Procesamiento del Lenguaje Natural

Tema 9. Aplicaciones del procesamiento del lenguaje natural

Índice

Esquema

APLICACIONES DEL PROCESAMIENTO DEL LENGUAJE NATURAL

Traducción automática

- **Divergencias del lenguaje:** la traducción entre dos lenguas no equivale a traducción de palabras individuales.
- **Modelo *encoder-decoder*:** generar la representación de la secuencia entrada del *encoder* (contexto) y obtener la traducción de esa entrada desde el *decoder*.
- **Diseño de *encoder-decoder*:** usando RNN o modelos de *transformers*.
- **Otros aspectos:** evaluación de la traducción considerando fidelidad, fluencia y posibles sesgos.

Autocompletado y generación automática de resúmenes

- **Arquitectura:** modelo *encoder-decoder* tanto para autocompletado como para generación de resúmenes.
- **Diferencias:** en el caso del autocompletado se tiene un LM causal, en el caso de la generación de resúmenes no es necesario.
- **Entrenamiento:** usando el concepto de *teacher forcing*.

Análisis de sentimientos

- **Análisis de sentimiento:** detectar el sentimiento expresado por el autor de un texto hacia un determinado objeto, pudiendo ser por ejemplo, un sentimiento negativo o positivo.
- **Preprocesado:** obtención de rasgos (ej.: BoW) desde textos.
- **Entrenamiento:** algoritmos de ML de clasificación supervisados.
- **Otras posibilidades:** uso de lexicones con sentimientos a nivel de palabra para inferir el sentimiento del texto.

Question Aswering

- **IR:** técnicas de recuperación de información. *Ad hoc retrieval,* donde se reciben consultas de usuarios y se busca responder a ellas utilizando un corpus.
- **Técnicas:** representación de consultas y documentos con BoW con *embeddings*.
- **IR para *factoid QA*:** dada una consulta y un texto, encontrar el pasaje del texto que responde a la pregunta.

Reconocimiento automático del habla y *text-to-speech*

- **Dificultades de un ASR:** tamaño del vocabulario, hacia quién se está comunicando el usuario, el canal de comunicación y el ruido, y los acentos u otras características.
- **Arquitectura ASR:** modelo *encoder-decoder* extrayendo rasgos de la señal acústica (entrada) y generando como salida el texto asociado.
- **Arquitectura TTS:** texto de entrada, normalizado (vocalización), generación del espectrograma y módulo de vocoding.

9.1. Introducción y objetivos

A continuación, se introducirá algunas de las aplicaciones industriales más habituales basadas en PLN. En primer lugar, se presentarán los **sistemas de traducción automática,** donde se hablará de algunas de las técnicas basadas en redes neuronales más actuales, cómo poder evaluar los resultados obtenidos, así como otros aspectos éticos para tener en cuenta en el desarrollo del modelo, como es el tema de los sesgos. Posteriormente, se hablará de los **sistemas de autocompletado de textos y de generación automática de resúmenes.**

También, incluye una **presentación al problema del análisis de sentimientos,** así como una explicación sobre elementos para tener en cuenta en su desarrollo. Tras ello, se verá una de las **aplicaciones más relevantes de PLN** en la industria: los sistemas de *Question Answering,* con los que se busca responder a consultas de los usuarios en base a la información disponible en una colección de documentos.

Finalmente, se verán las aplicaciones de PLN tanto para generar texto desde audio, como para generar audio desde texto.

Objetivos

- Entender cómo funciona un sistema de traducción automática, así como contar con las nociones clave de qué algoritmos basados en redes neuronales se pueden usar para diseñarlo.
- Saber cómo se construyen los sistemas de autocompletado y de generación automática de resúmenes, y qué algoritmos basados en redes neuronales se pueden usar para desarrollarlos.
- Conocer los conceptos básicos del problema de análisis de sentimientos, así como entender algunos de los elementos clave necesarios para construir un sistema que pueda clasificar sentimientos en un texto.

Ideas clave

- Entender cómo funciona un sistema de *Question Answering* en el contexto del campo de la recuperación de información, y qué algoritmos de PLN se pueden usar para desarrollarlos.
- Conocer cómo funcionan los sistemas de reconocimiento automático del habla y los de *text-to-speech.*

9.2. Traducción automática

En este capítulo se va a presentar la tarea de PLN de **traducción automática** *(Machine Translation,* MT), junto con algunos aspectos para tener en cuenta y las arquitecturas habituales que se usan para implementarla. El problema que se busca abordar con MT es, partiendo de un texto en una lengua, poder **generar automáticamente su traducción a otra lengua.** Ahora bien, como se comentará a continuación, una traducción precisa no pasa por convertir las palabras individuales de un texto en un lenguaje a su equivalente en otro, sino que se debe tener en cuenta toda la **información de la secuencia del texto** para hacer su traducción.

Divergencias del lenguaje

De cara a las tareas de MT es importante tener en cuenta que, a pesar de que existen elementos universales del lenguaje comunes a todas las lenguas, existen también **diferencias entre ellas.** Estas diferencias pueden ser a nivel de palabras concretas, lo que hace referencia a aspectos **idiosincrático o léxicos** (como, por ejemplo, palabras que significan cosas distintas en según qué lengua), o pueden ser a **nivel sistemático** afectando a toda la estructura de una frase (por ejemplo, el orden en el que se sitúan los verbos y los objetos directos). Estas diferencias hacen que la traducción de un texto de una lengua a otra sea una tarea compleja, donde no es suficiente con traducir las palabras individualmente, sino que hay que tener en cuenta todo el contexto y estructura de la frase.

Para completar el estudio de esta sección y ver en detalle los distintos aspectos que diferencias entre sí las lenguas, puedes leer el Capítulo 10 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall. https://web.stanford.edu/~jurafsky/slp3/10.pdf

Modelo encoder-decoder

En el ámbito de PLN se han propuesto arquitecturas específicas de aprendizaje profundo de cara a poder realizar tareas de MT teniendo en cuenta las singularidades de cada lengua y las diferencias que existen entre ellas cuando se quieren realizar traducciones. Este es el caso de la **arquitectura *encoder-decoder,*** también conocida como *sequence-to-sequence* (seq2seq), que también es utilizada en otras tareas de PLN como la **generación de resúmenes o la generación de respuestas.**

El esquema de una arquitectura *encoder-decoder* aparece en la siguiente imagen. Se puede apreciar que estos modelos reciben la secuencia de datos de entrada dentro de una primera capa *(encoder)* con la que se genera una **representación (contexto).** Esta representación sirve de entrada para la segunda **capa (decoder)** con la que se genera una secuencia de salida.

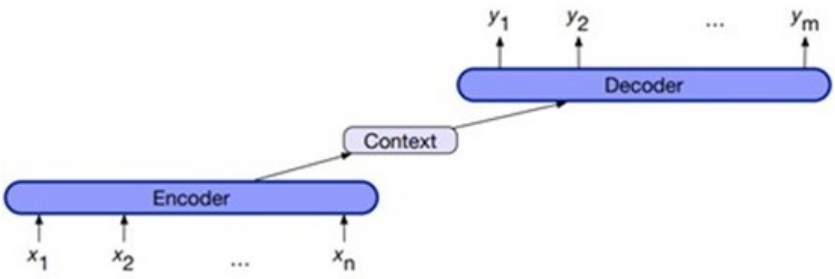

Figura 1. Esquema *encoder-decoder.* Fuente: Jurafsky, 2021.

Modelo *encoder-decoder* con RNN

Este esquema se puede construir con distintos modelos de aprendizaje profundo, como es el caso de las Redes Neuronales Recurrentes (*Recurrent Neural Networks*, RNN). Esta aproximación se muestra en la siguiente imagen. Como se puede apreciar, la secuencia de texto original sirve como entrada de la capa *encoder* convirtiendo cada palabra en un vector (ej., su *word embedding*). La última salida de la capa *encoding* corresponderá al vector de la última etapa, h_n. Este vector, que representa el *contexto* de la secuencia de entrada, se usará como entrada de la capa *decoder*, donde se construirá un modelo del lenguaje (*language model*, LM) con el que se prediga cada palabra dada la información de la secuencia hasta ese momento de la frase que se quiere traducir, junto con la información del «contexto». Un ejemplo simplificado de esto aparece en la siguiente imagen:

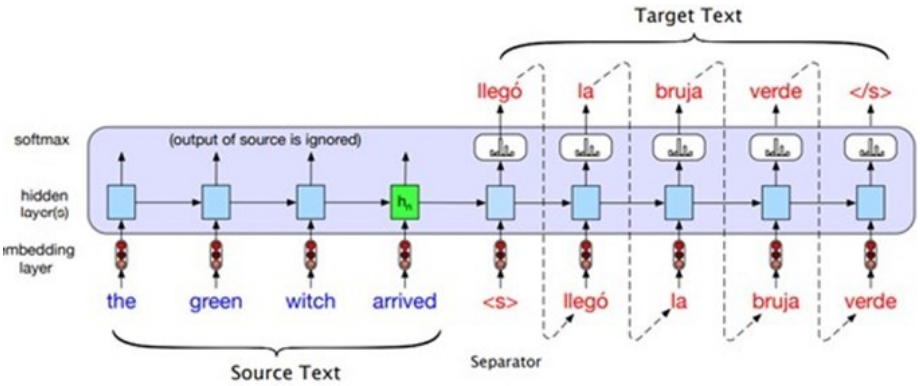

Figura 2. Arquitectura de MT basada en un esquema encoder-decoder con RNN. Fuente: Jurafsky, 2021.

Más en detalle, si la capa decoder es un LM, formalmente se expresaría de la siguiente manera:

$$p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1,y_2)...P(y_m|y_1,...,y_{m-1})$$

Donde la probabilidad de tener una palabra en concreto depende de la secuencia previa. Ahora bien, como se ha comentado, se dispone también de la información del «contexto» obtenida desde la capa del *encoder*, de manera que quedaría lo siguiente:

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)...P(y_m|y_1,...,y_{m-1},x)$$

Donde el vector *x* hace referencia a la secuencia del texto de origen. Con ello, el modelo LM predeciría la siguiente palabra dado el vector del texto original y la secuencia del texto traducido hasta ese momento. La imagen previa muestra como el token que conecta ambos modelos es un token separador <s> a partir del cual se empieza a predecir la secuencia traducida. Este modelo simplificado estaría considerando el vector del contexto como entrada directa solo en la primera etapa de la capa *encoder*. Ahora bien, esto podría hacer que la información del contexto fuese perdiendo relevancia a medida que se avanza por las etapas del modelo *decoder*. Para evitar esto, una versión más sofisticada de la arquitectura consideraría el vector del contexto como entrada de todas las etapas de la capa *decoder*, tal como muestra la siguiente imagen.

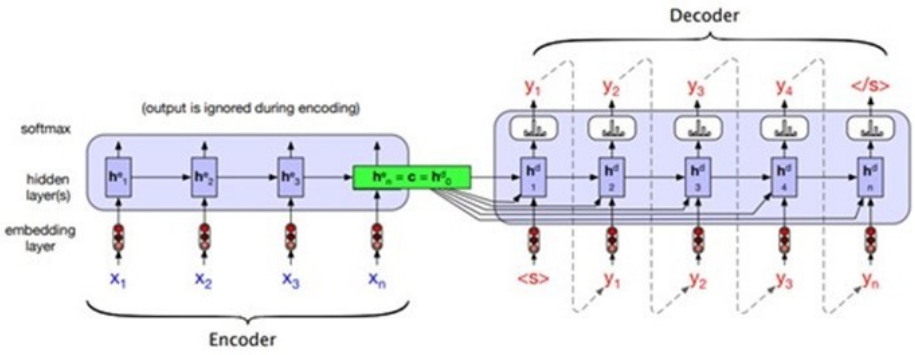

Figura 5. Arquitectura de MT donde el contexto sirve de entrada en todas las etapas del *decoder*. Fuente: Jurafsky, 2021.

Formalizado con ecuaciones, se tiene lo siguiente:

$$
\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t) \\
\hat{y}_t &= \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1})
\end{aligned}
$$

Donde los superíndices *e* y *d* hacen referencia al modelo de *encoder* o *decoder* respectivamente, *g* y *f* son funciones de activación, y cada salida del modelo *decoder* corresponde a la salida de aplicar la función *softmax,* donde la palabra que se predeciría correspondería a la que tiene mayor probabilidad ($\widehat{y}_t$).

Como se puede observar, entrenar un modelo de MT con esta arquitectura requiere de un corpus donde existan frases originales y sus correspondientes frases traducidas. El ajuste de los pesos se haría calculando la función de coste (ej., entropía cruzada) resultante de comparar las palabras que se deberían haber

predicho en cada fase del *decoder* con respecto a las palabras que ha predicho el modelo.

Ahora bien, el modelo, tal y como se ha descrito hasta ahora, estaría considerando la **palabra previa predicha** como entrada junto con el vector de salida de la capa oculta de la etapa anterior y el vector del contexto. El problema que tiene esta aproximación es que, si la predicción no es correcta, se estaría usando información que no es correcta para hacer la predicción, y el modelo tendería a irse desviando cada vez más de la frase correcta. Por este motivo se ha propuesto el uso del concepto de **teacher forcing**, con el que en la entrada de cada etapa durante el entrenamiento del modelo no corresponde a la salida predicha en la etapa previa, sino que se usa la palabra objetivo que se tendría que haber predicho. El esquema del entrenamiento con el modelo simplificado se ilustra en la siguiente imagen:

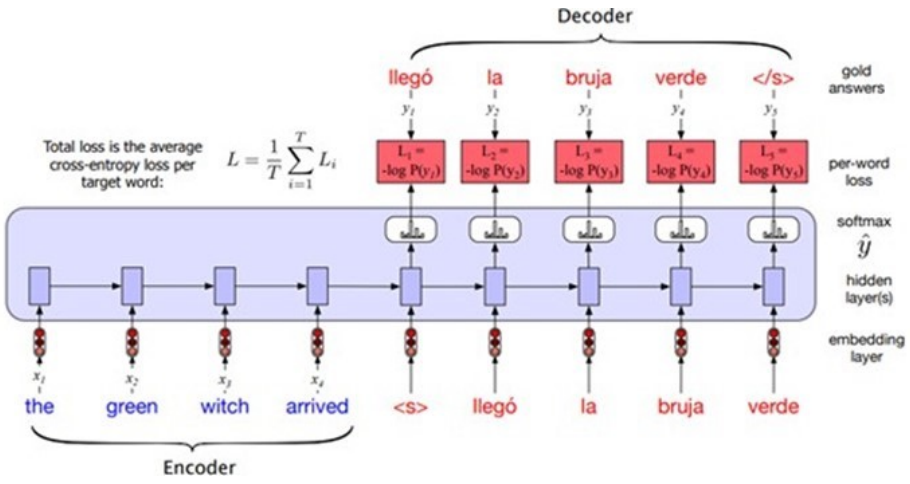

Figura 8. Esquema de entrenamiento del modelo para MT usando *teacher forcing.* Fuente: Jurafsky, 2021.

Modelo encoder-decoder con attention

Una limitación que tiene el esquema previo de *encoder-decoder* es que el *decoder* solo recibe la información de la última etapa del *encoder,* sin que se reciba directamente la información que hay en el resto de las etapas. Para solventar esta limitación, se ha propuesto el uso del mecanismo de ***attention*** dentro de estos

modelos. Con ello, en lugar de usar como contexto un vector estático resultante de la última etapa del *encoder,* se construye uno en base a la suma ponderada de los vectores de todas las etapas del *encoder.* Esto se ilustra en la siguiente imagen.

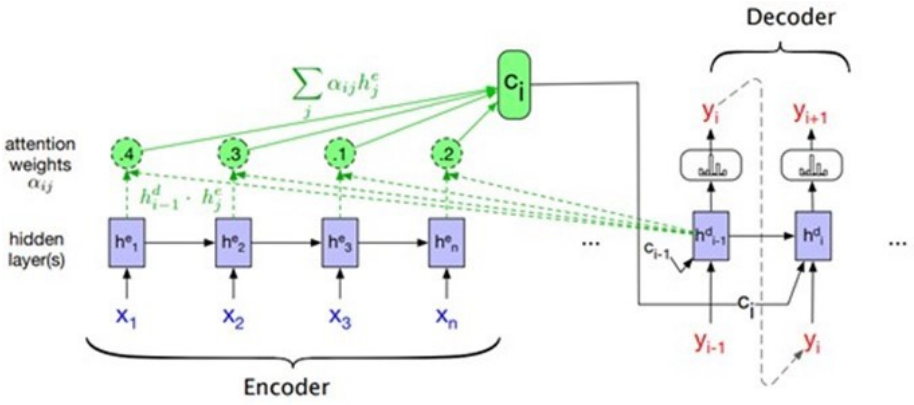

Figura 9. Esquema de *encoder-decoder* con mecanismo de *attention.* Fuente: Jurafsky, 2021.

Como se puede observar, la suma ponderada depende un parámetro α_{ij}, que para una etapa *i* del *decoder* se ajustará en función de una métrica de similitud entre el vector de la etapa *j* del *encoder* y el vector de la etapa *i-1* del *decoder*. De esta manera, en el vector del *contexto* se dará más importancia a unas etapas u otras del *encoder* en función de la etapa concreta del *decoder*. El valor de este parámetro, normalizado con una función *softmax* y para una etapa *i* del *decoder* es:

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \;\; \forall j \in e) \\ &= \frac{\exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)}{\sum_k \exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}\end{aligned}$$

De manera que el vector del *contexto* correspondiente es:

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}^e_j$$

La métrica de similitud (el *score*) se puede calcular de distintas maneras. Puede ser, por ejemplo, el producto escalar entre el vector de cada una de las etapas del *encoder* y la etapa correspondiente del *decoder:*

$$score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) = \mathbf{h}_{i-1}^{d} \cdot \mathbf{h}_{j}^{e}$$

El valor del *score* con este producto escalar también se puede ajustar usando una matriz de pesos propia,

W_s , de manera que quedaría:

$$score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}) = \mathbf{h}_{t-1}^{d} \mathbf{W}_{s} \mathbf{h}_{j}^{e}$$

Modelo encoder-decoder con transformers

La arquitectura de un modelo de MT también se puede construir con *transformers,* donde se mantiene el esquema *encoder-decoder.* El *encoder* se implementa mediante modelos de *multiheaded attention* bidireccionales, con los que se construye el vector del contexto. Por otro lado, el *decoder* se implementa con un esquema *multiheaded attention* como un LM causal. La siguiente imagen ilustra el esquema del modelo:

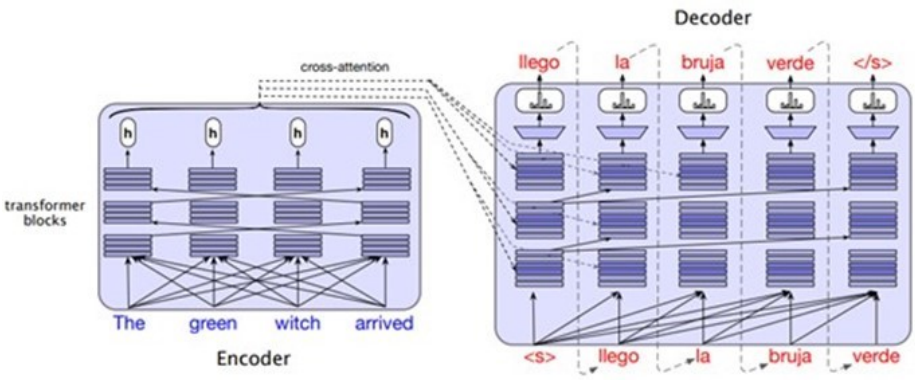

Figura 14. Modelo *encoder-decoder* con *Transformers*. Fuente: Jurafsky, 2021.

Cada bloque de *transformers* del modelo de *decoder* (tres bloques en la arquitectura anterior) va a estar compuesto de las siguientes capas:

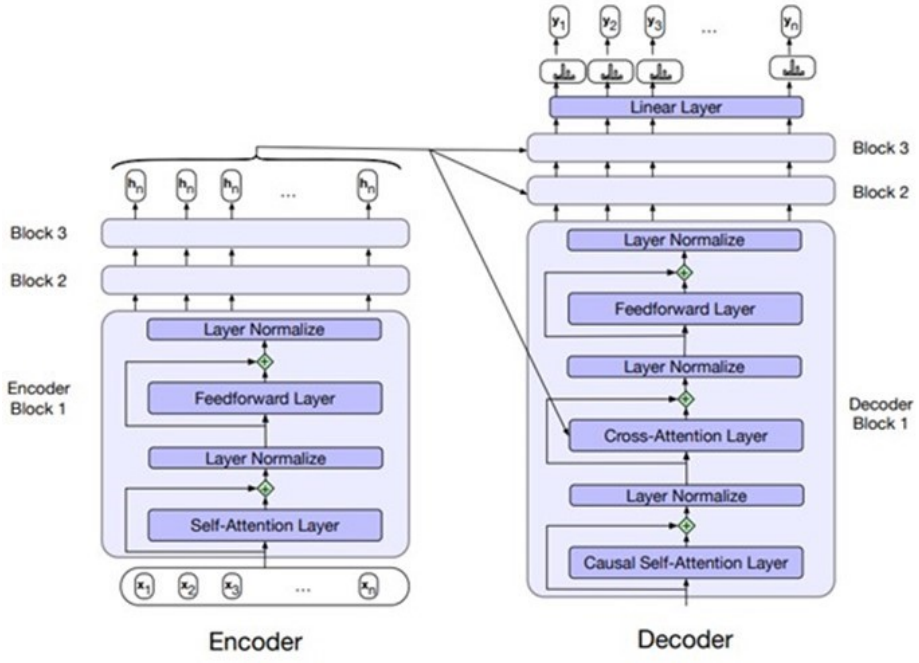

Figura 15. Esquema en detalle de un bloque de *transformers* del modelo *decoder*. Fuente: Jurafsky, 2021.

Como se puede observar, hay **dos tipos de modelos de *attention*.** Por un lado, aparece la **capa causal**, donde se sigue el esquema habitual de modelo *self-attention* en el que la entrada se basa en la secuencia recorrida hasta ese momento. Ahora bien, también aparece una **capa *cross-attention***, donde el valor **Q** (*query*) corresponde a la información de salida de la capa previa, pero **K** (*key*) y **V** (*value*) corresponden al **vector de contexto** obtenido del *encoder*. De manera que, si W^Q,W^K yW^V son las matrices de pesos para las entradas Q, K y V respectivamente, la salida de esta capa (normalizada y tras aplicar la función *softmax*) sería:

$$\mathbf{Q} = \mathbf{W}^{\mathbf{Q}}\mathbf{H}^{dec[i-1]};\ \mathbf{K} = \mathbf{W}^{\mathbf{K}}\mathbf{H}^{enc};\ \mathbf{V} = \mathbf{W}^{\mathbf{V}}\mathbf{H}^{enc}$$

$$CrossAttention(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\intercal}}{\sqrt{d_k}}\right)\mathbf{V}$$

Con ello, la capa de *cross attention*, permite que el modelo *decoder* reciba como información para la traducción la información asociada al modelo *encoder* sobre el texto original.

De cara al entrenamiento del modelo, se suele aplicar el concepto de ***teacher forcing***, igual que se hizo con las RNN.

Evaluación de un modelo de MT

Desde el punto de vista de la evaluación de los modelos de MT se analizan dos aspectos. Por un lado, la **adecuación** o **fidelidad**, que mide cómo de bien se representa la frase traducida de referencia con la traducción obtenida automáticamente. Por otro, la **fluencia** que mide si la traducción es clara, legible, natural, gramáticamente correcta.

La fidelidad se puede medir con métricas como ***character F-score*** (chrF), que compara el solape en n-gramas que hay entre la frase traducida con el modelo de MT y la traducción original. El cálculo es similar a la métrica de F-score de los modelos de ML: se calcula el porcentaje de n-gramas de la traducción de referencia que aparecen en la frase traducida automáticamente (hipótesis), y viceversa. Esto se hace todos los n-gramas anteriores (es decir, se calcula para bigramas y para unigramas). Posteriormente, se hace la media de estos valores, y se calcula chrP (que tiene el valor calculado de la traducción automática frente a la referencia) y chpR (que tiene el valor calculado de la referencia frente a la traducción automática). Especificando un parámetro

β , el cálculo de chrF sería:

$$\text{chrF}\beta = (1+\beta^2)\frac{\text{chrP}\cdot\text{chrR}}{\beta^2\cdot\text{chrP}+\text{chrR}}$$

La **métrica chrF** es **útil** especialmente **para comparar dos modelos de traducción frente a un mismo conjunto de datos,** más que como métrica para analizar un modelo de manera aislada o para comparar dos modelos entrenados en conjuntos de datos de distintas lenguas.

Existen otro tipo de métricas con las que, en lugar de comparar las traducciones obtenidas con la referencia a nivel de cadenas de caracteres, se comparan a nivel de *embeddings.* Esto ofrece mayor flexibilidad en las comparaciones, ya que una traducción puede ser correcta a pesar de no seguir exactamente el orden de los elementos de la frase de referencia. Esta aproximación también está bien para comparar traducciones donde puede que se hayan obtenido sinónimos de palabras, pero que también serían válidos como traducción. Un ejemplo es la métrica **BERTScore,** con la que se obtienen los *embeddings* de las palabras de la traducción de referencia

$\widetilde{x_j}$ y las de la traducción obtenida

x_i y se comparan ambos vectores, de manera que se pueden calcular métricas de precisión y *recall* de la siguiente manera:

$$R_{\text{BERT}} = \frac{1}{|x|}\sum_{x_i\in x}\max_{\tilde{x}_j\in\tilde{x}} x_i\cdot\tilde{x}_j \qquad P_{\text{BERT}} = \frac{1}{|\tilde{x}|}\sum_{\tilde{x}_j\in\tilde{x}}\max_{x_i\in x} x_i\cdot\tilde{x}_j$$

Sesgos en los modelos

Un aspecto importante a tener en cuenta en los modelos de PLN en general, y en los de MT en particular, es el tema de los sesgos. Como cualquier modelo de aprendizaje automático, un modelo de MT aprende a traducir en base a un conjunto

de datos de origen que puede contener distintos tipos de sesgos que se propagarán al modelo y a sus predicciones. Por ejemplo, si se tiene un modelo de MT con el que se quieren traducir estos textos:

Hungarian (gender neutral) source	English MT output
ő egy ápoló	she is a nurse
ő egy tudós	he is a scientist
ő egy mérnök	he is an engineer
ő egy pék	he is a baker
ő egy tanár	she is a teacher
ő egy vesküvőszervező	she is a wedding organizer
ő egy vezérigazgató	he is a CEO

Figura 19. Ejemplo de traducciones sesgadas entre textos con género neutro (en húngaro) y textos con el género especificado (inglés). Fuente: Jurafsky, 2021.

Se puede observar cómo partiendo de un texto de género neutro, se tiene una tendencia a asignar un género concreto a algunas profesiones y otro a otras, cuando esta especificación del género no aparece en las frases de origen que se quieren traducir. Este es un claro ejemplo de **sesgos de género** en los modelos: debido a que en los textos de origen no hay un equilibrio entre las referencias a determinadas profesiones y las referencias a determinados géneros (ej., hay muchos más textos que hablan de *engineers* junto con *he* que con *she)*, el modelo aprende en base a este sesgo y lo usa en sus predicciones.

De cara a tratar con este problema de sesgos en los modelos de PLN y MT, existen distintas soluciones, desde la detección a nivel de datos de origen, a nivel del entrenamiento del modelo, o a nivel de sus predicciones, hasta distintas maneras de mitigarlo. Esto se enmarca dentro del ámbito de la **Inteligencia Artificial Responsable** (Responsible Artificial Intelligence, RAI), concretamente dentro del ámbito de las técnicas de *Fairness.*

Ideas clave

Para ampliar el tema de los sesgos en el ámbito de PLN se puede completar el estudio con el Capítulo 10 del libro, pg. 25: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall. https://web.stanford.edu/~jurafsky/slp3/10.pdf

9.3. Autocompletado y generación automática de resúmenes

Los bloques de *transformers* se pueden usar también para construir modelos de **autocompletado de texto.** Siguiendo con la arquitectura de un *transformer* para la construcción de un LM causal, se entrena un modelo que trate de predecir, dada una secuencia de N tokens, los M tokens siguientes. Esto se refleja en la siguiente imagen:

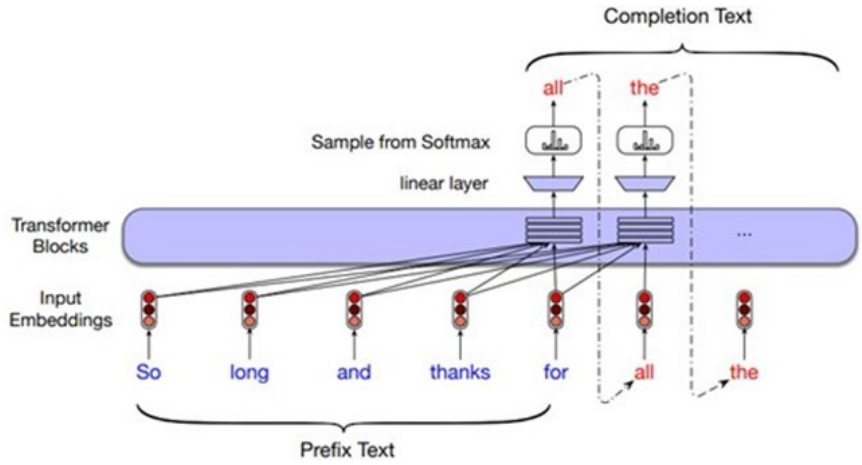

Figura 20. Arquitectura para un modelo de autocompletado de texto basado en *transformers.* Fuente: Jurafsky, 2021.

Como se puede observar, dada la secuencia:

- *So long and thanks for.*

El modelo va prediciendo uno a uno los siguientes tokens, generando así el texto autocompletado. Una vez que genera el primer token, este junto con la secuencia de entrada, se usan para predecir el siguiente token, y así sucesivamente.

Otro ejemplo de uso del bloque de *transformers* es para la construcción de modelos para tareas de **generación automática de resúmenes**, usando una arquitectura

similar a la descrita previamente. Suponiendo que se tienen pares de textos originales y sus correspondientes resúmenes, tal que

$(x_1,\ldots,x_m)$ es la secuencia de texto original, y

$(y_1,\ldots,y_n)$ el resumen correspondiente, se pueden concatenar ambas secuencias, usando un token que delimite una de la otra. La secuencia final sería, por tanto:

$$(x_1,\ldots,x_m,\delta,y_1,\ldots,y_n)$$

Con

δ el token que delimita una de la otra, *m* el tamaño de la secuencia del texto original, *n* la del texto resumido, y *n + m + 1* la secuencia usada en el modelo. Con ello, usando un LM causal con *transformers*, se tratarán de predecir ambas secuencias como una única. Una vez entrenado un modelo con los datos de entrenamiento, se podrá usar para, dado un texto de origen, obtener la secuencia que le seguiría tras el token final

δ, que correspondería a la secuencia del texto resumen. Estos modelos se suelen entrenar con técnicas de ***teacher forcing***. Una ilustración de la arquitectura aparece en la siguiente imagen:

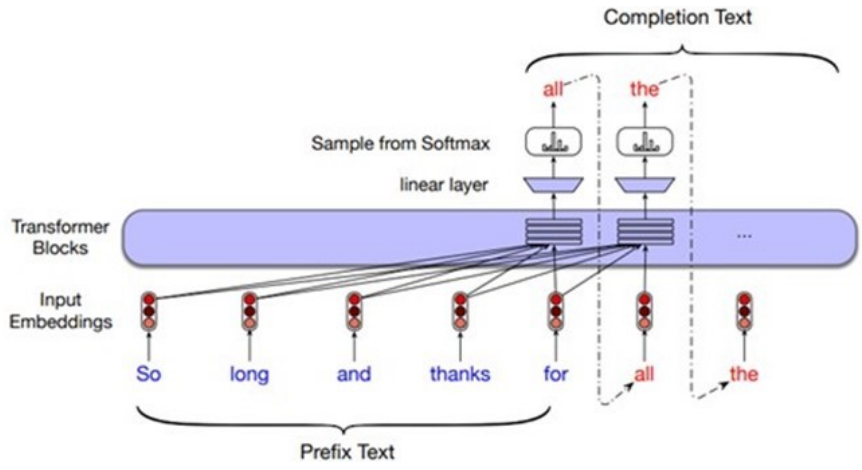

Figura 20. Arquitectura para un modelo de resumen de textos basado en *transformers*. Fuente: Jurafsky, 2021.

9.4. Análisis de sentimientos

El análisis de sentimientos es una tarea dentro del ámbito de PLN con el que se busca **detectar el sentimiento expresado por el autor** de un texto **hacia un determinado objeto,** pudiendo ser, por ejemplo, un sentimiento negativo o positivo.

Así, el objetivo de la clasificación de sentimientos a nivel computacional es usar distintos algoritmos que permitan hacer este análisis de manera automática.

Por ejemplo, dadas las siguientes frases, el objetivo sería detectar que en la primera el sentimiento es negativo, y en la segunda, positivo:

- Este producto es terrible. No cumple con las expectativas y lo he tenido que devolver.
- Magnífica compra y muy buena atención por parte del vendedor. El producto me ha encantado.

Una manera de hacer esa detección automática es **mediante el uso de algoritmos supervisados de aprendizaje automático.** Para ello, es necesario partir de un corpus en el que se tengan ejemplos de frases etiquetadas con su correspondiente sentimiento (por ejemplo, positivo o negativo), y se trate de predecir ese sentimiento en base a rasgos *(features)* obtenidas de cada uno de los textos. Existen distintas maneras de obtener esos rasgos, como por ejemplo con las técnicas de representación vectorial de bolsas de palabras (BoW). De esta manera, se transformarían los textos de entrada en una matriz donde cada fila representa uno de los textos y cada uno de los rasgos correspondería a una de las palabras del vocabulario, como se muestra en el siguiente ejemplo:

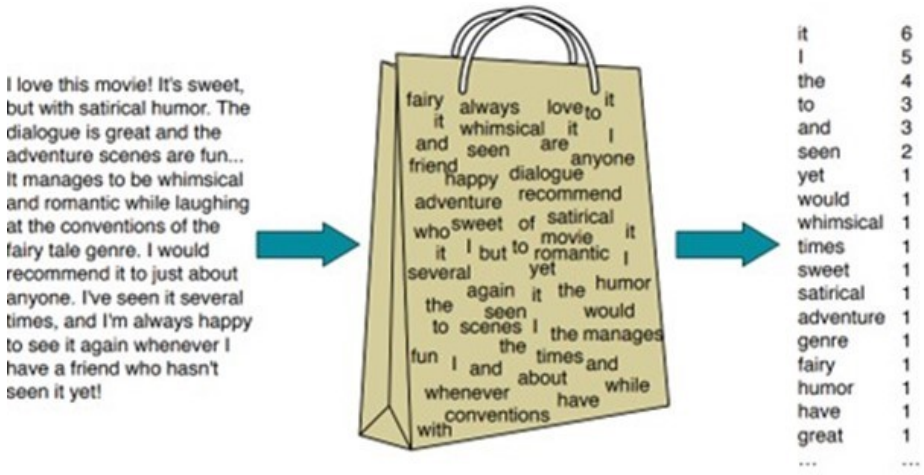

Figura 21. Ejemplo de BoW sobre un texto. Fuente: Jurafsky, 2021.

Teniendo esta matriz, y teniendo las clases (los sentimientos) que se quieren predecir, se pueden usar distintos algoritmos de aprendizaje supervisado para predecir el sentimiento asociado sobre cualquier texto de entrada.

El modelado de los textos de entrada y la construcción de los rasgos asociados no sólo se tiene por qué realizar con técnicas como BoW, sino que también se puede realizar con técnicas basadas en *word embeddings*.

Además de esta aproximación, que exige disponer de un corpus con textos etiquetados con su correspondiente sentimiento, existen otras posibilidades basadas e n **lexicones**, donde la **información de los sentimientos** se tiene a **nivel de palabras individuales.** Partiendo de algún lexicón disponible, se trata de inferir el sentimiento asociado a todo el texto desde el sentimiento que tienen sus palabras individuales, apoyándose en el principio de composición.

Una opción simple de aplicar esto es mediante el uso de **reglas de clasificación** definidas a priori, que asocian como sentimiento principal de un texto aquél que aparezca más veces entre las palabras que lo componen. Así, se puede calcular una métrica de *sentimiento positivo*

f^+ contando las palabras *w* del texto que tengan sentimiento positivo

θ_w^+ , y lo mismo para el *sentimiento negativo*

f^- con las palabras de sentimiento negativo

θ_w^- . Con ello, obteniendo los cocientes

$\frac{f^+}{f^-}$ y

$\frac{f^-}{f^+}$, comparándolos con un valor umbral

λ , se decide si clasificar ese texto como positivo o negativo.

$$f^+ = \sum_{w \text{ s.t. } w \in positivelexicon} \theta_w^+ count(w)$$

$$f^- = \sum_{w \text{ s.t. } w \in negativelexicon} \theta_w^- count(w)$$

$$sentiment = \begin{cases} + & \text{if } \frac{f^+}{f^-} > \lambda \\ - & \text{if } \frac{f^-}{f^+} > \lambda \\ 0 & \text{otherwise.} \end{cases}$$

La representación de un texto en base al sentimiento de las palabras individuales que lo componen (obtenido desde un lexicón), **también sirve como rasgos adicionales** de un modelo de aprendizaje supervisado para el caso en el que se disponga de un corpus de entrada con sentimientos principales anotados en textos.

Ideas clave

Así, además de los rasgos obtenidos en base a técnicas como BoW o *word embeddings,* se pueden incorporar rasgos adicionales basados en la información que proporcionan esos lexicones.

Puedes completar el estudio sobre la clasificación de sentimientos y el uso de lexicones con el Capítulo 20 del libro: Jurafsky, D. y Martin, J. H. (2021*). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall. https://web.stanford.edu/~jurafsky/slp3/20.pdf

---

9.5. Question Answering

Un sistema de *Question Answering* (QA) tiene como **objetivo poder responder a consultas o preguntas que formule un usuario.** Estos sistemas forman parte de los buscadores de información, asistentes virtuale de manera que un usuario realice una consulta y el sistema pueda proporcionar una respuesta a la misma.

La mayoría de los sistemas de QA tratan de responder a un tipo de preguntas concretas: las preguntas sobre hechos *(factoid questions),* que hacen referencia a consultas sobre datos o hechos particulares, y que se pueden responder de manera directa.

Un ejemplo de preguntas de este tipo sería consultas como «¿Dónde se encuentra el Coliseo?» o «¿A qué edad se puede conducir legalmente en USA?». Este tipo de preguntas es el caso **contrario a las preguntas abiertas,** donde no se puede responder con una única respuesta (como un dato concreto o un hecho particular). Un ejemplo de preguntas de este segundo tipo sería «¿Cómo aprender a tocar el piano?»

Dentro de los paradigmas que se pueden utilizar para construir un sistema de QA existen diversas opciones, como se verá a continuación.

Information retrieval

Information Retrieval (IR) es un campo que incluye distintas técnicas de recuperación de información, y donde un caso concreto es el de ***ad hoc retrieval,*** donde se reciben consultas de usuarios (como textos de entrada) y se busca responder a ellas utilizando un corpus de datos (ej., una colección de documentos). Un ejemplo de esta arquitectura aparece en la siguiente imagen.

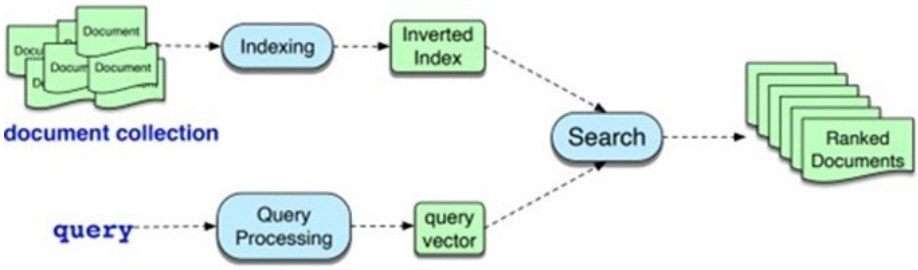

Figura 23. Esquema de un sistema de IR. Fuente: Jurafsky, 2021.

Como se puede observar, se recibe una **consulta *(query)*** de un usuario, y se expresa mediante un vector (con alguna de las técnicas de «representación vectorial» ya vistas). En el ejemplo básico de un IR, se usarían técnicas basadas en bolsas de palabras (BoW, *bag-of-words)* para la representación de la *query* de entrada y de los documentos de la colección, para usar después una métrica de similitud vectorial y encontrar el documento más cercano a la *query.* Usando una representación basada en TF-IDF, la métrica de similitud entre una *query* **q** y un documento **d** se puede aproximar como:

$$\text{score}(q,d) = \sum_{t \in q} \frac{\text{tf-idf}(t,d)}{|d|}$$

Donde, dado que la *query* tendrá menos términos que un documento, sólo se consideran los términos del documento que aparecen en la *query,* normalizando el resultado en función de los términos que tenga el documento. La arquitectura anterior contiene un elemento adicional, el **bloque *inverted index,*** con el que se hace que la búsqueda de documentos sea más eficiente. En lugar de consultar siempre toda la colección para una *query* de entrada, se usa un diccionario intermedio que tiene como claves los distintos términos, y asociado a ellas los documentos que tienen esos términos, junto con información adicional como la frecuencia de los términos en esos documentos. De esta manera, frente a esa *query* se seleccionan sólo los documentos relevantes y se calcula la similitud respecto a ellos.

IR con vectores densos

El esquema previo de QA usando técnicas basadas en BoW tiene como problema que solo se recuperarán documentos que tengan palabras que coincidan exactamente con la *query* del usuario. Ahora bien, no se considerarían otras circunstancias como, por ejemplo, cuando se usen sinónimos de palabras.

Una manera de hacer esto es mediante el uso de *embeddings* obtenidos desde modelos preentrenados, como BERT. BERT proporcionaría el vector de *embedding* para una palabra concreta. De cara a representar todo el texto como un único vector, se pueden usar técnicas de *pooling,* de manera que se combinen los valores de los vectores de cada palabra en un único vector que represente el texto. Así, representando con un vector la *query* y con otro el documento, se puede obtener el valor de la similitud entre ellos.

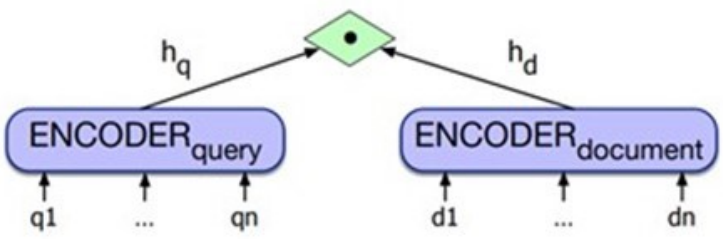

Figura 25. IR mediante el uso de embedddings. Fuente: Jurafsky, 2021.

IR para factoid QA

Los sistemas de IR se pueden usar para no solo encontrar documentos relevantes para una consulta, sino también para **poder responder a consultas** que consisten en preguntas específicas usando la información disponible en la colección de documentos. Estos sistemas de QA basados en IR (denominados IR-based QA) se pueden diseñar con una aproximación de recuperación y lectura, tal y como indica la siguiente imagen.

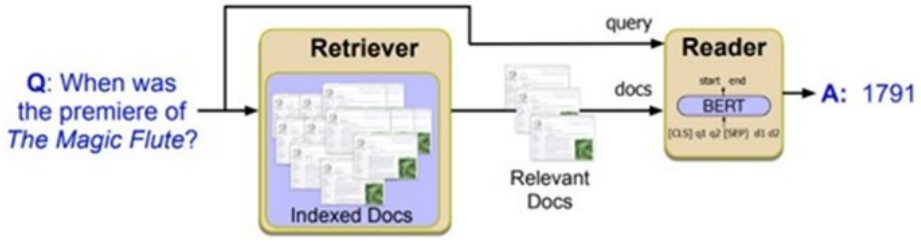

Figura 26. Sistema de IR para *factoid questions*. Fuente: Jurafsky, 2021.

Esta aproximación tiene **dos fases.** En la primera, se **recuperan documentos de la colección** que son relevantes en relación con la consulta de entrada. En la segunda, se lleva a cabo un proceso de **comprensión del texto mediante algoritmos** (basados, por ejemplo, en redes neuronales) con los que se trata de encontrar el fragmento de texto concreto que respondería a la consulta del usuario. Así, esta fase de comprensión mediante la detección de fragmentos (*spans*) de texto que respondan a la pregunta de entrada pasa por identificar una subcadena de caracteres que sirva como respuesta.

Formalmente, la tarea sería: dada una consulta *q* de *n* tokens

$q_1, q_2, \ldots, q_n$, y dado un documento *p* de *m* tokens

$p_1, p_2, \ldots, p_m$, calcular la probabilidad de que la cadena de texto *a* sea la respuesta,

$P(a|q,p)$. Este cálculo de probabilidad se puede aproximar considerando sólo las posiciones iniciales y finales de la cadena *a*, tal que:

$$P(a|q,p) = P_{start}(a_s|q,p)\, P_{end}(a_e|q,p)$$

De manera que para cada token

p_i del documento *p*, se obtengan dos probabilidades,

$p_{start}(i)$ y

$p_{end}(i)$ que representen la probabilidad de que ese token sea el principio o fin de la respuesta, respectivamente. La imagen siguiente muestra la arquitectura de un modelo para esta tarea.

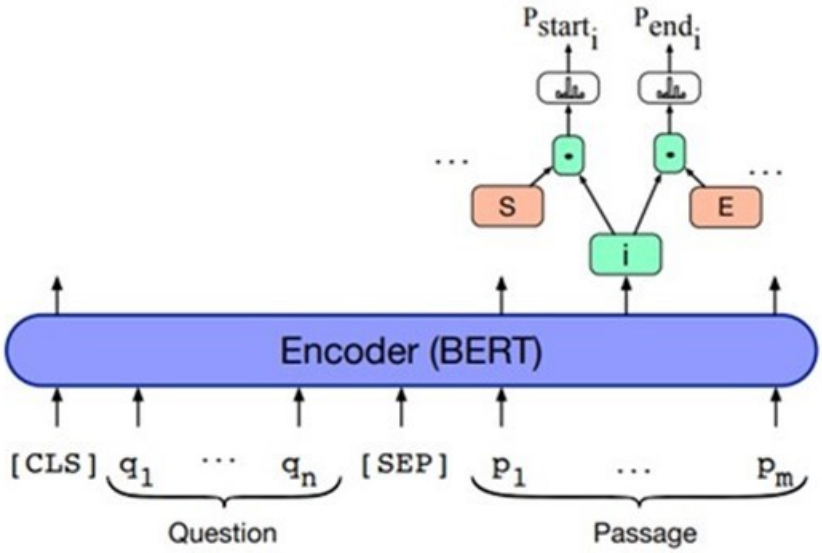

Figura 27. Arquitectura de un modelo para QA basado en fragmentos de documentos. Fuente: Jurafsky, 2021.

Partiendo de los tokens de la consulta de entrada y de un documento en cuestión, se concatenan usando un token genérico [SEP], sirviendo como entrada de un modelo de *encoding*, como por ejemplo uno basado en la arquitectura de BERT. Con ello, se obtiene primero un vector de salida

p_i' para cada etapa *i* correspondiente a un token de entrada

p_i del documento. Este vector de salida se usa en un producto escalar con un vector de *embedding* S para obtener la probabilidad de que ese *token*

p_i del documento correspondiese al inicio de la subsecuencia de respuesta. Análogamente, se multiplicaría con un vector E para obtener la probabilidad de que correspondiese al fin de la secuencia. Los valores de los vectores de *embeddings* S y E se obtendrían durante el entrenamiento (*fine tuning*) del modelo sobre el conjunto de datos usado en el entrenamiento. De esta manera, normalizando con una función *softmax,* el cálculo de estas probabilidades sería:

$$P_{\text{start}_i} = \frac{\exp(S \cdot p'_i)}{\sum_j \exp(S \cdot p'_j)}$$

$$P_{\text{end}_i} = \frac{\exp(E \cdot p'_i)}{\sum_j \exp(E \cdot p'_j)}$$

El entrenamiento del modelo se haría usando conjuntos de datos con pares de preguntas-documentos donde está indicado el inicio y el fin de la respuesta. Con ello, la subsecuencia candidata como respuesta a la consulta sería la que va del token *k* al token *j*, con

$j \geq k$,

P'_{start_k} la probabilidad máxima de inicio (que corresponde al token *k*), y

P'_{end_j} la probabilidad máxima de fin (que corresponde al token *j*). Comparando los tokens de inicio-fin de la secuencia propuesta como respuesta con los tokens de inicio y fin reales del conjunto de datos de entrenamiento, se obtendría **la métrica de error.**

Además de esto, el modelo tiene que ser capaz de predecir también los casos en los que la respuesta a **la pregunta no se pueda responder con ningún documento**. Una manera de hacer esto es tratando esos casos como aquellos en los que la respuesta correspondiese al token [CLS] (que sería tanto el inicio como el fin de la secuencia).

Finalmente, una arquitectura como la de BERT tiene un **número limitado de tokens de entrada** (512 como máximo para la arquitectura estándar). El problema es que los tokens de una pregunta-documento pueden superar este valor. Para tratar estas casuísticas, se recorre el documento con ventanas de tokens que correspondan al máximo permitido (descontando el número de tokens de la consulta) y se predice si la respuesta pudiera estar en esa ventana o no. Posteriormente, si hay varias respuestas posibles correspondientes a distintas ventanas con una probabilidad superior al caso por defecto de que la respuesta sea [CLS] (es decir, que no hay respuesta), se elige de entre ellas la de mayor probabilidad.

Evaluación de las respuestas

De cara a evaluar si el sistema funciona correctamente y es capaz de responder a una pregunta con la respuesta adecuada, existen métricas como ***mean reciprocal rank*** (MRR). Con MRR, se compara la lista de respuestas posibles ordenadas por probabilidad con respecto a la pregunta de referencia de los datos de evaluación. Si, por ejemplo, la respuesta correcta aparece en la tercera posición, el valor RR (*reciprocal rank*) sería 1/3. Con ello, para un conjunto de evaluación de Q preguntas, la métrica final sería la media de los distintos RR, como expresa la siguiente ecuación:

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

Puedes consultar otros sistemas de QA basados tanto en aprendizaje neuronal como en técnicas clásicas en el Capítulo 23 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall. https://web.stanford.edu/~jurafsky/slp3/23.pdf

9.6. Reconocimiento automático del habla y text-to-speech

Dentro de las aplicaciones industriales de PLN aparece el **Reconocimiento Automático del Habla** (Automatic Speech Recognition, ASR), con el que se busca **transformar a texto el audio generado por el habla de un usuario.** Esto es importante para muchas aplicaciones basadas en PLN, como por ejemplo los asistentes virtuales, donde un usuario puede comunicarse libremente con el sistema para realizar preguntas o pedirle a este que realice determinadas tareas.

Así, con el ASR se busca poder tener una interfaz de comunicación con un usuario, de manera que este se pueda expresar con el habla, y esa información se convierta en texto para poder ser procesada computacionalmente.

Ahora bien, esta tarea es compleja, y aparecen varios aspectos a considerar.

Por un lado, es importante el **aspecto relacionado con el tamaño del vocabulario usado por los usuarios.** La complejidad de la tarea de ASR no es la misma si, por ejemplo, se espera que el usuario conteste con un «Sí» o «No», frente a un caso en el que el usuario puede comunicarse con el sistema de manera abierta.

En segundo lugar, la **complejidad** dependerá de **hacia quién se esté comunicando el usuario o la manera en la que este lo está haciendo.** Por ejemplo, no es lo mismo la claridad de la comunicación cuando el usuario se dirige a un sistema (donde las órdenes o cuestiones que formula son relativamente claras o directas), frente a la claridad que puede tener la comunicación de un usuario cuando este se encuentra conversando con otra persona. La tarea de ASR será menos compleja en ese primer caso que en el segundo.

En tercer lugar, **influye el canal de comunicación y el ruido.** La complejidad del tratamiento de la información se verá afectada en función de aspectos como el lugar donde se esté comunicando el usuario con el sistema, siendo, por ejemplo, más compleja cuando se está en un lugar ruidoso (como una calle) frente a otros lugares más tranquilos (como una sala cerrada sin más usuarios).

Finalmente, **influyen** también aspectos **como el acento u otras características concretas de cada usuario particular.** No serán igual de precisos los resultados del ASR cuando se reciban mensajes en inglés con un determinado tipo de acento si el sistema sólo se ha entrenado sobre datos de un acento en concreto.

Además de las tareas de ASR, existe también el **proceso inverso:** partiendo de un texto escrito, poder **generar automáticamente un mensaje de audio asociado.** Esta tarea se conoce como **text-to-speech** (TTS).

Así, la tarea de TTS es el proceso inverso al ASR, como se indica en la siguiente imagen. En ella se indica como la representación del mensaje es o bien en modo texto, o bien en modo audio, caracterizada por una onda acústica.

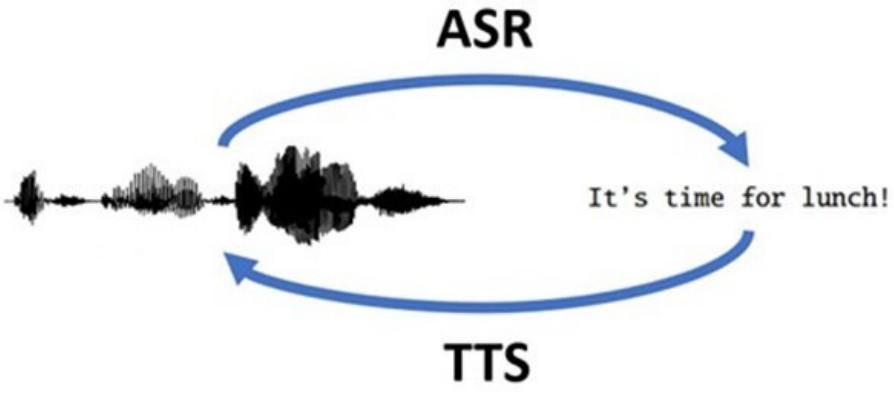

Figura 31. Tareas de reconocimiento del habla: ASR para pasar de voz a texto, y TTS para pasar de texto a voz. Fuente:

Ideas clave

Extracción de rasgos para ASR

Para poder pasar de una onda acústica al texto correspondiente mediante el uso de técnicas de ASR, el **primer paso** es **identificar una serie de variables** o rasgos que representen la información fundamental de dicha onda, algo análogo al tratamiento de otras fuentes de datos cuando van a ser utilizadas en modelos de aprendizaje automático.

A nivel intuitivo, la idea es, partiendo de la onda, tener un vector de variables que pueda servir de entrada de un modelo de aprendizaje automático. Esto se lleva a cabo mediante distintas fases relacionadas con el procesamiento de señales: muestrear la señal para tener distintas muestras discretas *(sampling)*, recoger la información de la amplitud de onda con el proceso de cuantificación *(quantify)*, recorrer estos datos mediante ventanas suficientemente pequeñas para que el comportamiento de la señal sea estacionario dentro de ella, y aplicar distintas transformaciones para convertir estos datos en un vector de variables.

Puedes consultar más información sobre aspectos relacionados con el tratamiento de la señal y sobre la fonética en los Capítulos 25 y 26 del libro*: Jurafsky, D. y Martin, J. H. (2021). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall. https://web.stanford.edu/~jurafsky/slp3

Arquitectura de un sistema de ASR

Una de las arquitecturas más habituales para ASR usando redes neuronales se basa en el uso de *encoders-decoders,* de modo similar a cómo ocurría con los modelos de MT. Partiendo de la señal acústica, se extraen los rasgos que sirven de entrada al modelo. La salida del modelo será o bien una predicción a nivel de palabras (qué palabras aparecen en función de la secuencia de entrada) o a nivel de caracteres o letras. El siguiente ejemplo muestra una arquitectura en la que la salida sirve para predecir el texto asociado a la señal de entrada a nivel de letras.

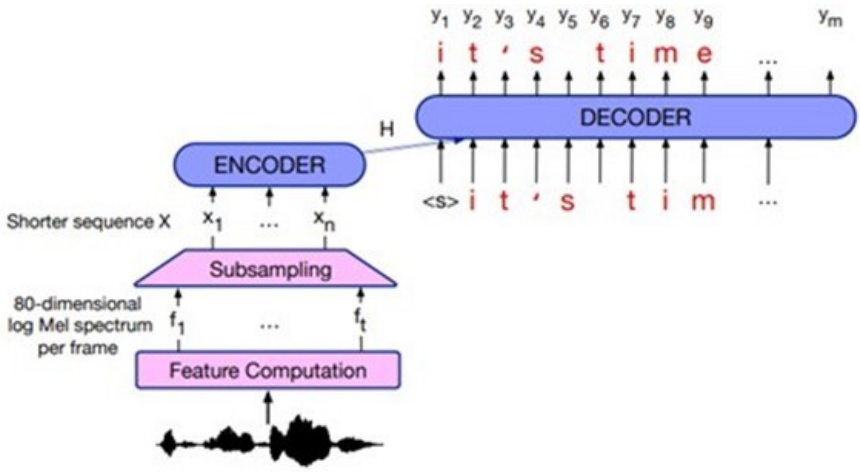

Figura 32. Ejemplo de arquitectura *encoder-decoder* para un sistema de ASR. Fuente: Jurafsky, 2021.

Si se trata de predecir a nivel de letras, se considera entonces un problema de clasificación en el que los posibles valores de salida corresponden a las distintas letras o caracteres de un alfabeto. Por ejemplo, los valores posibles de y serían (incluyendo letras, números y caracteres especiales):

$$y_i \in \{a, b, c, ..., z, 0, ..., 9, \langle \text{space} \rangle, \langle \text{comma} \rangle, \langle \text{period} \rangle, \langle \text{apostrophe} \rangle, \langle \text{unk} \rangle\}$$

Ideas clave

Una consideración para tener en cuenta es que, aunque los *modelos encoding-decoding* son útiles cuando el número de datos de entrada no coincide con el de salida, el caso de ASR es especialmente acentuado en este aspecto, ya que una salida de caracteres asociados a una palabra (que podrían ser, por ejemplo, cinco para una palabra como perro), está asociada a una gran cantidad de variables de entrada. El motivo es que la señal acústica que asociada a esa palabra sería del orden de segundos (por ejemplo, 2 seg.), y las variables que se generan desde esa secuencia vendrían de hacer una partición en múltiples segmentos de mucha menor duración.

Por este motivo, **se añade un paso adicional** antes de llegar al modelo de *encoding* con el que comprime la información de entrada para que el vector de variables sea más reducido. Hecho esto, el resto del modelo funciona como un modelo estándar de *econding-decoding* como los ya vistos, donde se usa habitualmente la aproximación de *teacher forcing* en el entrenamiento.

Una consideración particular es que el problema de ASR es a nivel general un problema de modelado causal, donde solo está disponible la información de la secuencia previa para hacer las predicciones.

Por este motivo, una aproximación considerada es usar también la información de modelos de lenguaje entrenados para predecir la posible secuencia que continuaría a la ya detectada, y combinar ambos modelos en uno, usando una única función de coste.

Evaluación de un sistema de ASR

Una métrica de evaluación habitual para los ASR es **la tasa de error por palabra** *(word error rate)*. Con esta métrica se analiza cuanto difiere la secuencia predicha de la secuencia real. En concreto, se contabiliza cuántas palabras adicionales hay en la secuencia predicha que no están en la real *(insertions)*, cuantas palabras hay en la secuencia real que no aparecen en la predicha *(deletions)*, y cuántas palabras hay en la secuencia predicha que no coinciden con las de la original *(substitutions)*. La fórmula que contabiliza todos estos aspectos sería:

$$\text{Word Error Rate} = 100 \times \frac{\text{Insertions} + \text{Substitutions} + \text{Deletions}}{\text{Total Words in Correct Transcript}}$$

Arquitectura de un sistema de TTS

Con un sistema de TTS **se parte de una cadena de texto,** y se **busca generar la señal acústica.** Esto es importante en distintas aplicaciones, como los agentes conversacionales, donde el sistema interactúa con un usuario y le habla con voz.

Como ocurre con otras aplicaciones de PLN, los sistemas de TTS también se pueden construir con un esquema *encoder-decoder,* donde la entrada es una cadena de texto y la salida la señal acústica.

Ahora bien, hay una diferencia en la aproximación de un TTS frente a un ASR más allá de que un proceso sea el inverso del otro. En el **caso del ASR,** el **sistema debería ser independiente del hablante.** Es decir, que se debe entrenar con audios de distintos hablantes para ser agnóstico al usuario concreto que se comunique con el sistema. En el **caso del TTS** no es relevante que el sistema sea entrenado con datos de distintos hablantes, ya que lo que **se busca** es **que se genere un tipo de voz fija.**

Por este motivo los TTS son dependientes del hablante, y se necesitan menos datos para entrenarlos que en el caso de los ASR.

Un sistema de TTS tiene **dos etapas de predicción**. En primer lugar, se parte de una **cadena de texto** para predecir el **espectrograma asociado a ella** (en concreto, una aproximación común es mapear letras al espectrograma de Mel). En segundo lugar, los elementos del espectrograma se mapean a la señal acústica asociada con un proceso denominado **vocoding.**

En la arquitectura de un TTS hay un paso adicional a considerar que sirve para normalizar los caracteres especiales que pueda haber en el texto de entrada. Cuando aparecen números, caracteres como $, signos de puntuación… se tienen que convertir a palabras para poder generar el audio correspondiente (ej., $ a «dólares»). Ahora bien, en algunos casos esto dependerá del contexto de la frase. Por ejemplo, un mismo número, como por ejemplo 1910, se podría leer distinto dependiendo de si hace referencia a una contraseña (donde el texto sería «uno nueve uno cero»), o a una fecha (donde sería «mil novecientos diez»). Así, **aparecen distintos tipos de verbalizaciones posibles según el contexto.** Según los casos, habrá distintas verbalizaciones preferidas, identificadas como clases semióticas. La siguiente imagen muestra algunos de estos ejemplos.

semiotic class	examples	verbalization
abbreviations	**gov't**, *N.Y., mph*	government
acronyms read as letters	**GPU**, *D.C., PC, UN, IBM*	G P U
cardinal numbers	**12**, *45, 1/2, 0.6*	twelve
ordinal numbers	*May* **7**, *3rd, Bill Gates III*	seventh
numbers read as digits	*Room* **101**	one oh one
times	*3.20*, **11:45**	eleven forty five
dates	**28/02** *(or in US, 2/28)*	February twenty eighth
years	**1999**, *80s, 1900s, 2045*	nineteen ninety nine
money	**$3.45**, *€250, $200K*	three dollars forty five
money in tr/m/billions	**$3.45 billion**	three point four five billion dollars
percentage	**75%** *3.4%*	seventy five percent

Figura 35. Distintos ejemplos de normalización en función de la verbalización buscada. Fuente: Jurafsky, 2021.

Esta normalización para generar el texto adecuado para la verbalización se puede hacer con **reglas predefinidas.** Con estas reglas se identifican estos caracteres y, en función de la clase semiótica detectada, se normalicen de una manera u otra para generar el texto final adecuado para la verbalización. Esta tarea de normalización también se puede llevar a cabo con *encoders-decoders* cuando se dispone de un corpus de entrenamiento con los pares de caracteres y verbalización correspondiente según el contexto.

Con todo ello, un ejemplo de arquitectura TTS es la **Tacotron2** (Shen et al., 2018), que aparece ilustrada en la siguiente imagen.

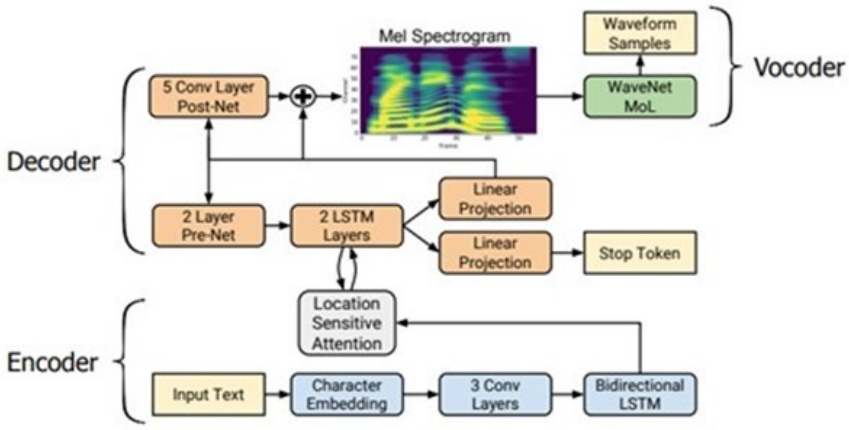

Figura 36. Arquitectura Tacotron2 para TTS. Fuente: Shen et al., 2018.

Como se puede apreciar, en la fase de *encoding* se utilizan distintas capas de redes neuronales para generar la representación de los textos de entrada, que se tratan a nivel de caracteres. Posteriormente, la fase de *decoding* general el espectrograma que se usará de entrada en vocoder para generar la señal acústica. Un elemento adicional que tiene esta arquitectura es que, en cada fase, se lleva a cabo una predicción de **stop token** de cara a que el sistema decida si tiene que seguir produciendo secuencias de salida o no.

Ideas clave

Puedes consultar más información sobre cómo es el bloque de vocoding y otros aspectos del sistema TTS, puedes consultar el capítulo 26 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall. https://web.stanford.edu/~jurafsky/slp3/26.pdf

9.7. Referencias bibliográficas

Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall.

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang Y., Wang, Y., Skerry-Ryan, R., Saurous, R. A., Agiomyrgiannakis Y., and Wu, Y. (2018). *Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions.* ICASSP.

Natural Language Processing with Python

Jurafsky, D. y Martin, J. H. (2009*). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics.* Prentice-Hall. https://web.stanford.edu/~jurafsky/slp3/

Una parte importante de las referencias del curso en general, y de este tema en particular, provienen de este libro. En concreto, para entender en más detalles las distintas aplicaciones de PLN, se puede consultar este libro. También, sirve de base para tener una referencia sobre las publicaciones científicas más relevantes asociadas a estas aplicaciones.

Deep Learning

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep Learning.* MIT Press. https://www.deeplearningbook.org/

Como se ha visto en este tema, así como en temas previos, gran parte de las arquitecturas modernas de sistemas de PLN se basan en modelos de aprendizaje profundo. Si se quiere conocer más sobre estos modelos, cómo funcionan, el detalle de cómo se entrenan se puede consultar esta referencia.

1. Indica las afirmaciones verdaderas en el contexto de las tareas de MT:

A. Un buen sistema de MT puede centrarse sólo en la traducción individual de las palabras, sin considerar el contexto general de la frase.

B. Las traducciones entre dos lenguas son directas ya que las diferencias que pueda haber entre ellas no son relevantes para hacer una traducción.

C. Los aspectos de divergencia léxica hacen referencia a diferencias entre dos lenguas en relación con la variación en el significado que puede tener una misma palabra entre ellas.

D. Ninguna de las anteriores.

2. ¿Cuál es el objetivo de usar *teacher forcing* dentro de un modelo *encoder-decoder?*

A. Conseguir que la ejecución de la fase del *encoding* sea más rápida

B. Calcular la función de coste en el entrenamiento del modelo de manera que durante cada etapa t se calcule con respecto a la palabra real de la secuencia en t-1, y no la palabra que se predijo en t-1. Con ello, se busca evitar que se degraden la calidad de las predicciones a medida que se avanza en el *decoder.*

C. Reducir el tamaño del conjunto de entrenamiento para facilitar la fase de aprendizaje del modelo.

D. Mejorar la calidad de las predicciones del modelo *encoding-decoding* mediante el uso de LM causales tanto en la parte del *encoder* como en la parte del *decoder.*

3. ¿Qué diferencias existen entre una capa *cross-attention* y otra causal *self-attention* dentro de un bloque del decoder para un modelo de MT?

A. La capa causal *self-attention* se encarga de recibir el contexto generado en el *encoder,* mientras que la capa *cross-attention* es una capa estándar *self-attention* que recibe como entrada la secuencia recorrida en el *decoder* hasta ese momento.

B. La capa *cross-attention* se encarga de recibir el contexto generado en el *encoder,* mientras que la capa causal *self-attention* es una capa estándar que recibe como entrada la secuencia recorrida en el *decoder* hasta ese momento.

C. La capa *cross-attention* es siempre la primera capa de los bloques del *decoder,* mientras que la capa causal *self-attention* es siempre la última.

D. Ninguna de las anteriores.

4. ¿Cuál de estas afirmaciones es correcta con respecto a las diferencias entre la arquitectura de un modelo basado en *transformers* para autocompletado de textos frente a otro para la generación automática de resúmenes?

A. En el caso del autocompletado de textos no se dispone desde el principio de la secuencia completa, por lo que el modelo de transformers seguirá un esquema como el de un LM causal. En cambio, en la generación de resúmenes sí que se dispone de todo el texto, con lo que el *transformer* puede ser bidireccional.

B. En el caso de la generación automática de resúmenes no se dispone desde el principio de la secuencia completa, por lo que el modelo de *transformers* seguirá un esquema como el de un LM causal. En cambio, en el autocompletado de textos sí que se dispone de todo el texto, con lo que el *transformer* puede ser bidireccional.

C. En el caso del autocompletado de textos se puede usar el concepto de *teacher forcing* en el entrenamiento, mientras que en la generación automática de resúmenes no es posible.

D. Ninguna de las anteriores.

5. ¿Cómo se llevaría a cabo un análisis de sentimientos no supervisado basado la información de sentimientos a nivel de palabras disponible en un lexicón?

A. Se generarían rasgos que caractericen el texto (ej., BoW), y con eso se entrenaría un modelo, como por ejemplo un Naive-Bayes, con el que predecir las clases (sentimientos) asociadas al texto.

B. Se caracterizarían los sentimientos de las palabras que componen cada texto y, por ejemplo, se aplicarían reglas de clasificación que asignen un sentimiento a cada texto en función del sentimiento asociado a sus palabras individuales.

C. Se generarían rasgos que caractericen el texto (ej., BoW), y con eso se entrenaría un modelo, como por ejemplo una Red Neuronal Recurrente, con el que predecir las clases (sentimientos) asociadas al texto.

D. Ninguna de las anteriores.

6. ¿Cuál de estas preguntas sería un ejemplo de *factoid question?*

A. ¿Cuál es la composición de la atmósfera de Marte?

B. ¿Cuál es la mejor forma de viajar a Roma?

C. ¿Qué me recomiendas para aprender PLN?

D. Todas las anteriores.

7. ¿Para qué propósito se usa el *inverted index* dentro de un sistema de IR para *ad hoc retrieval?*

A. De cara a mejorar el *scoring* de similitud entre el vector de la consulta y el vector de variables que representa el documento.

B. De cara a realizar una búsqueda más eficiente al usar un diccionario intermedio que indica los documentos que tienen ciertas palabras, calculando con ello la similitud entre consulta-documento solo si ese documento tiene palabras que aparezcan en la consulta.

C. De cara a poder utilizar técnicas de *embeddings* dentro del IR.

D. Ninguna de las anteriores.

8. Indica cuál de las siguientes afirmaciones es correcta en relación con un IR para *factoid QA:*

A. Se desarrollan sistemas que llevan a cabo dos etapas: 1) Para una consulta de entrada, elegir el conjunto de documentos de la colección que podrían servir para responderla. 2) Llevar a cabo una tarea de comprensión donde se encuentre el fragmento de texto dentro de un documento que responda a la consulta.

B. Dentro de que se usan documentos de una colección para responder a preguntas de un usuario, el sistema tiene que poder identificar cuándo ninguno de los documentos tiene secuencias de texto que sirvan como respuesta a una consulta particular. Para ello, se usa como token por defecto el [CLS], de manera que si se predice ese token es que no se ha encontrado respuesta.

C. Debido a que la secuencia de entrada tiene un tamaño limitado, se tiene que recorrer cada documento con un tamaño de ventana acotado al buscar la secuencia de texto que pueda responder a la consulta de entrada.

D. Todas son correctas.

9. Indica cuál de estas afirmaciones es incorrecta sobre un sistema de ASR:

A. La entrada corresponde a una señal de audio, de la que se obtienen distintos rasgos tras aplicar técnicas de muestreo, que sirven directamente de entrada a un modelo (ej., un *encoder)*.

B. Los rasgos asociados a la señal acústica de entrada dan lugar a un vector de variables de mucha mayor dimensionalidad que el vector de salida. Por este motivo se suele comprimir la información de entrada antes de que entre al modelo con el que se predice el texto.

C. El problema de ASR es un problema causal, con lo que se pueden mejorar las predicciones del modelo combinándolo con un LM.

D. Todas son incorrectas.

10. Indica cuál de estas afirmaciones es incorrecta sobre un sistema de TTS:

A. En general, se puede considerar que un sistema de TTS es dependiente del hablante, es decir, no es necesario que sea entrenado en múltiples voces distintas.

B. En la arquitectura de un modelo de TTS es necesario incluir un bloque denominado Vocoder para obtener la señal acústica asociada al espectrograma predicho por el modelo.

C. Se suelen usar modelos *encoding-decoding* donde la salida de la última capa de la red neuronal del *decoder* corresponde con la predicción de la señal acústica asociada al texto de entrada.

D. Todas son incorrectas.