Proyecto de Redes Neuronales: Similitud Semántica Textual utilizando codificadores de transformador

Por Adriana Michel Ávila García

Justificación

El proyecto consiste en utilizar una red neuronal para tratar resolver el problema planteado en la competencia de Kaggle, Quora Question Pairs, en la que se busca encontrar un modelo que, dadas dos preguntas de Quora, decida si significan lo mismo o no.

Quora es un sitio en el que los usuarios plantean preguntas que otros usuarios responden. Muchas veces las preguntas tienen el mismo significado, pero están redactadas de manera distinta. Es preferible que estas preguntas duplicadas sean identificadas, para agrupar sus respuestas, y que los usuarios no tengan que leer las respuestas a varias preguntas.

El problema planteado en la competencia de Quora Question Pairs es un problema de similitud semántica textual (Semantic Textual Similarity, o STS). Este tipo de problemas se ha estado investigando durante muchos años, y se han encontrado varias maneras de resolverlo con distintos grados de éxito.

Previamente un método comúnmente usado era entrenar una red neuronal siamesa con una red neuronal recurrente (específicamente, algún tipo de LSTM). Por otro lado, otra forma en que se ha tratado de resolver este problema es buscando modelos que puedan crear *sentence embeddings*. Estos sentence embeddings se pueden ver como representaciones abstractas (son vectores) de oraciones, en las que está capturado el significado de la oración.

Con la aparición de los transformadores en el 2017, se exploraron nuevas formas de crear sentence embeddings para la resolución de diversos problemas en el campo del procesamiento del lenguaje natural, no sólo el problema de la similitud semántica textual. Algunos modelos propuestos fueron Skip-Thought (Kiros et al., 2015), que entrena una arquitectura codificador-decodificador; InferSent (Conneau et al., 2017), que utiliza una red siamesa LSTM bidireccional con max-pooling; Universal Sentence Encoder (Cer et al., 2018), que utiliza transformadores.

Uno de los modelos más exitosos es Sentence-BERT (SBERT). SBERT es un modelo propuesto en 2019 por Reimers et. al, en el artículo Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Este proyecto se basa en gran parte en la arquitectura propuesta en dicho artículo.

Para la realización de este proyecto se decidió usar una red siamesa que contuviera una serie de encoders de transformador, de forma que para cada par de preguntas ingresadas se genere un embedding que, de cierta forma, capture el significado de las preguntas, para después comparar ambos embeddings usando alguna métrica de similitud de vectores.

Conjunto de datos utilizado

El conjunto de datos se encuentra en la [sección *data* de la página de la competencia](#).

El dataset proporcionado tenía más de 400,000 pares de preguntas en el archivo train.csv. Sin embargo, trabajamos únicamente con las que tenían 15 palabras o menos, que eran aproximadamente 300,000.

Propuesta

La arquitectura propuesta es una red siamesa con una serie de *m* capas de encoder de transformador en ambos lados, precedidas por una capa de codificación posicional (como es usado en los encoders de transformador), y seguidas por una capa de *mean pooling*. Los dos resultados de estas series de capas (uno por cada pregunta) son comparados usando la función de *cosine similarity*. El resultado es la similitud entre el par de preguntas.
Usamos *mean pooling* y *cosine similarity* porque es lo que se utiliza en la arquitectura propuesta en el artículo de SBERT.

Se hicieron pruebas con 3 versiones de la arquitectura propuesta: una con un sólo encoder, dos con varios encoders apilados, y una con un BERT de cada lado. En todas se utilizó Adam como optimizador.
A continuación explicamos los resultados obtenidos con cada una.

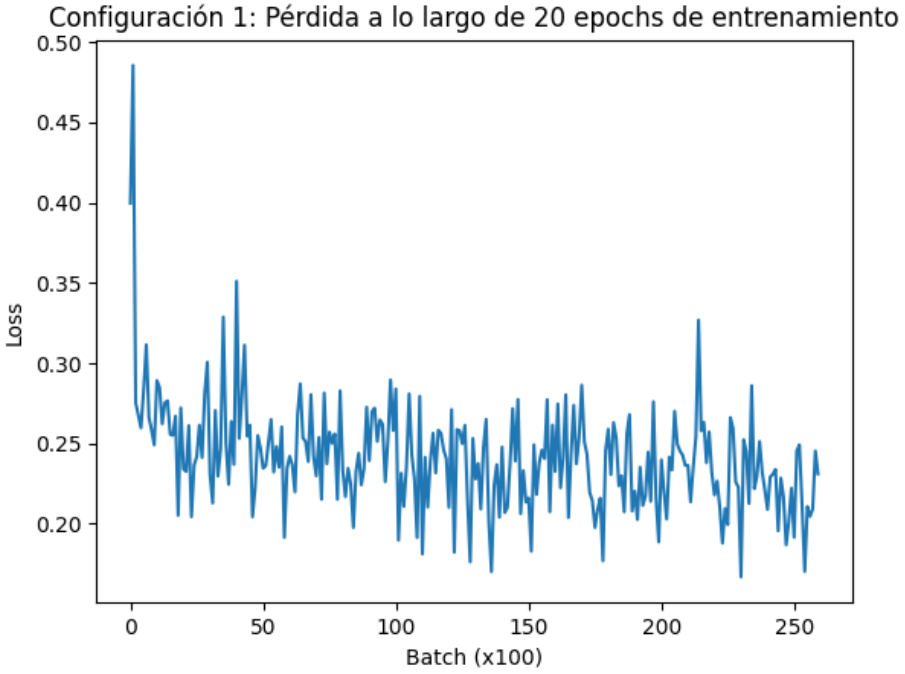
Configuración 1: Red siamesa con 4 codificadores de transformador

En esta configuración la red tenía 4 codificadores. Todos con 5 cabezas de atención, y 32 neuronas ocultas en su capa de alimentación hacia adelante. Además, tenían dropout de 0.1.
Se entrenaron dos modelos usando esta configuración: uno usando aproximadamente 80,000 ejemplares, y otro usando aproximadamente 240,000 (que es el 80% de los ejemplares que teníamos disponibles).

Ambos modelos se entrenaron con una tasa de aprendizaje de 0.0001

- **Con 80k ejemplares:**
  Tamaño de batch: 64.
  Se entrenó durante **20 epochs**. La red neuronal da como resultado, para cada ejemplar, un número entre 0 y 1, representando la similitud entre las dos preguntas. Entre más grande sea este número más similares son. Para obtener resultados binarios convertimos en 1 todos los resultados mayores a **0.65**, y el resto en 0.
  Después del entrenamiento, con la configuración 1, y esta forma de interpretar el resultado, se obtiene los siguientes valores de precisión para 4 batches random de tamaño **1024** del conjunto de prueba:

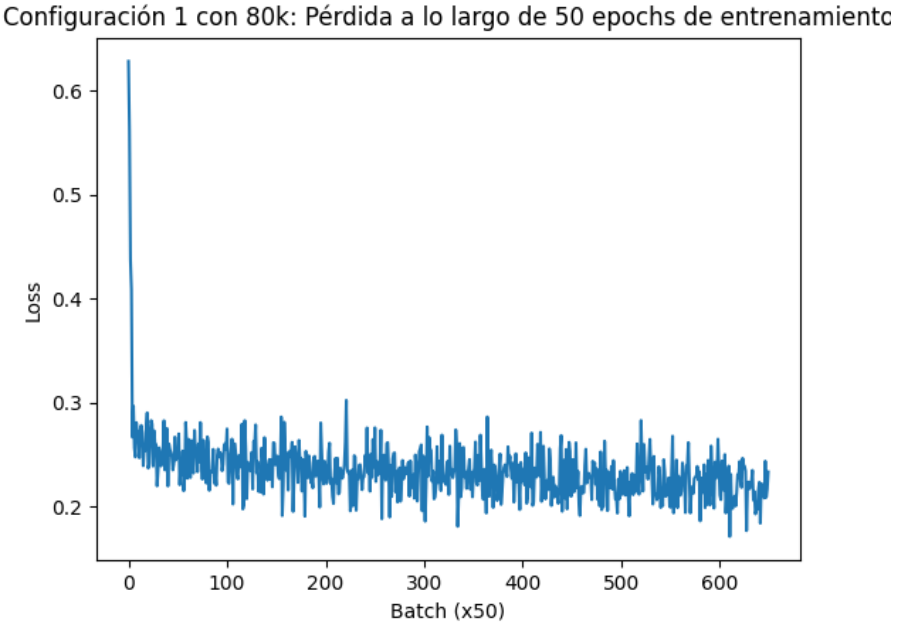
- Accuracy batch 1: 0.6728515625
- Accuracy batch 2: 0.65625
- Accuracy batch 3: 0.662109375
- Accuracy batch 4: 0.6611328125



Notamos que en los primeros batches disminuye rápidamente, pero luego parece estancarse entre los valores 0.20 y 0.25. Aunque en general parece seguir disminuyendo muy lentamente.

- **Con 80k ejemplares y 50 epochs:**
  Tamaño de batch: 128.
  Después de 50 epochs se tomó la precisión de 5 ejemplos random del conjunto de prueba:

  - Accuracy batch 0: 0.630859375
  - Accuracy batch 1: 0.625
  - Accuracy batch 2: 0.640625
  - Accuracy batch 3: 0.5859375
  - Accuracy batch 4: 0.697265625



Notamos que agregar más épocas no parece disminuir mucho la pérdida.

- **Con 240k ejemplares:**
  Tamaño de batch: 64.
  Después de **un epoch** (porque en esta configuración cada epoch tardaba lo mismo que 10 epochs en la configuración 1) se obtiene los siguientes valores de precisión para 4 batches random de tamaño **512** del conjunto de prueba:

  - Accuracy batch 1: 0.59765625
  - Accuracy batch 2: 0.609375
  - Accuracy batch 3: 0.62890625
  - Accuracy batch 4: 0.615234375

De igual forma, la pérdida parece quedarse atorada entre 0.2 y 0.3.

Configuración 2: Red siamesa con 6 codificadores de transformador

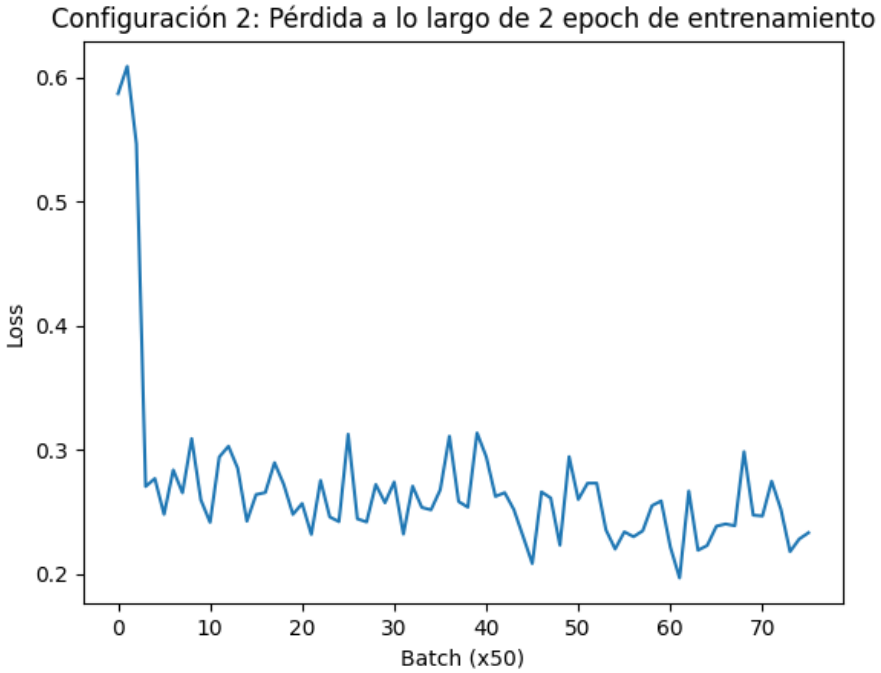
En esta configuración la red tenía 6 codificadores. Todos con 6 cabezas de atención, y 384 neuronas ocultas en su capa de alimentación hacia adelante. Además, tenían dropout de 0.1.
Nótese que esta configuración trata de ser una BERT pequeña, usando la mitad de los valores que usa BERT en sus parámetros.
En esta configuración sólo se entrenó un modelo usando aproximadamente 240,000 ejemplares, y tasa de aprendizaje de 0.00005.

- **Con 240k ejemplares:**
  Tamaño de batch: 64
  Después de **un epoch** (porque en esta configuración cada epoch tardaba lo mismo que 10 epochs en la configuración 1) se obtiene los siguientes valores de precisión para 4 batches random de tamaño **512** del conjunto de prueba:

  - Accuracy batch 1: 0.6171875
  - Accuracy batch 2: 0.6015625
  - Accuracy batch 3: 0.611328125
  - Accuracy batch 4: 0.6015625

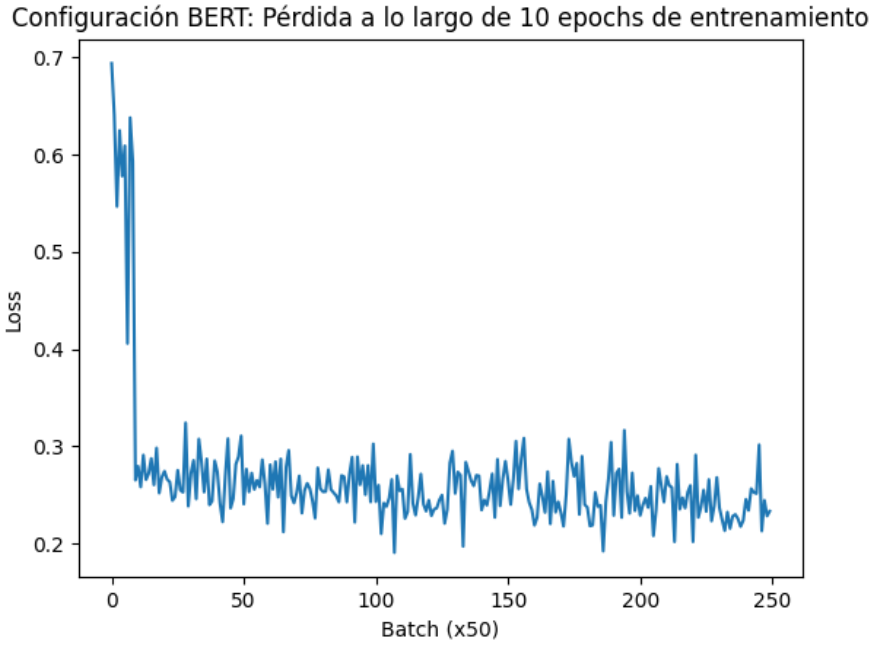
Después de **dos epoch**:

- Accuracy batch 1: 0.576171875
- Accuracy batch 2: 0.61328125
- Accuracy batch 3: 0.634765625
- Accuracy batch 4: 0.62109375

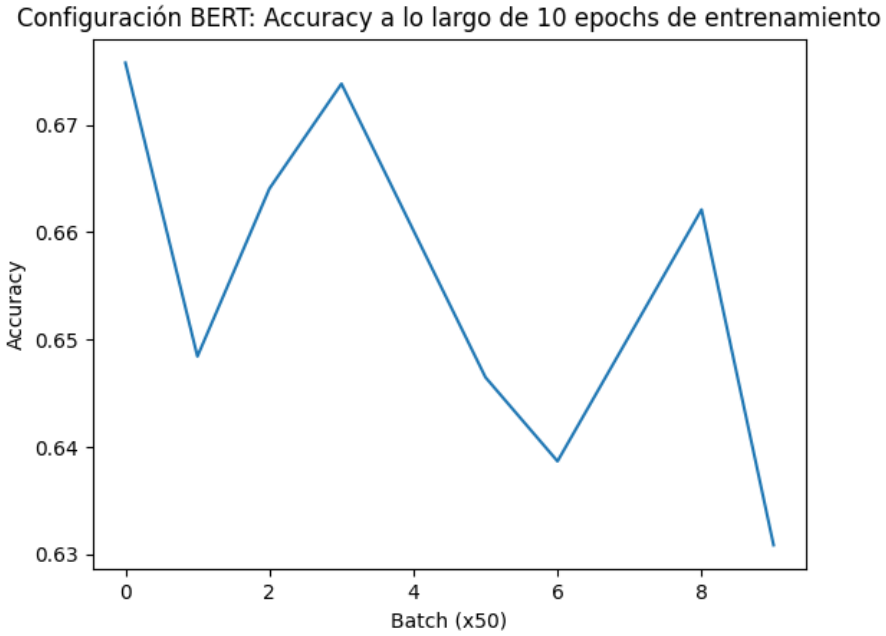


Configuración BERT: Red siamesa con 12 codificadores de transformador

En esta configuración la red tenía 12 codificadores. Todos con 12 cabezas de atención, y 768 neuronas ocultas en su capa de alimentación hacia adelante. Además, tenían dropout de 0.1. Estos son los mismos parámetros que la red BERT original, por lo que se esperaba que, si la causa de que los otros modelos se estancaran en una pérdida mayor a 2 era que la red no era lo suficientemente expresiva, con esta configuración se lograran mejores resultados.
Los batches eran de tamaño 64, y se usó una tasa de aprendizaje de 0.00002 porque es la que se utiliza en el artículo de SBERT. Se entrenó durante **10 epochs**.



En esta gráfica no se nota mucho, pero el modelo llegó a una pérdida de 0.24 y precisión de 0.67 en tan sólo **el primer epoch**, algo que los demás modelos se tardaron más épocas en alcanzar (algunos nunca llegaron a la precisión 0.67). Sin embargo, después de la primera época, la pérdida parece estancarse alrededor de 0.25, igual que los demás modelos.



Al graficar la precisión del modelo sobre el conjunto de prueba se puede notar que se está haciendo un overfitting, ya que disminuye al aumentar los epochs, especialmente a partir del tercer epoch.

Conclusiones

Ya que los modelos no mejoraron al aumentar la expresividad de la red, y parecían estancarse al aumentar las épocas, y ya que se probó también con tasas de aprendizaje muy pequeñas, pensamos que, tal vez, para mejorar su desempeño se podría probar entrenar una red con la configuración BERT en un conjunto de datos más grande. Tal vez incluso combinando el conjunto de datos de quora question pairs con otros conjuntos utilizados para entrenar redes que resuelvan problemas de similitud semántica textual.

En el artículo original de SBERT utilizaban una red BERT pre-entrenada, y únicamente utilizaban la configuración en red siamesa para mejorar los embeddings que producía. Así que creemos que esta diferencia entre la cantidad de datos con los que se entrena puede ser una causa del estancamiento.

Referencias

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. En Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (pp. 123-130). Asociación de Lingüística Computacional. Recuperado de https://arxiv.org/abs/1908.10084