de texto pareadas pre-entrenadas, permitiendo de este modo, obtener representaciones bidirecionales del lenguaje altamente informativas.

El modelo pre-entrenado bajo estas tareas puede ser aplicado a cualquier otra añadiendo capas *fully-connected* sobre alguna representación de salida de interés, entrenadas mediante *fine-tuning*.

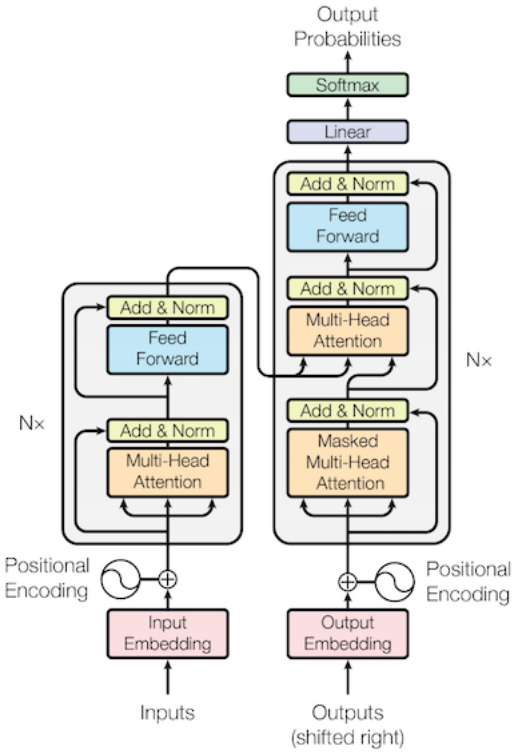

Figura 1: Esquema de arquitectura de modelo *transformer*. Extraído de [8].

4. Experimentos

En la presente sección se describen los experimentos realizados abocados a la comparación de desempeño en la tarea de reconocimiento de entidades nombradas de los modelos descritos en la sección 2, frente a variaciones en la configuración de sus hiperparámetros, reportando y analizando los resultados obtenidos.

4.1. Redes Neuronales Recurrentes

Por completitud y antes de introducir los diversos experimentos llevados a cabo con arquitecturas de redes recurrentes cabe considerar primero el *baseline* provisto como base y considerado como *benchmark* a superar. Este modelo consiste en una capa de *embedding* con la cual se obtienen representaciones de dimensión 100 para cada palabra del diccionario extraído del *corpus*. Los *embeddings* obtenidos para la secuencia son alimentados a una red LSTM unidireccional de dos capas cuyos estados son vectores de 128 dimensiones. La clasificación de cada palabra de una secuencia dada se obtiene alimentando una capa lineal con el estado correspondiente a dicha palabra. Adicionalmente, se emplea una capa de *dropout* entre la capa de *embedding* y la LSTM, dentro de la misma LSTM (aplicado sobre los estados ocultos) y a la entrada de la capa lineal utilizada para la clasificación.

El desempeño promedio de este modelo sobre el conjunto de validación, obtenido con 5 pruebas se presenta en la siguiente tabla:

Tabla 1: Desempeño en validación del modelo *baseline*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,221	0,58	0,64	0,58
2	0,226	0,57	0,63	0,58
3	0,222	0,56	0,62	0,57
4	0,219	0,55	0,62	0,54
5	0,226	0,53	0,61	0,53
Promedio	0,2228	0,558	0,624	0,56

Cabe destacar que en cada una de las pruebas el modelo se entrenó durante 10 épocas, de las cuales el con mejor desempeño en términos de *loss* es guardado y empleado para reportar los resultados con los que se construyó la tabla 1. Este mismo procedimiento para guardar el mejor modelo encontrado en cada entrenamiento es empleado en los experimentos posteriores.

4.1.1. Número de Épocas y Early Stopping

Un primer elemento a considerar para modificar es la variación del número de épocas y la implementación de *early stopping*. Por un lado, para no limitar la información que puede aprender la red a partir de los datos se propone aumentar el número de épocas (en principio hasta 100). Evidentemente esto repercute en tiempos de entrenamiento largos y, más importantemente, en

overfitting. Para evitar esto se incorpora además *early stopping*. Particularmente, el criterio de término considerado es el siguiente: una vez guardado un modelo por ser el que mejor *loss* reporta en validación hasta la n-ésima época, si durante las m épocas posteriores este desempeño no es superado, se detiene el entrenamiento. En caso contrario, se guarda nuevamente el modelo.

Cabe destacar que se considera $m = 10$, puesto que se observó que, una vez alcanzada una pérdida mínima, tras esa cantidad de épocas ya es posible apreciar un claro sobreajuste y por tanto se tiene un criterio suficiente para detener el entrenamiento.

Otro aspecto interesante a considerar es la elección de la métrica en base a la cual evaluar el criterio de *early stopping*. La principal justificación para optar por la pérdida por sobre las otras métricas es que al computar estas no se consideran las palabras cuya etiqueta es $'O'$ (es decir no corresponde a ninguna entidad nombrada). Evaluar las métricas de esta forma tiene sentido debido a que el 87.6 % de las palabras en el *corpus* corresponden a dicha categoría, por lo que una red que solo es capaz de predecir $'O'$ puede obtener valores muy cercanos a 1, en cambio al calcularlas de esta forma se evidencia más claramente el desempeño del modelo respecto a las clases de interés. Pero hacer la elección del modelo en base a este criterio significa entonces que el modelo puede incurrir en falsos positivos y que no son detectados puesto que no se está considerando el desempeño sobre la clase $'O'$. Por lo tanto se opta por usar la *loss* (que captura el desempeño sobre todas las clases) para guardar el mejor modelo obtenido durante el entrenamiento y las otras tres métricas se utilizarán para comparar diferentes arquitecturas evaluadas en experimentos diferentes.

Dicho lo anterior, al entrenar durante (máximo) 100 épocas, considerando el *early stopping* descrito se obtiene un modelo con el desempeño promedio dado por la tabla 2:

Tabla 2: Desempeño en validación del modelo *baseline* entrenado considerando *early stopping*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,223	0,57	0,63	0,58
2	0,223	0,57	0,63	0,58
3	0,231	0,57	0,64	0,58
4	0,217	0,58	0,63	0,59
5	0,21	0,58	0,64	0,59
Promedio	0,2208	0,574	0,634	0,584

Cabe destacar que ya con la mera inclusión de más épocas de entrenamiento con *early stopping* se obtiene un mejor desempeño promedio en cada una de las métricas y en cuanto a pérdida. Por lo tanto, en los experimentos que siguen también se consideran estos elementos en el entrenamiento.

4.1.2. Tipos de Red Recurrente

En la sección 3.3 se introducen tres tipos clásicos de RNN (implementados en forma nativa en `Pytorch`). En esta sección se presenta el desempeño de cada una, considerando los mismos parámetros del modelo *baseline* (tamaño y número de capas, *dropout*, etc.). En la tabla 3 se presetan los resultados obtenidos empleando una red GRU.

Tabla 3: Desempeño en validación del modelo *baseline* cambiando la red LSTM por una GRU.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,198	0,61	0,68	0,59
2	0,209	0,59	0,65	0,58
3	0,211	0,6	0,67	0,59
4	0,213	0,57	0,64	0,57
5	0,208	0,56	0,65	0,54
Promedio	0,2078	0,586	0,658	0,574

Cabe destacar que para una red Elman RNN `Pytorch` ofrece la posibilidad de reemplazar la no-linealidad de la ecuación 7 por la función `ReLU`. Por lo tanto en la tabla 4 se presentan los resultados obtenidos considerando la sigmoide, mientras que en la tabla 5 los obtenidos al emplear la función `ReLU` en el modelo.

Tabla 4: Desempeño en validación del modelo *baseline* cambiando la red LSTM por una Elman RNN con función de activación sigmoide.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,223	0,53	0,61	0,51
2	0,231	0,55	0,64	0,54
3	0,224	0,56	0,63	0,55
4	0,223	0,59	0,65	0,58
5	0,233	0,5	0,59	0,48
Promedio	0,2268	0,546	0,624	0,532

Tabla 5: Desempeño en validación del modelo *baseline* cambiando la red LSTM por una Elman RNN con función de activación `ReLU`.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,208	0,56	0,62	0,56
2	0,207	0,59	0,64	0,59
3	0,213	0,57	0,64	0,56
4	0,203	0,57	0,64	0,56
5	0,199	0,59	0,65	0,59
Promedio	0,206	0,576	0,638	0,572

De las tablas 4 y 5 se aprecia que hay una ganancia evidente y contundente en desempeño al incorporar `ReLU` como función de activación.

Los resultados presentados en las tablas 2, 3 y 5 son, en términos generales, similares; y las ganancias que una arquitectura ofrece en alguna métrica se ve compensada por una disminución en otra. Por lo que para no descartar una arquitectura en forma apresurada y sin explorar sus

hiperparámetros, en las pruebas posteriores se considerarán estos tres modelos: LSTM, GRU y Elman RNN (con `ReLU`).

4.1.3. Bidireccionalidad

En la presente sección se incorpora a los 3 modelos elegidos en la anterior de bidireccionalidad, teniéndose los resultados obtenidos con LSTM bidireccional en la tabla 6, con Elman RNN en la tabla 7 y con GRU en la tabla 8:

Tabla 6: Desempeño en validación del modelo *baseline* bidireccional.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,18	0,6	0,65	0,61
2	0,187	0,6	0,66	0,6
3	0,187	0,57	0,65	0,56
4	0,191	0,57	0,63	0,57
5	0,183	0,58	0,63	0,59
Promedio	0,1856	0,584	0,644	0,586

Tabla 7: Desempeño en validación del modelo Elman RNN bidireccional con `ReLU`.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,183	0,62	0,66	0,63
2	0,193	0,6	0,67	0,6
3	0,192	0,6	0,64	0,61
4	0,186	0,61	0,66	0,62
5	0,191	0,61	0,66	0,61
Promedio	0,189	0,608	0,658	0,614

Tabla 8: Desempeño en validación del modelo GRU bidireccional.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,181	0,62	0,68	0,62
2	0,181	0,6	0,66	0,6
3	0,195	0,6	0,67	0,6
4	0,178	0,62	0,68	0,61
5	0,18	0,62	0,68	0,61
Promedio	0,183	0,612	0,674	0,608

En comparación a los resultados de la sección anterior, para cada modelo se aprecia una ganancia general en desempeño, llegando incluso a obtener valores de F1-Score y Recall superiores a 0.6. Con esto se concluye empíricamente que la adición de bidireccionalidad permite encontrar

modelos más poderosos para la tarea estudiada. Una posible causa de esta mejora devenida con la bidireccionalidad es la adición de información contenida en todo el contexto de cada palabra x_i y ya no solo en la sub-secuencia previa $x_{1:i-1}$.

4.1.4. Tamaño de las Capas de las Redes Recurrentes

En esta sección se explora el efecto del tamaño de las capas de las redes recurrentes en los distintos modelos. Particularmente se consideran primeramente una reducción de la cantidad de neuronas, obteniéndo estados h_t de 64 dimensiones, y aumentarla a 256 y 512 unidades. Para el primer caso, se presentan los resultados obtenidos en las tablas 9, 10 y 11:

Tabla 9: Desempeño en validación del modelo LSTM bidireccional con estados de 64 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,2	0,61	0,65	0,62
2	0,199	0,6	0,67	0,6
3	0,207	0,54	0,62	0,54
4	0,204	0,58	0,64	0,59
5	0,203	0,6	0,66	0,59
Promedio	0,2026	0,586	0,648	0,588

Tabla 10: Desempeño en validación del modelo Elman RNN bidireccional con estados de 64 dimensiones y `ReLU`.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,19	0,57	0,63	0,57
2	0,183	0,59	0,65	0,59
3	0,198	0,58	0,65	0,58
4	0,195	0,57	0,63	0,58
5	0,198	0,53	0,6	0,53
Promedio	0,1928	0,568	0,632	0,57

Tabla 11: Desempeño en validación del modelo GRU bidireccional con estados de 64 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,196	0,57	0,65	0,577
2	0,186	0,58	0,64	0,58
3	0,185	0,61	0,67	0,61
4	0,202	0,61	0,67	0,6
5	0,188	0,57	0,63	0,57
Promedio	0,1914	0,588	0,652	0,5874

En términos generales, si se compara estos resultados con los obtenidos con cada tipo de red en la sección anterior se aprecia una disminución importante de desempeño. La única posible excepción es la LSTM en la cual se obtiene una mejora marginal en cuanto a las métricas, pero un deterioro en términos de pérdida. Esto indicaría que si bien se mejora la detección de entidades (reflejada en las métricas) se incurre en un aumento de los falsos positivos al clasificas palabras como entidades cuando estas corresponden a la categoría $'O'$.

Por lo tanto, se concluye que disminuir el tamaño de las capas de las redes recurrentes de modo tal que se obtengan estados 64-dimensionales se traduce en la obtención de peores modelos.

Por otra parte, los resultados obtenidos al aumentar la dimensionalidad de los estados h_t de 128 a 256 se presentan en las tablas 12, 13 y 14:

Tabla 12: Desempeño en validación del modelo LSTM bidireccional con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,182	0,63	0,68	0,63
2	0,187	0,57	0,64	0,56
3	0,193	0,6	0,67	0,59
4	0,173	0,62	0,67	0,62
5	0,179	0,62	0,67	0,63
Promedio	0,1828	0,608	0,666	0,606

Tabla 13: Desempeño en validación del modelo Elman RNN bidireccional con estados de 256 dimensiones y `ReLU`.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,411	0,44	0,58	0,4
2	0,35	0,45	0,56	0,43
3	0,407	0,46	0,57	0,43
4	0,358	0,54	0,64	0,51
5	0,419	0,49	0,6	0,47
Promedio	0,389	0,476	0,59	0,448

Tabla 14: Desempeño en validación del modelo GRU bidireccional con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,185	0,62	0,69	0,62
2	0,204	0,63	0,69	0,62
3	0,181	0,62	0,67	0,61
4	0,185	0,59	0,67	0,57
5	0,195	0,61	0,68	0,59
Promedio	0,19	0,614	0,68	0,602

De estos resultados se aprecia una una mejora en ambos modelos con compuertas, más importantes para el caso de LSTM que para GRU (cuya mejora en métricas es leve). Por su parte el modelo Elman RNN empeora considerablemente, llegando incluso a tener un peor desempeño en todos los aspectos si se compara con el modelo *baseline*.

El aumento aún mayor de neuronas en las redes recurrentes, pasando a estados de 512 dimensiones, se aprecia un notable deterioro en el desempeño de los modelos con compuertas, lo cual se muestra en las tablas 15 y 16. Para el caso de Elman RNN el optimizador no fue capaz de entrenar la red.

Tabla 15: Desempeño en validación del modelo LSTM bidireccional con estados de 512 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,187	0,59	0,65	0,59
2	0,187	0,57	0,64	0,57
3	0,18	0,61	0,67	0,62
4	0,187	0,54	0,62	0,54
5	0,184	0,61	0,68	0,61
Promedio	0,185	0,584	0,652	0,586

Tabla 16: Desempeño en validación del modelo GRU bidireccional con estados de 512 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,201	0,61	0,67	0,61
2	0,204	0,57	0,64	0,57
3	0,19	0,61	0,68	0,61
4	0,215	0,61	0,68	0,6
5	0,197	0,57	0,66	0,56
Promedio	0,2014	0,594	0,666	0,59

En vista de los resultados obtenidos en la presente sección para las siguientes se consideran las configuraciones con 256 dimensiones para las redes con compuertas y se mantienen las 128 de secciones anteriores para el modelo Elman RNN.

4.1.5. Learning Rate y Scheduler

En los experimentos realizados en las secciones anteriores se empleó `Adam` como optimizador con sus parámetros por defecto, en particular un *learning rate* de 0.01. En estos, se tiene que durante el entrenamiento el mejor modelo en cuanto a pérdida en validación se encuentra antes de las 5 épocas, sugiriendo que posiblemente el modelo podría no estar explorando lo suficiente el espacio de parámetros al converger demasiado rápido a una configuración.

Para explorar esta posibilidad se realizaron pruebas disminuyendo el *learning rate* a 0.0005 y, además, incorporando un `scheduler`. Particularmente, para el `scheduler` se prueba que cada 2 y 3 épocas el *learning rate* decaiga según un factor de 0.9 y 0.75. Tras probar estas cuatro combinaciones

posibles se obtienen resultados prácticamente idénticos a los exhibidos en la sección anterior, por lo que se concluye que una disminución del *learning rate* no conlleva un mejor aprendizaje o capacidad de encontrar un mejor modelo.

Una justificación para este comportamiento es que en principio el `scheduler` está pensado para acelerar el entrenamiento de modelos grandes. La idea es que para modelos que requieren de muchas épocas para ser entrenados y por ende necesitan una gran cantidad de tiempo para converger si se considera una tasa de aprendizaje que permita explorar el espacio de parámetros con cierta fineza. Por lo tanto, en etapas tempranas del entrenamiento se plantean tasas de aprendizaje altas que permitan llegar rápidamente a un entorno de parámetros adecuado, para luego reducir la tasa de aprendizaje para explorar dicho entorno con mayor precisión. También es posible reducir el *learning rate* en forma progresiva.

No obstante, como ya se vio que el entrenamiento de los modelos estudiados converge rápidamente, el uso de un `scheduler` como el presentado en el párrafo anterior pierde sentido. Luego, disminuir el *learning rate* solo lleva a desempeños prácticamente idénticos en una mayor cantidad de tiempo; mientras que por el otro lado aumentar la tasa de aprendizaje solo puede reducir la capacidad del optimizador para explorar el espacio de parámetros debido a que los saltos pueden ser muy grandes en magnitud.

Por lo tanto, en los experimentos de las secciones siguientes se mantiene el optimizador usado hasta el momento, y sin un `scheduler` para el *learning rate*.

4.1.6. Profundidad de las Redes Recurrentes

En esta sección se experimenta con la profundidad de la red recurrente, es decir con la cantidad de redes recurrentes apiladas en el modelo. Dado que el modelo *baseline* cuenta con dos redes apiladas, en la tablas 17, 18 y 19 se muestran los resultados obtenidos con las 3 arquitecturas escogidas hasta el momento, pero empleando tan solo una capa, es decir solo una red recurrente:

Tabla 17: Desempeño en validación del modelo LSTM bidireccional de profundidad 1 con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,188	0,6	0,68	0,6
2	0,185	0,6	0,67	0,6
3	0,188	0,58	0,65	0,57
4	0,178	0,6	0,66	0,6
5	0,184	0,59	0,65	0,58
Promedio	0,1846	0,594	0,662	0,59

Tabla 18: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 1 con estados de 128 dimensiones y `ReLU`.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,186	0,59	0,66	0,58
2	0,185	0,62	0,69	0,6
3	0,183	0,61	0,67	0,6
4	0,183	0,59	0,66	0,59
5	0,174	0,6	0,66	0,6
Promedio	0,1822	0,602	0,668	0,594

Tabla 19: Desempeño en validación del modelo GRU bidireccional de profundidad 1 con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,177	0,62	0,68	0,61
2	0,192	0,57	0,65	0,55
3	0,185	0,58	0,66	0,58
4	0,176	0,6	0,67	0,6
5	0,182	0,58	0,65	0,57
Promedio	0,1824	0,59	0,662	0,582

Al emplear los modelos con 1 capa recurrente se tiene un ligero deterioro en el desempeño, pudiendo ser la excepción la arquitectura Elman RNN en la cual se obtienen ciertas mejoras en algunas métricas, estas no superan en magnitud a las pérdidas incurridas las otras.

Habiendo concluido que disminuir el número de redes recurrentes empleadas, en las tablas 20, 21 y 22 se muestran los resultados obtenidos al apilar 3 redes recurrentes en lugar de las 2 consideradas en las secciones anteriores:

Tabla 20: Desempeño en validación del modelo LSTM bidireccional de profundidad 3 con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,186	0,57	0,64	0,58
2	0,198	0,55	0,62	0,56
3	0,198	0,57	0,63	0,59
4	0.184	0,57	0,63	0,58
5	0,188	0,57	0,64	0,57
Promedio	0,1925	0,566	0,632	0,576

Tabla 21: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 3 con estados de 128 dimensiones y `ReLU`.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,191	0,62	0,66	0,63
2	0,198	0,6	0,65	0,61
3	0,194	0,57	0,62	0,58
4	0,197	0,63	0,67	0,63
5	0,195	0,6	0,65	0,62
Promedio	0,195	0,604	0,65	0,614

Tabla 22: Desempeño en validación del modelo GRU bidireccional de profundidad 3 con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,194	0,61	0,67	0,61
2	0,197	0,57	0,64	0,56
3	0,19	0,62	0,68	0,63
4	0,196	0,58	0,64	0,58
5	0,184	0,63	0,69	0,63
Promedio	0,1922	0,602	0,664	0,602

Al comparar estos resultados con los obtenidos con los modelos de profundidad 2 se aprecia que las arquitecturas con compuertas se ven severamente dañadas en desempeño. Para el caso Elman RNN esta se muestra más robusta, en el sentido que las pérdidas son de menor magnitud manteniendo incluso su desempeño en cuanto a recall. Aún así, se aprecia que con la adición de una tercera capa recurrente los modelos son menos capaces de desempeñarse en la tarea.

Una posible explicación de esto puede ser que al aumentar la cantidad de parámetros considerados, la exploración en el espacio de estos se complicaría y por ende es más difícil encontrar un punto óptimo en dicho espacio.

Para verificar esta hipótesis se aumenta nuevamente la cantidad de redes apiladas en cada modelo, utilizando 4 capas de profundidad. Estos resultados se observan en las tablas 23, 24 y 25:

Tabla 23: Desempeño en validación del modelo LSTM bidireccional de profundidad 4 con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,2	0,59	0,65	0,59
2	0,187	0,59	0,64	0,6
3	0,204	0,56	0,63	0,56
4	0,2	0,62	0,66	0,64
5	0,203	0,56	0,63	0,56
Promedio	0,1988	0,584	0,642	0,59

Tabla 24: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 4 con estados de 128 dimensiones y `ReLU`.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,2	0,58	0,63	0,6
2	0,193	0,59	0,63	0,59
3	0,2	0,57	0,63	0,58
4	0,212	0,53	0,58	0,54
5	0,198	0,61	0,65	0,62
Promedio	0,2006	0,576	0,624	0,586

Tabla 25: Desempeño en validación del modelo GRU bidireccional de profundidad 4 con estados de 256 dimensiones.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,2	0,55	0,61	0,56
2	0,213	0,55	0,61	0,55
3	0,199	0,55	0,62	0,55
4	0,214	0,61	0,67	0,62
5	0,202	0,59	0,65	0,58
Promedio	0,2056	0,57	0,632	0,572

En estos últimos resultados se muestra un deterioro aún mayor en el desempeño respecto a lo obtenidos con 2 y 3 redes apiladas.

Habiendo visto que tanto con 1, 3 y 4 capas apiladas los modelos muestran un peor desempeño al compararse con los obtenidos previamente con 2 capas, en las siguientes secciones se consideran arquitecturas con 2 redes apiladas.

4.1.7. Dropout

Como se discutió en secciones anteriores, los modelos empleados hasta ahora poseen una componente de *dropout* con parámetro 0.25 entre la capa de embedding y la red recurrente, entre los estados de la propia red recurrente y entre la salida de esta y la red de clasificación.

Para estudiar el efecto del *dropout* en las distintas partes del modelo se muestran primero en las tablas 26, 27 y 28 los resultados obtenidos sin ninguna componente de *dropout*:

Tabla 26: Desempeño en validación del modelo LSTM bidireccional de profundidad 2 con estados de 256 dimensiones. Sin *dropout*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,191	0,59	0,66	0,59
2	0,202	0,57	0,66	0,55
3	0,191	0,58	0,65	0,56
4	0,18	0,6	0,66	0,6
5	0,189	0,59	0,67	0,58
Promedio	0,1906	0,586	0,66	0,576

Tabla 27: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 2 con estados de 128 dimensiones y `ReLU`. Sin *dropout*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,195	0,6	0,67	0,61
2	0,201	0,58	0,64	0,58
3	0,2	0,59	0,65	0,59
4	0,2	0,61	0,67	0,61
5	0,205	0,59	0,65	0,59
Promedio	0,2002	0,594	0,656	0,596

Tabla 28: Desempeño en validación del modelo GRU bidireccional de profundidad 2 con estados de 256 dimensiones. Sin *dropout*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,19	0,59	0,65	0,59
2	0,19	0,6	0,67	0,59
3	0,204	0,58	0,67	0,56
4	0,19	0,6	0,67	0,59
5	0,184	0,61	0,67	0,62
Promedio	0,1916	0,596	0,666	0,59

Cabe notar que al eliminar toda componente de *dropout* de los modelos se obtiene un peor desempeño en cada uno de los modelos. No obstante, estos resultados sirven como base para analizar el impacto de cada una de las componentes descritas. En lo que resta del trabajo y por claridad se denota por *input dropout* a la capa de *dropout* tras la capa de *embedding* y la red recurrente, *rnn dropout* al *dropout* interno de la red recurrente que se aplica a los estados ocultos h_t y por *output dropout* al *dropout* aplicado a la salida de la red recurrente antes de alimentar la red de clasificación.

Otro resultado interesante de los entrenamientos sin componente de *dropout* alguna es que al momento de detener este según el criterio de *early stopping* los modelos presentaban un sobreajuste

considerable en comparación al exhibido en experimentos anteriores, lo cual se condice con las propiedades de *dropout* como regularizador.

Dicho lo anterior, se probaron los siguientes valores de *input dropout*: 0.1, 0.25, 0.5 y 0.75. Por simplicidad se muestran en las tablas 29, 30 y 31 el desempeño de los modelos basados en LSTM, Elman RNN y GRU con un *input dropout* de 0.75, que se concluyó que fue la mejor *performance* en promedio.

Tabla 29: Desempeño en validación del modelo LSTM bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,161	0,67	0,72	0,66
2	0,16	0,66	0,7	0,66
3	0,162	0,65	0,7	0,64
4	0,163	0,64	0,68	0,65
5	0,161	0,64	0,69	0,63
Promedio	0,1614	0,652	0,698	0,648

Tabla 30: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 2 con estados de 128 dimensiones y `ReLU`. Con 0.75 de *input dropout*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,166	0,63	0,68	0,64
2	0,16	0,65	0,69	0,66
3	0,17	0,61	0,67	0,61
4	0,165	0,62	0,66	0,64
5	0,16	0,63	0,68	0,63
Promedio	0,1642	0,628	0,676	0,636

Tabla 31: Desempeño en validación del modelo GRU bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,167	0,63	0,69	0,62
2	0,166	0,64	0,7	0,64
3	0,165	0,61	0,67	0,61
4	0,159	0,64	0,7	0,64
5	0,166	0,63	0,69	0,63
Promedio	0,1646	0,63	0,69	0,628

Para los tres tipos de redes recurrentes se aprecia una mejora considerable y en cada una de las

métricas en comparación tanto con los resultados de las tablas 26, 27 y 28 como con los obtenidos en secciones anteriores con 0.25 de *input dropout*, *rnn dropout* y *output dropout*.

Respecto a la componente de *rnn dropout* se experimentó con los mismos valores (0.1, 0.25, 0.5 y 0.75), con los cuales se obtuvo un deterioro del desempeño en cada uno de los aspectos. La única excepción encontrada es el caso del modelo basado en GRU con un *rnn dropout* de 0.5, obteniendo los resultados presentes en la tabla 32:

Tabla 32: Desempeño en validación del modelo GRU bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout* y 0.5 *rnn dropout*.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,162	0,64	0,7	0,63
2	0,169	0,64	0,69	0,64
3	0,163	0,65	0,7	0,64
4	0,167	0,64	0,69	0,64
5	0,166	0,65	0,7	0,66
Promedio	0,1654	0,644	0,696	0,642

En este último caso, si bien existe un deterioro en términos de pérdida en validación también se observa una ganancia considerable en cuando a F1-Score y Recall y en menor medida en Precisión. Dado que este modelo presenta resultados altamente competitivos con los mostrados en las tablas 29, 30 y 31 en las secciones posteriores se consideran estas cuatro configuraciones, estudiando entonces un modelo basado en LSTM, uno en Elman RNN y dos en GRU.

4.1.8. Uso de Embeddings

En la presente sección se hace un análisis del impacto de la capa de *embedding* en el desempeño de los modelos, considerando las 4 configuraciones dadas en la sección anterior. Para esto se varía el tamaño de los *embeddings* y su fuente, ya sea entrenar desde cero la capa de *embedding*, inicializarla y fijarla mediante *embeddings* pre-entrenados y, finalmente usar estos *embeddings* pre-entrenados solo como inicialización y realizar un *fine-tunning* de estos.

Para estos experimentos se toman en consideración los *embeddings* pre-entrenados en español disponibles en [3]. Estos fueron entrenados por medio de `FastText` obteniendo representaciones de 10, 30, 100 y 300 dimensiones. Más aún, se disponen de 2 matrices de *embedding* cuyos vectores son de dimensionalidad 300, diferenciándose en el tamaño del vocabulario que las conforman. Para más claridad se denominan a los *embeddings* pre-entrenados según su dimensionalidad, denotando por XS a los de dimensionalidad 10, S a los de dimensionalidad 30, M a los de dimensión 100, L al primer set de *embeddings* pre-entrenados de dimensionalidad 300 (con 1313423 *wordvectors*) y L' al segundo set de *embeddings* pre-entrenados de dimensionalidad 300 (con 1451827 *wordvectors*).

Debido a que se experimenta con 4 arquitecturas diferentes, 4 tamaños distintos para los *embeddings* y 5 fuentes diferentes de *embeddings* pre-entrenados se estudiaron cerca de 60 configuraciones considerando entrenar la capa de *embbedding* inicializada aleatoriamente, usar los pre-entrenados y re-entrenar estos; cada una en 5 pruebas para obtener los resultados promedio, por simplicidad se presentan solo algunos de los resultados obtenidos, con el fin de ilustrar las tendencias generales.

En primer lugar se observó que para cada una de las 4 arquitecturas consideradas se obtiene un peor desempeño al inicializar la capa de *embedding* con los vectores pre-entrenados y se congela esta capa, entrenando solamente los pesos de las redes recurrentes y de clasificación. Si los vectores pre-entrenados se utilizan como mera inicialización y se continúa entrenando la capa de *embedding* a partir de esta inicialización se obtiene una mejora contundente respecto a la congelación de esta capa, pero sin superar el desempeño obtenido al entrenar la capa con inicialización aleatoria. Ejemplos de esto sobre la arquitectura LSTM con *embeddings* de dimensionalidad 100 se observan en las tablas 33, 34, 35; y en las tablas 36, 37, 38 para la misma arquitectura pero con *embeddings* de dimensionalidad 300:

Tabla 33: Desempeño en validación del modelo LSTM bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout* y *embeddings* de dimensionalidad 100.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,162	0,65	0,7	0,64
2	0,164	0,64	0,7	0,64
3	0,158	0,64	0,68	0,65
4	0,164	0,64	0,7	0,63
5	0,159	0,64	0,69	0,64
Promedio	0,1614	0,642	0,694	0,64

Tabla 34: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 2 con estados de 128 dimensiones y `ReLU`. Con 0.75 de *input dropout* y *embeddings* pre-entrenados (M) de dimensionalidad 100.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,29	0,53	0,59	0,55
2	0,272	0,53	0,58	0,55
3	0,261	0,56	0,61	0,57
4	0,268	0,51	0,55	0,53
5	0,289	0,54	0,58	0,56
Promedio	0,276	0,534	0,582	0,552

Tabla 35: Desempeño en validación del modelo GRU bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout* y *embeddings* re-entrenados (M) de dimensionalidad 100.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,212	0,6	0,64	0,61
2	0,254	0,63	0,66	0,65
3	0,282	0,6	0,65	0,62
4	0,25	0,61	0,64	0,63
5	0,253	0,64	0,67	0,66
Promedio	0,2502	0,616	0,652	0,634

Tabla 36: Desempeño en validación del modelo LSTM bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout* y *embeddings* de dimensionalidad 300.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,166	0,64	0,68	0,64
2	0,166	0,62	0,66	0,63
3	0,165	0,63	0,69	0,64
4	0,16	0,64	0,69	0,64
5	0,167	0,63	0,69	0,63
Promedio	0,1648	0,632	0,682	0,636

Tabla 37: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 2 con estados de 128 dimensiones y `ReLU`. Con 0.75 de *input dropout* y *embeddings* pre-entrenados (L) de dimensionalidad 300.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,267	0,53	0,58	0,54
2	0,27	0,56	0,6	0,58
3	0,274	0,53	0,57	0,55
4	0,269	0,53	0,58	0,54
5	0,258	0,51	0,56	0,52
Promedio	0,2676	0,532	0,578	0,546

Tabla 38: Desempeño en validación del modelo GRU bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout* y *embeddings* re-entrenados (L) de dimensionalidad 300.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,255	0,61	0,64	0,63
2	0,255	0,64	0,67	0,67
3	0,259	0,64	0,67	0,67
4	0,252	0,6	0,65	0,63
5	0,24	0,63	0,66	0,65
Promedio	0,2522	0,624	0,658	0,65

De estos resultados se determina que el impacto de los *embeddings* obtenidos de [3] aplicados en la presente tarea y con los datos de la competencia es insuficiente como para significar una mejora sustancial. Luego, es natural que la elección de los mejores modelos en esta ocasión se reduce a variar la dimensionalidad de los *embeddings* entrenados en la capa correspondiente a partir de una inicialización aleatoria.

Particularmente, para el caso de la arquitectura con LSTM el mejor desempeño se obtiene entrenando *embeddings* de 100 dimensiones, que se muestra en la tabla 33. Para el caso de arquitecturas basadas en Elman RNN el mejor desempeño también se obtuvo entrenando *embeddings* 100-dimensionales a partir de una inicialización aleatoria, resutados que se muestran la tabla 39.

Tabla 39: Desempeño en validación del modelo Elman RNN bidireccional de profundidad 2 con estados de 128 dimensiones. Con 0.75 de *input dropout* y *embeddings* de dimensionalidad 100.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,164	0,65	0,69	0,66
2	0,166	0,64	0,68	0,64
3	0,164	0,65	0,69	0,66
4	0,167	0,64	0,67	0,65
5	0,168	0,65	0,69	0,66
Promedio	0,1658	0,646	0,684	0,654

Para el caso de las arquitecturas basadas en GRU el mejor desempeño obtenido en términos de métricas se presenta en la tabla 40, considerando una capa de *embeddings* de 300 dimensiones, *input dropout* de 0.75 y *rnn dropout* de 0. No obstante, estas ganancias van acompañadas de un deterioro considerable en *loss*. Luego, el resultado más balanceado para GRU se obtuvo con la configuración presentada en la tabla 32.

Tabla 40: Desempeño en validación del modelo GRU bidireccional de profundidad 2 con estados de 256 dimensiones. Con 0.75 de *input dropout* y *embeddings* de dimensionalidad 300.

Prueba	Loss	F1-Score	Precisión	Recall
1	0,169	0,62	0,68	0,62
2	0,166	0,65	0,71	0,64
3	0,181	0,65	0,71	0,65
4	0,177	0,65	0,7	0,65
5	0,176	0,66	0,71	0,65
Promedio	0,1738	0,646	0,702	0,642

Finalmente, comparando las mejores configuraciones encontradas, se concluye que aquel modelo basado en este tipo de redes que mejor resuelve la tarea de *Named Entity Recognition* propuesta consiste en una red LSTM bidireccional de profundidad 2 y capas recurrentes de tamaño 256, con un *input dropout* de 0.75 aplicado sobre *embeddings* de 100 dimensiones entrenados en paralelo con el resto del modelo. El desempeño de este modelo se mostró en la tabla 33 y 29. Que los resultados obtenidos sean tan similares demuestra además que el modelo es consistente y ofrece baja varianza en términos de desempeño, lo cual es a su vez una posible ventaja respecto a otros modelos.

4.2. Transformers

Como los modelos *transformers* contienen una enorme cantidad de parámetros, se desea utilizar un modelo pre-entrenado sobre un corpus amplio. Debido a que el conjunto de datos provisto se encuentra en lenguaje español, se deben escoger modelos pre-entrenados en el mismo idioma, y en nuestro conocimiento, el modelo BERT-base es el único modelo pre-entrenado sobre un amplio corpus en español, por tanto, los experimentos se limitaron a la implementación y variación paramétrica de este modelo. Si bien, se encuentran disponibles modelos *transformers* multi-lenguaje, en [1] muestran que el desempeño de BERT en español (BETO) presenta mejor desempeño que los demás, atribuyéndosele a su entrenamiento con datos de contexto idiomático específico, lo cual sustenta la decisión anteriormente expuesta.

El modelo BERT-base se compone de 12 capas de auto-atención con 16 capas de atención cada una, con una dimensión de representación oculta de 1024, resultando en un modelo de 110 millones de parámetros, el cual se encuentra pre-entrado bajo las tareas descritas en la sección 3 usando un corpus en idioma español. Para la tarea de reconocimiento de entidades nombradas, se agrego una capa MLP para la clasificación de las representación a nivel de *token* obtenidas por el modelo de dimensión $\mathbb{R}^{1024\times 9}$ (dimensión de representación oculta $\times$ cantidad de clases). Como la tokenización del modelo separa ciertas palabras en *sub-tokens* (ej. `playing` $\rightarrow$ `play`, `##ing`), se generan una cantidad superior de *tokens* respecto a la cantidad de etiquetas por palabra provistas, por lo que se le asignó la etiqueta original al primer *sub-token*, asignándole una etiqueta auxiliar `<sub>` a los demás *sub-tokens* de una misma palabra, los cuales son ignorados en la etapa de entrenamiento y en el cálculo de la métricas de evaluación.

El modelo en cuestión fue re-entrenado completamente para la tarea de reconocimiento de entidades nombradas utilizando el conjunto de datos provisto (*fine-tuning*), respetando la elección

de hiper-parámetros (*dropout*, funciones de activación, optimizador *Adam* con momentum) utilizada en el pre-entrenamiento para evitar impactar negativamente el desempeño del modelo. Sin embargo, se exploraron variaciones de hiperparámetros menos impactantes en la etapa de pre-entrenamiento, tales como la utilización de las variantes *cased/uncased*, el tamaño de batch y el largo de las secuencias de entrada.

4.2.1. BERT cased/uncased

Se exploró utilizando la versión del modelo capaz de distinguir caracteres en mayúsculas (*cased*) y la que trata los caracteres indistintamente (*uncased*). Se observó que el modelo BERT-textituncased presentó un paupérrimo desempeño en la tarea NER (incluso peor a los desempeños presentados en las secciones anteriores), sin embargo, el modelo BERT-*cased* presento un desempeño bastante superior a los modelos presentados en las secciones anteriores, lo cual se le atribuye a la gran capacidad de representación de dependencias temporales extensas, permitiendo extraer información contextual altamente informativa del texto no estructurado en idioma español. La radical diferencia entre los desempeños de los modelos BERT-*cased* y BERT-*uncased* radica en que el uso de mayúsculas es decisivo para la detección de entidades, ya que en la gran mayoría de los casos, las entidades presentan un caracter inicial en mayúsculas, por lo tanto, ignorar deliberadamente esta fundamental característica induce un pésimo desempeño en la tarea de reconocimiento de entidades nombradas.

4.2.2. Largo de Secuencias de Entrada

Como el modelo BERT recibe una secuencia de entradas de tamaño fijo (con tamaño máximo 512), los datos deben ser truncados a un determinado largo de secuencia. Se exploraron secuencias de tamaño `max_len` $= \{100, 200, 300, 512\}$. No se observaron diferencias significativas en el desempeño al variar el valor de `max_len`, sin embargo, los tiempos de entrenamiento se aumentaron proporcionalmente con el valor de `max_len`. El mejor desempeño se obtuvo con `max_len` $= 512$, presentando una diferencia aproximada de 0.1 en cada métrica de evaluación respecto a los demás valores del hiperparámetro, lo cual resulta esperable, ya que al truncar anticipadamente una serie de tiempo se pierde información sobre la dependencia temporal de la misma en ciertos segmentos, por lo cual, el valor que menos altera las secuencias (largo de secuencia máximo aceptado por el modelo) es el que debiera preservar de mejor manera la información de los datos, y en consecuencia, obtener un mejor desempeño.

4.2.3. Tamaño de Batch

Se exploraron distintos tamaños de batch $n = \{8, 16, 32\}$. No se apreciaron diferencias significativas en el desempeño obtenido al variar dicho hiperparámetro, sin embargo, los tiempos de entrenamientos resultaron inversamente proporcionales al tamaño de batch escogido y la cantidad de memoria requerida resultó directamente proporcional al mismo. Por tanto se decidió escoger el tamaño de batch máximo de acuerdo al valor `max_len` $= 512$, escogido en la sección anterior, tal que evite consumir la totalidad de memoria RAM disponible, siendo el valor $n = 16$ el tamaño de batch seleccionado.

4.3. Selección Final

En consecuencia los numerosos experimentos y exhaustivos análisis presentados anteriormente, el modelo de mejor desempeño resultó ser el modelo BERT-*cased* re-entrenado sobre el conjunto de datos provisto (*fine-tuning*), preservando los hiperparámetros utilizados en el pre-entrenamiento para evitar impactar negativamente en el desempeño, ajustando la secuencia de entrada al tamaño máximo permitido `max_len` = 512 y ajustando el tamaño de batch en función de la reducción de tiempos de entrenamiento sujeto a la cantidad de memoria disponible.

El modelo seleccionado obtuvo el primer lugar de la competencia en las métricas *F1-score* y *precision*, alcanzando un valor de 0.76 y 0.81, respectivamente, y el segundo lugar de la competencia en la métrica *recall* alcanzando un valor de 0.72.

5. Conclusiones

En el presente trabajo se realizaron una gran cantidad de experimentos mediante la utilización de distintos modelos de *deep learning* basados principalmente en redes neuronales recurrentes, *transformers* y *embeddings* pre-entrenados, comparando los desempeños obtenidos de acuerdo a las métricas *F1-score*, *precision* y *recall* en la tarea de reconocimiento de entidades nombradas en oraciones en idioma español.

Los experimentos realizados se centraron en comparar los desempeños obtenidos de los distintos modelos frente a variaciones en sus hiperparámetros, mostrando que algunos de estos presentaron un alto impacto en el desempeño de la tarea a resolver, tales como la adición de bi-direccionalidad y elección de función de activación en redes recurrentes y la utilización de un modelo *cased* o *uncased* en el modelo *transformer* BERT, otros impactaron marginalmente en su desempeño, tales como el tamaño de batch y el largo de secuencia de entradas en el modelo *transformer* BERT, y en cambio otros, no presentaron impacto alguno en el desempeño de los modelos, tales como la tasa de aprendizaje y su *scheduler* en el entrenamiento.

Se concluye que la elección de un modelo determinado es fundamental para lograr buenos desempeños en una tarea determinada, y su elección requiere de un profundo conocimiento de las capacidades y limitaciones de los mismos, en conjunto con una adecuada elección de sus hiperparámetros y un cabal conocimiento de la tarea a resolver y del tipo, cantidad y calidad de los datos disponibles. La elección de los hiperparámetros resultará fundamental en el desempeño del modelo escogido, y dependerá de la tarea y de los datos utilizados, por lo cual, siempre se requerirá un ajuste de los mismos por medio de un entendimiento cabal de su impacto en el modelo, pudiendo afectar en el desempeño, en la velocidad de entrenamiento e inferencia o en la cantidad de recursos computacionales demandados, requiriendo adicionalmente de la realización de una exhaustiva y planificada serie de experimentos que permitan encontrar la configuración óptima de los mismos.

Se evidenció la gran capacidad de los algoritmos de *deep learning* para la resolución de tareas de aprendizaje supervisado, permitiendo desempeños inalcanzables con técnicas de *machine learning* clásicas basadas en la selección manual de características relevantes. Los algoritmos de *deep learning* permiten extraer las características relevantes para la resolución de una determinada tarea directamente de los datos, evitando el proceso de selección manual de características, la cual, dependiendo de la dificultad de la tarea, resulta completamente no trivial, como es le caso del reconocimiento de entidades nombradas desarrollado en el presente trabajo. En este contexto, los modelos neuronales, en virtud de su alta capacidad de representar cualquier función parametrizada por sus enorme cantidad de parámetros, resultan ideales para atacar problemas de aprendizaje supervisado, y en particular, los modelos neuronales recurrentes y los *transformers*, resultan idóneos para tareas que involucran series de tiempo, cuya capacidad de representación contextual y modelamiento de altas dependencias temporales, resultan una elección idóneos para la resolución de problemas de procesamiento de lenguaje natural.

Como trabajo futuro para mejorar el desempeño en la tarea descrita se propone explorar la incorporación de una red neuronal recurrente bi-direccional como reemplazo de la capa de clasificación lineal utilizada en el modelo BERT, unificando los dos mejores modelos encontrados en el desarrollo experimental descrito, explorando adicionalmente la utilización de la función de costo *Conditional Random Fields* (CRF) [10], ampliamente utilizada en problemas de etiquetado de

secuencias. El nuevo modelo propuesto BERT+BiLSTM+CRF, originado a partir de la combinación de distintas técnicas exploradas y no exploradas en este trabajo podría presentar mejoras respecto al modelo BERT, o bien, podría resultar redundante, obteniendo diferencias marginales de desempeño respecto a este, siendo de todas formas una variante interesante de exploración experimental.

6. Referencias

[1] J. Cañete, G. Chaperon, R. Fuentes, and J. Pérez, "Spanish pre-trained bert model and evaluation data," in *to appear in PML4DC at ICLR 2020*, 2020.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.

[3] J. Cañete. Spanish word embeddings. [Online]. Available: https://github.com/BotCenter/spanishWordEmbeddings

[4] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," 2016.

[5] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[7] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[10] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[12] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[13] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, 2019, pp. 5754–5764.

[14] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.