Bases de Datos Masivas

Trabajo Final

Juan C. Cardona

1. Introducción

1.1 Objetivo

Realizar análisis de sentimiento sobre una colección de textos cortos y aplicando técnicas de aprendizaje profundo. El modelo de clasificación será una red neuronal convolucional (CNN) basada en una arquitectura que ha tenido éxito en tareas de procesamiento de texto libre.
También se hará uso de representaciones densas basadas en embeddings pre-entrenados de palabras con el objetivo de mejorar la performance del modelo.

1.2 Contexto

En el último tiempo, las tareas de procesamiento de lenguaje natural (NLP) para clasificación de textos libres han hecho uso de técnicas de machine learning. Sin embargo, estos modelos requieren de representaciones de vectores de features ralos y de alta dimensionalidad.

Nuevas aproximaciones que integran conocimientos del área de aprendizaje profundo han tenido éxito en la clasificación texto. El uso de redes neuronales permite aprovechar representaciones de vectores más densos que otorgan no solo beneficios computacionales sino también de generalización. A diferencia de representaciones ralas como bag of words, la cantidad de features es fijada por el usuario y se entrenan como un parámetro más de la red. De esta manera se logra hallar features abstractos pero de alto poder discriminativo.

Redes neuronales del tipo feed-forward como los perceptrones multi-capa hacen uso de un número de parámetros que supera fácilmente la decena del millón y cuyo cálculo tiene implícito un costo de cómputo alto. En este sentido las redes neuronales convolucionales son eficientes en tareas de clasificación ya que reducen este número de parámetros a partir de la búsqueda de regularidades en los datos sin importar su ubicación en la estructura.

A diferencia de textos de mayor longitud, los textos cortos carecen de contexto y suelen ser ambiguos. En casos particulares como el de los tweets, que son textos creados por usuarios de la red social Twitter, se acrecienta la aparición de términos pertenecientes al dialecto de una región o grupo social específico, así como también errores gramaticales. La solución aquí propuesta consiste, por un lado, en reforzar la tarea de preprocesamiento y normalización de los datos para optimizar el vocabulario y, por otro, en diseñar representaciones de texto más avanzadas para capturar mayor información. En concreto, se hace uso de embeddings de vectores de palabras pre-entrenados que se aprovechan de la co-ocurrencia de palabras para capturar reglas lingüísticas y suplir la falta de contexto inherente.

2. Temas abordados

2.1 Redes Neuronales

2.1.1 Definición

Son modelos organizados en nodos y capas que son capaces de aprender a clasificar grandes cantidades de datos aplicando técnicas matemáticas de optimización. Se compone de nodos/neuronas organizados en capas. Cada capa recibe un input, le aplica una funcional no lineal y genera un output para la siguiente capa. Existen capas de input y output y capas intermedias llamadas 'hidden' u 'ocultas'.

Lo interesante de este tipo de modelos es que son capaces de 'aprender' la tarea designada sin la intervención del usuario programador.

Durante la fase de entrenamiento, la red toma un subconjunto de datos y los clasifica. Luego compara estos resultados obtenidos contra los reales y ajusta los pesos de sus nodos buscando minimizar el error de la clasificación. Este último procedimiento es llamado 'backpropagation' y con él se busca perfeccionar la tarea de clasificación en cada iteración sobre los datos.

2.1.2 Redes convolucionales

Las redes neuronales convolucionales son una categoría de redes neuronales que han tenido éxito en áreas como reconocimiento de objetos y clasificación de texto. Son redes que hacen uso de una operación matemática llamada 'convolución'. La misma consiste en aplicar una operación lineal sobre una matriz y puede ser vista como una función que aplica una ventana deslizante, también llamada 'kernel' o 'filtro'[1] sobre toda la matriz de entrada y devuelve una nueva matriz llamada 'feature map'.

Una red convolucional entonces consta de una serie de capas donde una o más de ellas aplican una serie de convoluciones con diversos filtros y luego emplean funciones no lineales como ReLu o Tanh sobre los resultados. Es común que este tipo de redes incluyan también una capa de 'pooling' para realizar downsampling. La operación de 'max-pooling' por ejemplo toma los valores más altos de entre los vecinos y, de esta maneta, la red se mantiene invariante a pequeños cambios en los datos. También esta técnica aprovecha la noción de que la existencia de un feature es más importante que su ubicación exacta. El pooling además lleva a una reducción en el número de parámetros entrenables de la red, es decir, que el costo computacional disminuye en un alto porcentaje.

[1] De aquí en más los términos 'kernel' y 'filtro' serán utilizados indistintamente.

2.2 Representación de texto

2.2.1 Importancia

Lograr buenas representaciones es fundamental para el éxito de una tarea. Este fenómeno se manifiesta no solo en las ciencias de la computación sino también en la vida diaria. Así es que una persona puede realizar operaciones aritméticas sobre números arábigos con mayor facilidad que sobre números romanos. De la misma manera, no es sorpresa que la performance de algoritmos de áreas de búsqueda de conocimiento como machine-learning tengan un directo correlato con la representación elegida.

Sin embargo, la labor de extraer los elementos característicos o 'features' de una tarea no suele ser intuitiva al usuario ya que existen features que no son fácilmente visibles. En este sentido las técnicas de aprendizaje profundo permiten hallar features de alto nivel de abstracción para un diverso número de tareas diferentes.

2.2.2 Representaciones en NLP

El procesamiento de lenguaje natural busca que las computadoras sean capaces de analizar y entender el lenguaje utilizado por los humanos. A través de NLP se llevan a cabo tareas como las de traducción, resumen de textos o análisis de sentimiento. A continuación se describen dos grandes tipos de representaciones: las representaciones one-hot y las de vectores densos.

2.2.2.1 One-hot

Tradicionalmente se han usado en NLP estructuras de tipo one-hot como Bag-of-words (BoW) en donde cada documento o sentencia es descripto por la frecuencia de cada palabra del vocabulario en la colección, sin importar el orden de aparición. Cuando el documento no contiene una palabra del vocabulario, se coloca un cero. El problema de este tipo de estructuras nace del hecho de que un documento suele contar con un subconjunto muy acotado del vocabulario de la colección, lo cual lleva a generar matrices ralas o con muchos ceros y que son difíciles de computar.

Este problema se llama en la literatura 'the sparsity problem' y la búsqueda de una solución ha sido tema de discusión. El uso de técnicas como la de 'feature hashing' ha dado buenos resultados aunque lleva intrínseco el costo de pérdida de información.

2.2.2.2 Vectores densos

El gran salto que se dio en el paso de modelos de machine-learning a modelos de redes neuronales es dejar de utilizar estructuras ralas de one-hot para dar lugar a representaciones densas. Este enfoque consiste en 'embeber' o hacer un embedding de cada término en un espacio dimensional 'd' que usualmente no supera la milena. De esta manera se obtiene una matriz donde cada fila es un término del vocabulario con su grado de pertenencia a cada uno de los 'd' features de las columnas. Estos embeddings son luego

entrenados por la red como cualquier otro peso o parámetro y se pueden reutilizar entre proyectos.

Los beneficios de esta técnica son tanto computacionales como de habilidad de generalizar. El primer beneficio surge de la reducción del espacio dimensional de los vectores y la solución al sparsity problem. El poder de generalización por su parte, se debe a que features similares poseen vectores similares y es posible realizar operaciones vectoriales entre términos para derivar otros vectores.

Por último, cabe destacar que los embeddings de palabras pueden ser generados durante el entrenamiento de la red o previo al mismo. Para el último caso existen herramientas otorgadas por GloVe o Word2Vec que permiten generar embeddings a partir de un corpus dado. A su vez, otorgan embeddings pre-entrenados por redes neuronales no supervisadas aprovechando grandes corpus y de diversa índole. Dichos embeddings pueden ser fácilmente integrados a otras redes.

2.3 Análisis de sentimiento

El análisis de sentimiento es un área del NLP que clasifica textos en lenguaje natural según las emociones, opiniones o cualquier expresión de subjetividad que se manifieste. Se suelen aprovechar para capturar reacciones del público ante un producto, una persona o un evento particular. Básicamente se trata de una tarea de clasificación donde se mide el grado de polaridad de un texto en lenguaje natural.

Algunos de los desafíos que presenta el área y que serán atacados por este trabajo tienen que ver con:

- determinar qué documentos o qué porciones del documento expresan un sentimiento en particular y, además,
- determinar cuál es el sentimiento predominante del documento.

3. Evaluación experimental

En esta sección se describen en detalle los experimentos realizados y los resultados obtenidos. Primero se presenta el dataset utilizado, luego se ahonda en las particularidades de la red neuronal implementada y su configuración. Se incluye también un apartado de metodología en donde se describen los pasos seguidos para el entrenamiento y evaluación del modelo y finalmente, se analizan los resultados de dicha evaluación.

3.1 Dataset

Se utiliza la colección ‘’Twitter US Airline Sentiment”, tomada de la web Kaggle[2]. El dataset consta de casi 15 mil tweets referidos a vuelos de 6 grandes empresas de aerolíneas estadounidenses. Los datos fueron recolectados en febrero de 2015 y se clasificaron en 3 clases objetivos: positivos, neutrales y negativos.

3.2 Arquitectura de la red

A continuación se entra en detalle sobre la red neuronal convolucional implementada. Se describe su arquitectura, los hiper-parámetros seteados, los embeddings y las distintas variaciones sobre el modelo original.

3.2.1 Modelo

El modelo de red neuronal se basa en la arquitectura introducida por Yoon Kim en su paper de 2014: “Convolutional Neural Networks for Sentence Classification”. Dicha red ha tenido éxito en aplicaciones similares a la aquí presentada y ha sido citada por varios autores de renombre. La característica más destacada de este modelo es su simplicidad y versatilidad, ya que seteando unos pocos hiper-parámetros se ha adaptado a tareas de NLP de diverso calibre.

Al igual que lo ha hecho Kim en el mencionado paper, en este trabajo se entrena una red convolucional de una única capa de convolución pero con tres kernel de distinto tamaño, y que se aplica por encima de una capa de embeddings. En búsqueda de una clasificación óptima, se han realizado experimentos con vectores de embeddings pre-entrenados y sin pre-entrenar. Además, se hacen pruebas con vectores en estado estático y pruebas donde sus valores se entrenan como cualquier otro parámetro de la red.

3.2.2 Estructura de capas

La estructura de la red neuronal convolucional o CNN consiste en 8 capas que serán descriptas en orden de aparición.

[2] https://www.kaggle.com/crowdflower/twitter-airline-sentiment

3.2.2.1 Capa de input

La entrada a la red son vectores de tamaño ‘n’ donde cada vector es una sentencia o documento de la colección. ‘n’ es la longitud o cantidad de términos de cada sentencia, además es un valor fijo y será tratado como un hiper-parámetro de la red. Para optimizar el algoritmo, se reemplazan los términos por su id. Si la sentencia fuese de una longitud mayor, se toman los primeros ‘n’ términos, y si fuese de longitud menor, se rellena con ceros.

Esta capa no produce una transformación sobre la entrada y suele ser estudiada en conjunto con la capa de embedding.

3.2.2.2 Capa de embedding

Esta capa es esencialmente una tabla de lookup. Guarda la representación densa de cada término del vocabulario y, al recibir una sentencia de entrada, devuelve una matriz donde las filas son los términos de la sentencia con sus ‘f’ correspondientes features.

Inicialmente los embedding son valores aleatorios que luego se van ajustando en cada iteración o epoch sobre los datos, aunque también es posible entrenarlos por fuera de la red y cargarlos a posteriori.

La cantidad de features ‘f’ es otro de los hiper-parámetros de la red. Al momento no existe un valor universalmente aceptado, pero es común que no sobrepase la centena. Se recomienda testear distintos valores antes de definirlo.

3.2.2.3 Capa de convolución

La función de esta capa es extraer features de mayor grado de abstracción. En ese sentido, Kim propone aplicar una serie de filtros de distinto tamaño sobre la entrada. La convolución es 1-dimensional, es decir que el kernel toma el vector completo de cada término. El tamaño del filtro, entonces, será la altura del kernel o, en otras palabras, la cantidad de vectores completos de términos que serán convolucionados. De esta manera, la sentencia es procesada a diferentes resoluciones o n-gramas al mismo tiempo y el sistema aprende la mejor forma de integrarlas.

La salida de esta capa son una serie de features maps que posteriormente son reducidos por la capa de pooling.

Tanto el número de filtros como su tamaño son hiper-parámetros de la red aunque se suele recomendar que la cantidad de filtros coincida con la longitud del embedding.

3.2.2.4 Capa de max-pooling

Cada feature map es sumarizado por su propia capa de max-pooling. Es decir que existe una capa de pooling para cada tamaño de filtro aplicado. De esta manera se obtiene una estructura que contiene los nuevos features abstractos obtenidos por la capa anterior.

3.2.2.5 Capa de concatenación

Es la que se encarga de concatenar la salida de cada uno de los feature maps sumarizados por la capa de pooling.

3.2.2.6 Capa de nivelación o 'flatten'

Esta capa se encarga de nivelar la matriz o tensor resultado de la capa anterior. Lo que hace es cambiar la forma del input a la de un vector unidimensional de features, que será la estructura necesaria para realizar la clasificación.

3.2.2.7 Capa de dropout

En toda red neuronal se recomienda agregar capas de regularización de los datos para combatir el overfitting. En este caso se aplica dropout para deshabilitar una proporción de las neuronas durante el entrenamiento y así reducir la co-dependencia entre neuronas.

3.2.2.8 Capa de output

La capa de output se encarga de realizar la clasificación. Se recomienda una función de activación sigmoidea para clases binaria y softmax para multiclases.

3.2.3 Variaciones del modelo

En paralelo con el trabajo de Kim, se confeccionan tres variantes del modelo introducido en la sección anterior. Estas son:

- **CNN-rand**: Se inicializan los embeddings con valores aleatorios que se ajustan durante el entrenamiento.
- **CNN-static**: Se utilizan embeddings pre-entrenados obtenidos por el proyecto GloVe y que se encuentran abiertos al público[3]. Los mismos se mantienen estáticos durante el entrenamiento.
- **CNN-non-static**: La capa de embeddings se inicializa de la misma manera que para el modelo 'CNN-static' pero los valores se ajustan durante el entrenamiento.

Con el objetivo de comparar los resultados de la manera más igualitaria posible, los hiper-parámetros de las variaciones se mantienen uniformes.

3.2.4 Hiper-parámetros

Para la confección de este trabajo se han realizado pruebas de distintas configuraciones de red. A continuación se describe aquella que ha dado los mejores resultados.

- **Dimensionalidad de embeddings**: 200. También se obtuvieron buenos resultados con valores de 128 y 300. A mayor este valor, mayor es el impacto negativo sobre los tiempos de ejecución.

[3] https://nlp.stanford.edu/projects/glove/

- **Filtros**: Mismo valor que la dimensionalidad del embedding, 200. Se hacen las mismas consideraciones que en el ítem anterior.
- **Tamaño de filtros**: Habrá tres tamaños diferentes: 3, 4 y 5. Se ha probado con la adición de valores mayores y menores que los mencionados pero no se han visto mejoras en el accuracy del sistema.
- **Tamaño de sentencia**: Varía en cuanto a datasets. Para los tweets se ha tomado la longitud máxima de sentencia encontrada en el conjunto de entrenamiento. Para el dataset de tweets será de 51.
- **Tamaño de vocabulario**: Se toman todos los términos generados luego de las fases de preprocesamiento y normalización. Para el dataset de tweets será de 10113 términos.
- **Función de optimización**: adam.
- **Función de pérdida**: Binary cross-entropy para clasificaciones binarias y categorical cross-entropy para clasificaciones multi-clases.
- **Funciones de activación**: ReLu en la capa convolucional; sigmoidea o softmax en capa de salida según si se trata de una clasificación binaria o multiclase respectivamente.
- **Embeddings pre-entrenados**: Se utilizan aquellos proporcionados por GloVe en su página web. Para el dataset de tweets se toman los embeddings que han sido entrenados sobre un corpus de Twitter de 1.2 mil de millones de tweets.
- **Tamaño de batch**: Es la cantidad de sentencias que ingresan en batch a la red. Menores tamaños de batch mejoran el tiempo de ejecución de la red pero hacen menos precisa la estimación del gradiente. Luego de realizadas diversas pruebas, se decide tomar un tamaño de batch de 64.
- **Epochs**: Son las iteraciones del modelo sobre todos los datos. Para vectores de embeddings no estáticos el modelo converge rápidamente, siendo común sobre-ajustar en los primeros 4 o 5 epochs. En el presente trabajo, se coloca un número alto de epoch se guardan los pesos de cada iteración que mejore el accuracy y luego se evalúa sobre el modelo óptimo.

3.3 Metodología

En esta sección se describe cada una de las acciones llevadas a cabo para producir la clasificación. La metodología consta de una etapa de preprocesamiento y normalización del dataset; una etapa de división del dataset en conjuntos de entrenamiento y testeo; luego se codifican los vectores de sentencias y se realiza la clasificación sobre el conjunto de entrenamiento para obtener el modelo. La etapa final consiste en clasificar al conjunto de testeo y evaluar los resultados.

3.3.1 Preprocesamiento

Primero se extraen los siguientes tokens:

- **Urls**: Las urls del protocolo http o https se reemplazan por la expresión ‘<url>’. Ejemplo: ‘http://www.unlu.edu.ar/’

- **Menciones**: Tokens que comienzan con '@' son menciones a usuarios de la comunidad Twitter. Se reemplazan por la expresión '<user>'. Ejemplo: '@VirginAirline'.
- **Emoticones**: Son un indicativo fuerte de polaridad y es clave detectarlos. Los emoticones sonrientes se reemplazan por la expresión '<smile>' y '<lolface> según corresponda; emoticones tristes, por '<sadface>'; y emoticones neutrales, por '<neutralface>'. Se detectan tanto de izquierda a derecha como de derecha a izquierda. Ejemplos: ':-)', ':p', '(:' o ':-|'.
- **Corazones**: Otro fuerte indicativo de emocionalidad, se reemplaza por '<heart>'. Ejemplo: '<3'.
- **Números**: Se reemplazan por '<number>'. Ejemplo: '25.000'.
- **Hashtag**: Se reemplazan por '<hashtag>' y el contenido del hashtag en minúsculas y sin incluir el símbolo '#'. Ejemplo: '#NiUnaMenos', se reemplaza por: '<hashtag> niunamenos'.
- **Repeticiones de puntuación**: Tokens donde los caracteres de exclamación e interrogación se repiten, se reemplazan por el símbolo dado más la expresión '<repeat>'. Ejemplo: '!!!', se reemplaza por '! <repeat>'.
- **Alargamientos**: Casos donde la última letra de una palabra se repite se reemplaza por la palabra más la expresión '<elong>'. Ejemplo: 'siiiii', se reemplaza por 'si <elong>'.
- **Palabras en mayúscula**: Cuando una palabra esta escrita en mayúscula se reemplaza por la palabra en minúsculas más la expresión '<allcaps>'. Ejemplo: 'EXCELENTE', pasa a ser: 'excelente <allcaps>'.

El siguiente paso consiste en separar aquellas expresiones del tipo "you've" o "they're" en "you 've" o "they 're" respectivamente. También se separa en dos términos aquellos tokens que poseen alguno de los siguientes caracteres: '!', '?', '(', ')' . Finalmente se eliminan los caracteres especiales y se pasan todos los términos a minúsculas.

3.3.2 Conjunto de entrenamiento y testeo

El dataset original se divide en dos conjuntos generados aleatoriamente: uno de entrenamiento y otro de testeo. Este último cuenta con el 20% de los datos. Cabe destacar que se hace una división estratificada de la colección para conservar la proporción de ejemplos en cada clase objetivo.

En este trabajo se utilizan los mismo conjuntos para ejecutar cada una de las variaciones del modelo de red neuronal y así poder evaluar y comparar los resultados.

3.3.3 Codificación de sentencias

A partir del conjunto de entrenamiento se crea el vocabulario de la colección y un mapeo de término a un entero identificador o 'id'. Cada sentencia, documento o tweet se codifica de la siguiente manera:

1) Se reemplaza cada término por su id.

2) Se normaliza el tamaño de la sentencia según el hiper-parámetro ‘s’ correspondiente, si ‘s’ es menor a la longitud de la sentencia se toman los primeros ‘s’ términos; caso contrario, si la longitud es mayor, se rellena con ceros.

Los mismos pasos se realizan para el conjunto de testeo. Tener en cuenta que si un término no se encuentra en el vocabulario, se reemplaza por el valor cero.

3.3.4 Entrenamiento de la red

Una vez cargada la arquitectura del modelo y codificadas las sentencias, llega el momento de entrenar la red. Aquí entran en juego los hiper-parámetros de epochs y tamaño de batch. En cada epoch se evalúa la performance del modelo en base a un conjunto de validación y si se observa una mejoría, se guardan los pesos de la red.

3.3.5 Clasificación del conjunto de testeo

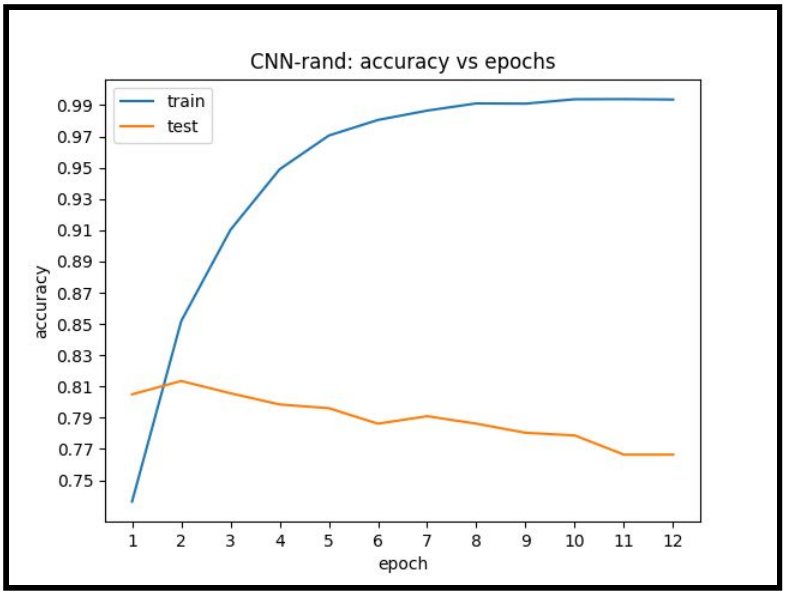
Con el modelo generado se realiza la clasificación del conjunto de testeo. En esta etapa, se generan las métricas de evaluación que serán revisadas en detalle en la próxima sección. En esta etapa es importante desactivar la capa de dropout debido a que ya no es necesaria aplicar regularización y puede derivar en clasificaciones menos precisas.

3.4 Resultados

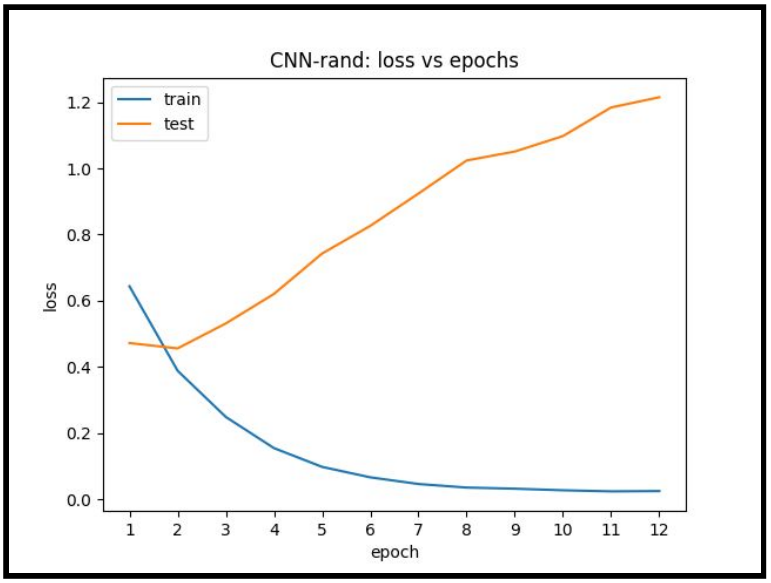
En primer lugar se analizan los valores de accuracy y loss en los epoch de cada modelo y se decide en cuál de ellos se realiza el corte. Luego, se confecciona la matriz de confusión de cada modelo y se evalúa en base a métricas de precisión, recall y f-score.

3.4.1 Accuracy y Loss

Para el modelo CNN-static es inmediato decidirse por el segundo epoch. A continuación el gráfico de accuracy:

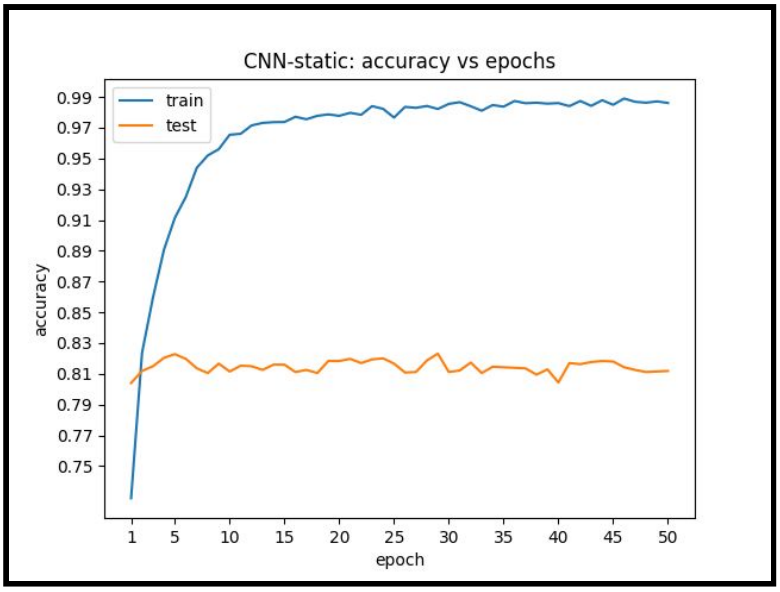


Se observa que la curva de test (en color naranja) llega a su punto máximo en el epoch 2, donde toma el valor de 0,8149. Si quedaran dudas, se acude al gráfico de loss vs epoch:



Aquí también se puede observar que el epoch 2 es el más conveniente ya que el valor de loss es minimizado. Por lo tanto el segundo epoch optimiza nuestro modelo y será el modelo seleccionado.

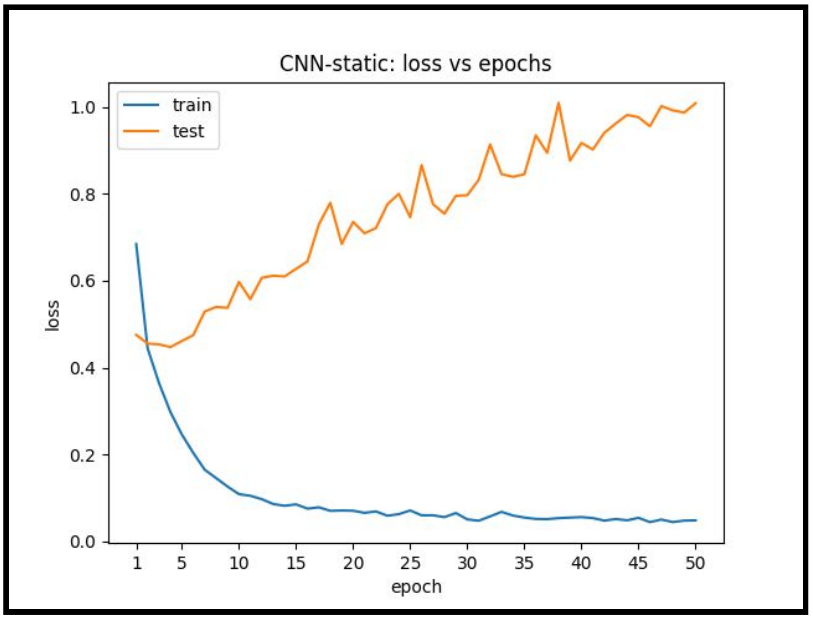
En el caso del modelo CNN-static, sin embargo, la decisión no es tan fácil de tomar. Primero echemos vistazo al gráfico de accuracy:



No parecen observarse grandes variaciones en la curva de test a lo largo de la ejecución aunque la curva de entrenamiento sobre ajusta rápidamente.

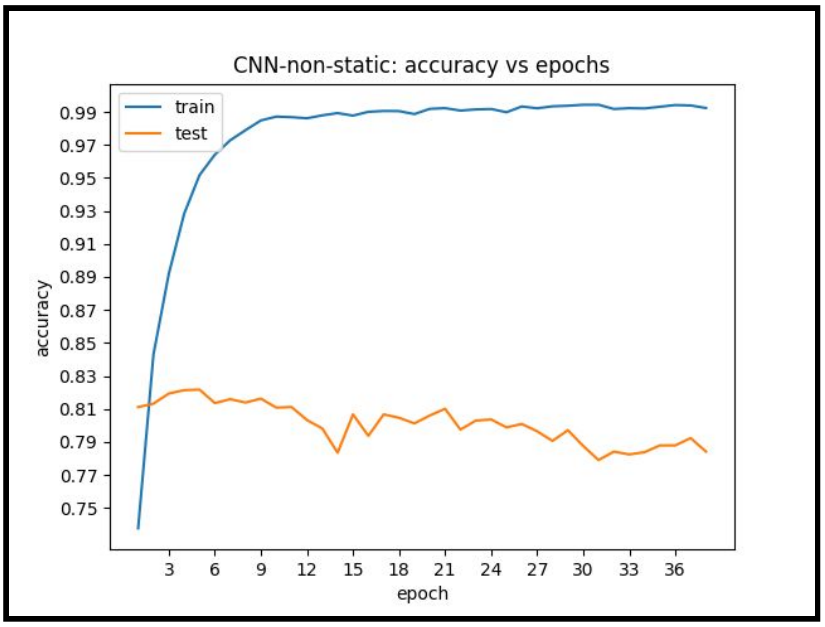
Si inspeccionamos el gráfico con más detalle, se ve que la curva de test aumenta en cada paso hasta el epoch 5, luego del cual comienza a descender hasta el epoch 9 y se mantiene estable entre mejoras y disminuciones de accuracy. Sin embargo, en el epoch 29 la curva alcanza su punto máximo logrando 0,8231 de accuracy, valor que supera levemente los 0,8227 del epoch 5.

Para terminar de definir la elección, es necesario acudir al gráfico de loss:

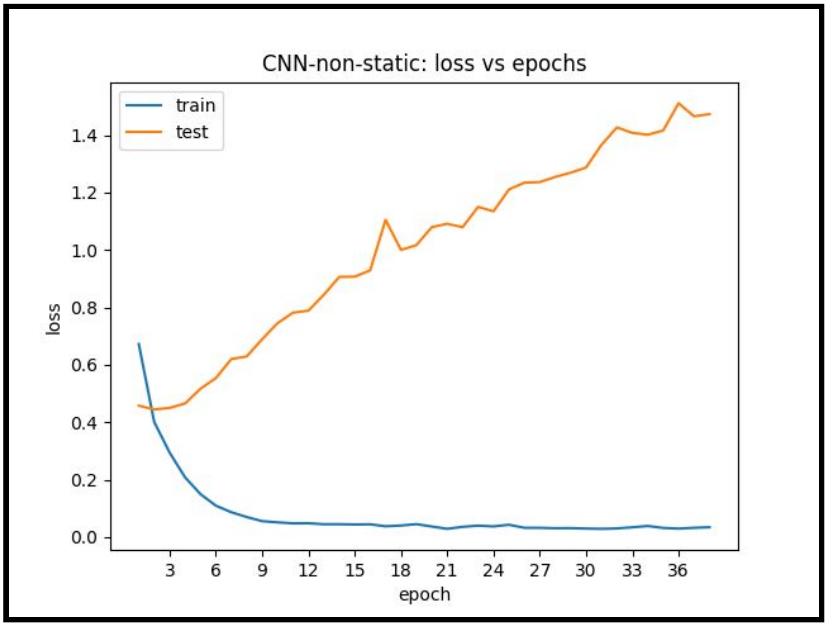


De manera similar, la curva de test disminuye hasta el epoch 5 para luego comenzar la pendiente ascendente. En el epoch 29, el valor de loss alcanza a ser casi de un 80% y si bien es un mínimo local, cortar en este epoch podría ser arriesgado ya que la posibilidad de caer en overfitting es alta. Por lo tanto, se decide en este trabajo cortar en el epoch 5.

Finalmente, observemos los resultados del modelo CNN-non-static:



El accuracy máximo es 0,8217 en el epoch 5. Esta elección la confirmamos observando el gráfico de loss:



Los resultados finales se resumen en la siguiente tabla:

Modelo	Epoch	Loss	Accuracy
CNN-rand	2	0,4559	0,8149
CNN-static	5	0,4613	**0,8227**
CNN-non-static	5	0, 517	0,8217

Se marca en negrita el accuracy del modelo ganador.

3.4.2 Matriz de confusión

La matriz de confusión es útil para sumarizar las clasificaciones correctas e incorrectas. Las columnas representan las clasificaciones realizadas o predichas y las filas representan la verdadera clasificación para cada uno de los registros del conjunto de testeo.

A continuación se adjunta la matriz de confusión y la matriz normalizada de cada variación en el modelo.

Modelo CNN-rand:

Matriz de confusión	Negativa	Neutral	Positiva
Negativa	1640	162	34
Neutral	172	401	47
Positiva	64	63	345

Matriz de confusión normalizada	Negativa	Neutral	Positiva
Negativa	0.89	0.09	0.02
Neutral	0.28	0.65	0.08
Positiva	0.14	0.13	0.73

Una alta proporción de ejemplos clasificados como negativos fueron correctamente clasificados, no así para la clase neutral que muestra una cantidad de verdaderos positivos aceptable pero mucho menor. Cabe mencionar que la clase objetivo ‘Negativa’ es también la de mayor cantidad de ejemplos en el dataset.

Modelo CNN-static:

Matriz de confusión	Negativa	Neutral	Positiva
Negativa	1696	111	29
Neutral	211	370	39

Positiva	73	56	343

Matriz de confusión normalizada	Negativa	Neutral	Positiva
Negativa	0.92	0.06	0.02
Neutral	0.34	0.60	0.06
Positiva	0.15	0.12	0.73

Con respecto al modelo anterior la cantidad de verdaderos positivos para la clase ‘Negativa’ ha aumentado considerablemente, al mismo tiempo que las demás clases se mantuvieron en valores similares.

Modelo CNN-non-static:

Matriz de confusión	Negativa	Neutral	Positiva
Negativa	1694	96	46
Neutral	219	345	56
Positiva	61	44	367

Matriz de confusión normalizada	Negativa	Neutral	Positiva
Negativa	0.92	0.05	0.03
Neutral	0.35	0.56	0.09
Positiva	0.13	0.09	0.78

Si bien este modelo muestra valores muy buenos para la clase ‘Negativa’, existe una diferencia muy grande con respecto a la clase ‘Neutral’, que tiene una tasa de verdaderos positivos que apenas supera el 50%. Con respecto a los modelos anteriores, muestra la mejor clasificación para la clase ‘Positiva’.

3.4.3 Precision, recall y f-score

El score de precisión es la proporción de los ejemplos clasificados como positivos que efectivamente lo son. El recall por su parte toma la proporción de ejemplos positivos que fueron clasificados como tales. En este sentido se puede decir que la precisión es una medida de 'exactitud' y el recall, de 'completitud'. Se suelen usar ambas métricas por separado o en conjunto a través de la medida de F-score.

Se procede a mostrar estos valores para cada uno de los modelos. De manera complementaria se agrega el valor de soporte o 'support' para explicitar el peso de cada clase en la colección.

Modelo CNN-rand:

Clase	Precision	Recall	F-score	Support
Negativa	0.87	0.89	0.88	1836
Neutral	0.64	0.65	0.64	620
Positiva	0.81	0.73	0.77	472

Los valores de precision y recall son parejos entre clases. El mayor valor de f-score es el de la clase 'Negativa' y el menor, el de la clase 'Neutral', valores ambos que son consistentes con la matriz de confusión.

Modelo CNN-static:

Clase	Precision	Recall	F-score	Support
Negativa	0.86	0.92	0.89	1836
Neutral	0.69	0.6	0.64	620
Positiva	0.83	0.73	0.78	472

Con respecto al modelo anterior, el f-score es superado para todas las clases. El recall de la clase 'Negativa' indica que un alto porcentaje de ejemplos de esta clase fueron clasificados correctamente. Esto en detrimento de la precisión, la cual ha disminuido. Sin embargo este comportamiento no es compartido por la clase 'Neutral' que sigue siendo baja en relación. Cabe mencionar, sin embargo, que la precisión de la clase 'Positiva' es particularmente alta.

Modelo CNN-non-static:

Clase	Precision	Recall	F-score	Support

Negativa	0.86	0.92	0.89	1836
Neutral	0.71	0.56	0.62	620
Positiva	0.78	0.78	0.78	472

Este modelo tiene:

- el valor más bajo de recall para la clase ‘Neutral’,
- una precisión en la clase ‘Positiva’ que es menor con respecto al modelo estático y además,
- no alcanza a mejorar el f-score de la clase ‘Negativa’.

4. Conclusiones

En este trabajo se ha expuesto el uso de redes neuronales convolucionales en tareas de análisis de sentimiento sobre documentos de texto libre y usando representaciones densas de palabras. Los resultados obtenidos de los experimentos han dado una muestra más de su practicidad y eficacia.

Sin el uso de complejas transformaciones sobre los datos y con un sencillo refinamiento de hiper-parámetros se ha conseguido una performance de clasificación que compite o incluso supera otros métodos más populares del machine-learning. La razón se debe a que las redes CNN logran capturar features de calidad a partir del uso de filtros de distinto tamaño y que se depuran gracias a las capas de pooling.

Si a la CNN se le agregan embeddings de palabras pre-entrenados se logra incluso un poder de generalización mayor ya que es posible adquirir conocimiento que trasciende a la colección de entrenamiento.

5. Bibliografía

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification.
Yoav Goldberg. 2015. A primer on Neural Network Models for Natural Language Processing.
Jin Wang, Zhongyuan Wang, Dawei Zhang, Jun Yan. 2017. Combining Knowledge with Deep Convolutional Neural Networks for Short Classification.
Nádia F.F. da Silva, Eduardo R. Hruschka, Estevam R. Hruschka Jr. 2014. Tweet Sentiment analysis with classifier ensembles.
Erfaneh Gharavi, Kayvan Bijari. 2017. Short text classification using deep representation: A case study of Spanish tweets in Coset Shared Task.
Aliaksei Severyn, Alessandro Moschitti. 2015. Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks.
Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.
Ye Zhang, Byron C. Wallace. 2016. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.
Jiawei Han, Micheline Kamber, Jian Pei. 2012. Tercera edición. Data Mining: Concepts and Techniques.
Daniel T. Larose. 2014. Segunda edición. Discovering Knowledge in Data: An Introduction to Data Mining.