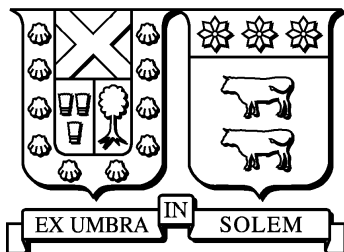


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INFORMÁTICA

SANTIAGO – CHILE



“HASHING ADAPTATIVO BASADO EN REDES
NEURONALES PARA BÚSQUEDA POR
SIMILITUD”

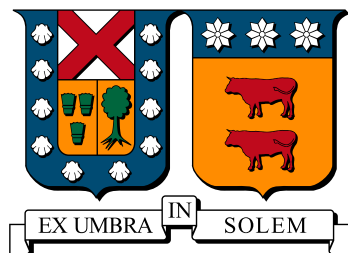
JOAQUÍN EDUARDO VELÁSQUEZ ARAYA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: RICARDO ÑANCULEF

JUNIO 2017

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



**“HASHING ADAPTATIVO BASADO EN REDES
NEURONALES PARA BÚSQUEDA POR
SIMILITUD”**

JOAQUÍN EDUARDO VELÁSQUEZ ARAYA

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO**

PROFESOR GUÍA: RICARDO ÑANCULEF
PROFESOR CORREFERENTE: CARLOS VALLES

JUNIO 2017

MATERIAL DE REFERENCIA, SU USO NO INVOLUCRA RESPONSABILIDAD DEL AUTOR O DE LA INSTITUCIÓN

Agradecimientos

Todo viaje tiene un inicio, aventuras en el trayecto, un climax y un desenlace, este es el desenlace de mi historia en la universidad. Muchas personas fueron parte de este viaje y lo mínimo que puedo hacer es agradecerles.

En primer lugar, agradezco a mi mamá, mi tío y a mis hermanas por estar ahí siempre, ya sea brindando el apoyo necesario, el consejo, orientación y por esos retos necesarios para no salirse del camino. Dudo haber llegado a donde estoy ahora si no fuera por mi familia y espero ser un hombre del que siempre se sientan orgullosos.

Agradezco también al departamento de informática, a los docentes, académicos y funcionarios, por el conocimiento y la ayuda que ofrecieron durante estos años, el mejor departamento de la UTFSM.

En la universidad conocí a mucha gente, muchos recuerdos y muchos aprendizajes. Con CEEINF 2011-2012 y P! aprendí lo importante de la unidad, el compañerismo y que hay ideales por los que vale la pena luchar. Con los primos no solo conocí gente buena sino que comprendí la importancia de un buen ambiente laboral, a donde sea que me lleve la vida espero encontrar un ambiente similar o contribuir en su creación. No puedo no mencionar a Barco Chiquito, hermandad de WOW, equipo de baby-futbol, algunos ya no están ni en el país pero tengo muy buenos recuerdos con ustedes. Al Erick, eterno compañero de tareas, FESW incluida, con quien siempre pude contar y de quien aprendí bastante.

En este viaje he conocido a mucha gente, hasta tuve enemigos, y si bien no los menciono a todos aquí quiero que sepan lo siguiente: gracias por haber sido parte de este viaje, fue un honor.

Resumen

Este trabajo está enfocado en la búsqueda por similaridad de documentos de texto, por ello se busca un modelo para obtener una representación binaria de documentos de texto que refleje la similitud semántica entre ellos y alcance una alta precisión.

La búsqueda por similitud de documentos de texto corresponde a obtener los documentos dentro de una colección que resultan semánticamente similares respecto a un documento de consulta, es decir, que están relacionados con dicha consulta en función de su significado o contenido. Una representación binaria de estos documentos que refleje su similitud semántica permite operar en el espacio de Hamming, en donde las operaciones necesarias para comparar las representaciones son de menor complejidad. En la recuperación de información suele ser relevante recuperar una baja cantidad de documentos pero alcanzando una alta precisión.

Se realizó una implementación en Python basado en el modelo propuesto en [28] para generar representación binaria de documentos de texto y se evaluó su desempeño variando parámetros del modelo. Este modelo fue modificado para experimentar con distintas arquitecturas, utilizar *Constrained Poisson Model* y se suprimió el ruido. De este modo, se encontraron modelos con alta precisión y poco profundos para la recuperación de una baja cantidad de documentos.

Abstract

This work is focus on similarity search of text documents, for this reason it shows a model to obtain a binary representation of text documents that reflects the semantics similarity between them and reach a high precision.

Similarity search of text documents corresponds to obtaining the documents within a collection that is semantically similar to a query document, that is, they are related to the query according to its meaning or content. A binary representation of these documents that reflects their semantic similarity allows to operate over Hamming space, where the operations necessary to compare the representations have less complexity. In information retrieval it is usual to recover a low number of documents but attaining high accuracy.

A model baser on [28] was implemented on Python to generate binary representacion of text documents and its performance was evaluated by varying model parameters. This model was modified to experiment with different architectures to use *Constrained Poisson Model* and the noise was suppressed. In this way, shallow models with high precision and for the low amount recovery were found.

Índice de Contenidos

Agradecimientos	III
Resumen	IV
Abstract	V
Índice de Contenidos	VI
Lista de Cuadros	IX
Lista de Figuras	X
Glosario	XV
Notación	XVI
1. Introducción	1
1.1. Descripción del Problema	5
1.2. Estrategia de Solución Adoptada	6
1.3. Objetivos	8
1.4. Contribuciones de esta memoria	8
2. Estado del Arte	10

2.1.	Hashing	10
2.1.1.	Búsqueda mediante hashing	11
2.1.2.	Aplicaciones	13
2.2.	Representación de Texto	13
2.2.1.	Vocabulario	14
2.2.2.	Word Count Vector	15
2.3.	Machine Learning	16
2.3.1.	Algoritmo de aprendizaje	16
2.3.2.	Aplicaciones	18
2.4.	Red Neuronal Artificial	18
2.4.1.	Entrenamiento	20
2.4.2.	Restricted Boltzman Machine	22
2.4.3.	Autoencoders	25
2.4.4.	Deep Belief Network	26
2.5.	Tipos de Dataset	29
2.5.1.	Métricas de Resultados	29
3.	Propuesta e implementación de solución	35
3.1.	Red Neuronal	35
3.1.1.	CPM	36
3.1.2.	DBN	38
3.2.	Dataset	41
3.2.1.	20 Newsgroups	41
3.2.2.	Reuters	42
3.3.	Pre-Procesamiento	42
3.4.	Representación	43

3.5. Métricas de Resultados	45
4. Resultados	47
4.1. Comparación de desempeño con baseline	47
4.1.1. 20 Newsgroups	47
4.1.2. Reuters	48
4.2. Evaluación de sensibilidad de resultados	49
4.2.1. 20 Newsgroups	50
4.2.2. Reuters	65
5. Conclusiones	87
5.1. Conclusiones	87
5.2. Trabajo futuro	89
Bibliografía	90

Índice de cuadros

4.1. F1 score promedio de las arquitecturas para recall hasta el 100 %, 10 % y 1 % en 20 Newsgroups.	62
4.2. Precisión para distintos Recall de las arquitecturas entrenadas en 20 Newsgroups.	65
4.3. F1 score promedio de las arquitecturas para recall hasta el 100 %, 10 % y 1 % en Reuters según el criterio predictivo con promedio micro.	80
4.4. F1 score promedio de las arquitecturas para recall hasta el 100 %, 10 % y 1 % en Reuters según el criterio predictivo con promedio macro.	81
4.5. Precisión para distintos Recall de las arquitecturas entrenadas en Reuters según el criterio predictivo con promedio micro.	85
4.6. Precisión para distintos Recall de las arquitecturas entrenadas en Reuters según el criterio predictivo con promedio macro.	86

Índice de figuras

2.1. DBN [1]	26
4.1. Precisión vs recall entre la implementación obtenida y la implementación original [28].	48
4.2. Comparación de precisión vs recall entre la implementación obtenida y la implementación original, se muestra el desempeño según el criterio micro y el criterio macro.	49
4.3. Precisión vs recall de arquitecturas con 1 capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	51
4.4. Precisión vs recall de arquitecturas con 2 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	52
4.5. Precisión vs recall de arquitecturas con 3 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	53
4.6. Precisión vs recall de arquitecturas con 4 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	54

4.7. Precisión vs recall de arquitecturas con 5 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	55
4.8. Precisión vs recall de arquitecturas con 250 neuronas por capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall. .	57
4.9. Precisión vs recall de arquitecturas con 500 neuronas por capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall. .	58
4.10. Precisión vs recall de arquitecturas con 1000 neuronas por capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall. .	60
4.11. Precisión vs recall de las arquitecturas con mejor desempeño. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	63
4.12. Precisión vs recall de arquitecturas con 1 capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	66
4.13. Precisión vs recall de arquitecturas con 1 capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	67
4.14. Precisión vs recall de arquitecturas con 2 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	68

4.15. Precisión vs recall de arquitecturas con 2 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	68
4.16. Precisión vs recall de arquitecturas con 3 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	70
4.17. Precisión vs recall de arquitecturas con 3 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	70
4.18. Precisión vs recall de arquitecturas con 4 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	71
4.19. Precisión vs recall de arquitecturas con 4 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	72
4.20. Precisión vs recall de arquitecturas con 5 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	73
4.21. Precisión vs recall de arquitecturas con 5 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	73

4.22. Precisión vs recall de arquitecturas con 250 neuronas por capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	75
4.23. Precisión vs recall de arquitecturas con 250 neuronas por capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	75
4.24. Precisión vs recall de arquitecturas con 500 neuronas por capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	77
4.25. Precisión vs recall de arquitecturas con 500 neuronas por capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	77
4.26. Precisión vs recall de arquitecturas con 1000 neuronas por capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	78
4.27. Precisión vs recall de arquitecturas con 1000 neuronas por capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	79
4.28. Precisión vs recall de arquitecturas con mejor desempeño según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	83

4.29. Precisión vs recall de arquitecturas con mejor desempeño según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.	83
---	----

Glosario

- *BPSD*: Búsqueda por Similitud en Documentos de texto.
- *TF*: Term Frequency
- *IDF*: Inverse Document Frequency
- *TF-IDF*: Term Frequency - Inverse Document Frequency
- *ANN*: Red Neuronal Artificial
- *RBM*: Restricted Boltzman Machine
- *CD*: Contrastive Divergence
- *EBM*: Energy Based Model
- *CPM*: Constrained Poison Model
- *AE*: Autoencoder
- *DAE*: Denoising Autoencoder
- *DBN*: Deep Belief Network

Notación

- k : Cantidad.
- D : Dataset.
- x, y : Elemento del Dataset.
- x_i, y_i : Atributo i del elemento x, y .
- p : Parámetro Distancia Minkowski
- $d(x, y)$: Función de distancia entre x e y .
- $Euclidiana(x, y)$: Distancia Euclidiana entre el elemento x e y .
- $Minkowski(x, y)$: Distancia Minkowski entre el elemento x e y .
- $Hamming(x, y)$: Distancia de Hamming entre el elemento x e y .
- $Jaccard(x, y)$: Similitud de Jaccard entre el elemento x e y .
- $DisJaccard(x, y)$: Distancia de Jaccard entre el elemento x e y .
- $hash(x)$: Función de hashing
- d : Documento del dataset D .
- t : Término t del documento d .
- $fr(t, d)$: Frecuencia del término t en documento d .
- $tf(t, d)$: TF del término t en documento d .

- $tfn(t, d)$: *TF* normalizado del término t en documento d .
- $idf(t, D)$: *IDF* del término t de dataset D .
- $tfidf(t, d, D)$: *TF-IDF* del término t en documento d de dataset D .
- $a(x)_j$: Función de pre-activación de neurona j con elemento x de entrada.
- w_{ij} : Peso entre neurona i y neurona j .
- W : Matriz de pesos entre neuronas de distintas capas.
- a_i : sesgo neurona i .
- b_j : Sesgo de neurona j .
- $h(x)$: Activación según elemento x .
- $g(a(x)_j)$: Función de activación neurona j según elemento x .
- $\sigma(a(x)_j)$: Función sigmoid para neurona j según elemento x .
- $\tanh(a(x)_j)$: Función tanh para neurona j según elemento x .
- $f(x)$: Función que genera la representación de x .
- z : Representación esperada de un elemento.
- $l(f(x), z)$: *Loss function* que compara representación generada por la red $f(x)$ con el resultado esperado z .
- v : Capa visible v .
- v_i : Neurona i de v .
- h : Capa oculta h .
- h_i : Neurona i de h .
- $E(v, h)$: Energía asociada a capa visible v y capa oculta h .
- $P(v_i, h)$: Probabilidad que neurona visible i se active respecto a capa oculta h .

- $P(h_j, v)$: Probabilidad que neurona oculta j se active respecto a capa visible v .
- v' : Reconstrucción de capa visible en Contrastive Divergence.
- h' : Reconstrucción de capa oculta en Contrastive Divergence.
- ϵ : learning rate.
- $PS(n, \lambda)$: Distribución de Poisson
- λ : Valor esperado de Poisson.
- q_i : Consulta i .
- t_j : Etiqueta j .
- $c(d)$: Conjunto de etiquetas del elemento d .
- TS_i : Conjunto de elementos que comparten la misma etiqueta con la consulta q_i
- TM_i : Conjunto de elementos que comparten al menos una etiqueta con la consulta q_i
- S_i : Conjunto de elementos recuperados por q_i .
- s_k : Elemento recuperado mediante q_i .
- r : Tamaño del radio de búsqueda de elementos similares.
- p_j : Precision para etiqueta j
- r_j : Recall para etiqueta j .
- p_i : Precision para consulta i .
- r_i : Recall para consulta i .
- P : Precision conjunto de prueba.
- R : Recall conjunto de prueba.
- n : Cantidad de consultas realizadas.

- $Jaccard(q_i, s_i)$: Similitud de Jaccard para consulta i para un documento recuperado s_i .
- SJ_i : Similitud de Jaccard de consulta q_i .
- SJ : Similitud de Jaccard del conjunto de prueba.
- n_{j++} : Contador de verdaderos positivos.
- n_{j+-} : Contador de falsos positivos.
- n_{j-+} : Contador de falsos negativos.
- ns_i : Documento no recuperado por consulta i .

Capítulo 1

Introducción

La búsqueda por similitud corresponde al problema de recuperar los elementos dentro de un conjunto de datos $D = \{x^{(i)}\}_{i=1}^N$, $x^{(i)} \in X$, que son similares a un cierto elemento de referencia $q \in X$, es decir, identificar los elementos de D que presenten criterios en común con q para una determinada aplicación de interés. Desarrollar algoritmos eficientes para esta tarea es importante debido al volumen cada vez mayor de información almacenada digitalmente y que en la actualidad, la búsqueda por similitud es un proceso fundamental para el desarrollo de múltiples aplicaciones en Minería de Datos e Informática en general, entre las que se pueden destacar:

- **Clustering:** En clustering, se buscan algoritmos capaces de agrupar elementos en función de su similitud. Algunas de las técnicas más utilizadas en esta área, como *DBSCAN* [4] y *k-means*, [7] corresponden a algoritmos iterativos donde cada iteración requiere la solución de múltiples problemas de búsqueda por similitud.
- **Reconocimiento de Patrones:** En reconocimiento de patrones, se busca implementar una función capaz de reconocer la categoría a la que pertenecen nuevos elementos en función de otros ya conocidos. Una técnica muy popular para esto es *k-nearest neighbor*, en donde se recuperan los k elementos más similares al objeto a reconocer y se predice una categoría en función de las categorías de estos k elementos [25].

- **Bio-Informática:** En bio-informática, se desea encontrar la relación entre sujetos aparentemente no relacionados. Para ello se busca encontrar extractos de secuencias de ADN que son similares entre sí, lo que corresponde a resolver múltiples problemas búsqueda por similitud sobre todas las posibles sub-secuencias de las secuencias originales.
- **Base de Datos:** Dentro de una base de datos es posible dar un orden a los elementos almacenados de modo que al realizar una consulta en base a los criterios deseados sea posible identificar y recuperar aquellos que coincidan, de forma parcial o total.
- **Recuperación de Elementos Multimedia:** Para las imágenes, audios y videos no es posible realizar consultas con métodos de indexación tradicionales pues estos elementos no se pueden ordenar ni es posible corroborar su igualdad. Para encontrar los elementos que coinciden con una consulta es necesario definir que atributos serán comparados, una medida de similitud entre ellos y dar una representación adecuada. [5]
- **Recuperación de Texto:** Los documentos de texto presentan distintas estructuras para temas similares, por ello la similitud no puede ser medida simplemente por la ocurrencia de ciertas palabras. Es por esto que requieren de una representación que refleje la similaridad semántica que existe entre los distintos documentos.

Para definir de modo más formal el problema de búsqueda por similitud, es necesario suponer la existencia de una medida que refleje la similaridad o disimilaridad entre un par de elementos $x, y \in X$. La aproximación tradicional [5], consiste en dotar X de una estructura métrica, es decir, definir una función $d : X \times X \rightarrow \mathbb{R}$, que denominaremos distancia, y que satisface las siguientes propiedades:

- $\forall x, y \in X, d(x, y) \geq 0$
- $\forall x, y \in X, d(x, y) = d(y, x)$
- $\forall x \in X, d(x, x) = 0$
- $\forall x, y, z \in X, d(x, y) \leq d(x, z) + d(z, y)$

Una vez que la métrica ha sido establecida, existen básicamente tres criterios [5] para definir qué documentos debiesen recuperarse en una búsqueda por similitud:

- **Range or proximity query:** Para una consulta $q \in X$ se desea recuperar todos los elementos en $D = \{x^{(i)}\}_{i=1}^N$ cuya distancia sea menor o igual a r .

$$S_q = \{x \in D : d(q, x) \leq r\}. \quad (1.1)$$

- **Nearest neighbor query:** Para una consulta $q \in X$ se desea recuperar el elemento de en $D = \{x^{(i)}\}_{i=1}^N$ que minimiza la distancia al consultado.

$$S_q = \{x \in D : d(q, x) \leq d(q, y), \forall y \in D\}. \quad (1.2)$$

- **k-Nearest neighbor query:** Similar al anterior, pero en vez de recuperar sólo un elemento se recuperan los k elementos más cercanos.

En la literatura existen distintos métodos para medir la distancia entre dos elementos. Estos varían según el criterio utilizado y el tipo de elementos a comparar [36]. Las siguientes medidas son ampliamente utilizadas en minería de datos.

- **Distancia Euclidiana:** Asumiendo que los elementos se encuentran representados vectorialmente, i.e., que $X \subset \mathbb{R}^p$, la distancia Euclidiana, corresponde a la raíz cuadrada de la suma de las diferencia cuadrática entre los atributos de los elementos comparados.

$$Euclidiana(x, y) = \sqrt{\sum_i^p (x_i - y_i)^2}. \quad (1.3)$$

- **Distancia de Minkowski:** Corresponde a una generalización de la Distancia Euclidiana para cualquier parámetro r .

$$Minkowski(x, y) = \sqrt[r]{\sum_i^r (x_i - y_i)^r}. \quad (1.4)$$

- **Distancia de Hamming:** Asumiendo que los elementos se encuentran representados como secuencias discretas (códigos) de largo p , la distancia de Hamming corresponde

a la cantidad de atributos diferentes entre dos elementos. Se puede utilizar la operación disyunción exclusiva si los elementos son representados mediante atributos binarios.

$$Hamming(x, y) = \sum_i^p x_i \oplus y_i. \quad (1.5)$$

Alternativamente a la definición de una métrica o distancia, es posible definir una medida de similaridad entre los elementos de X . Por ejemplo,

- **Índice de Jaccard:** Asumiendo que los elementos se encuentran representados como conjuntos, indica el número de elementos comunes con respecto al número de elementos totales, es decir, define el coeficiente de similaridad entre un par de elementos según la proporción entre la conjunción y disyunción inclusiva del conjunto de etiquetas de ambos elementos.

$$Jaccard(x, y) = \frac{|x \cap y|}{|x \cup y|}. \quad (1.6)$$

- **Similitud Coseno:** Asumiendo que los elementos se encuentran representados vectorialmente, i.e., que $X \subset \mathbb{R}^p$, la similaridad coseno se define como

$$Cosine(x, y) = \frac{x^T y}{\|x\| \|y\|}. \quad (1.7)$$

En estos casos, es posible re-utilizar las definiciones de búsqueda por similitud descritas anteriormente, definiendo una medida de distancia inversamente proporcional a las medidas de similaridad. Por ejemplo

- **Distancia de Jaccard:** Indica la disimilitud entre conjuntos de datos en modo complementario al Índice de Jaccard.

$$DisJaccard(x, y) = 1 - Jaccard(x, y). \quad (1.8)$$

- **Distancia Coseno:** Indica la disimilitud entre elementos representados vectorialmente en modo complementario a la medida de similaridad coseno:

$$DisCosine(x, y) = 1 - Cosine(x, y) \quad (1.9)$$

1.1. Descripción del Problema

Este trabajo está enfocado en la búsqueda por similitud de documentos de texto para optimizar tareas de reconocimiento de patrones. Concretamente, dado un documento $q \in X$, nos gustaría obtener los documentos de una determinada colección que resultan *semánticamente similares* a q , es decir, que están relacionados con q en función de su significado o contenido. En esta tarea, asumiremos que $X \subset \mathcal{R}^p$, es decir, que los documentos se encuentran representados vectorialmente, aproximación que es sin duda la más frecuente hoy en día. Nuestro objetivo es recuperar los documentos que pertenecen a las mismas categorías, temas o tópicos que q . En este proceso buscamos ser eficientes y precisos.

1.1.0.1 Eficiencia

Es fácil observar, que resolver el problema de búsqueda por similitud (adoptando cualquiera de las definiciones enumeradas en la sección anterior) mediante “fuerza bruta”, es decir, recorriendo secuencialmente la base de datos D hasta encontrar los elementos similares a la consulta, tiene una complejidad computacional $\mathcal{O}(N)$, tanto en tiempo como en espacio (memoria). Para colecciones grandes, esta complejidad puede resultar prohibitiva, sobre todo si la búsqueda debe realizarse más de una vez, como es común en algoritmos de reconocimiento de patrones y clustering. Los métodos tradicionales basados en la construcción de estructuras de datos eficientes para esta tarea (KD trees, Ball trees, etc), suelen escalar deficientemente en términos del número de atributos p utilizados para representar los ítems [40]. Para el problema que se aborda en esta memoria p es del orden de los cientos de miles, millones e incluso más, por lo que, de no mediar, técnicas de reducción de dimensionalidad, los métodos tradicionales no resultan aplicables.

El problema fundamental abordado en esta memoria es entonces la búsqueda por similitud **eficiente** sobre colecciones documentales.

1.1.0.2 Precisión

Como se discutió en el capítulo anterior, la formalización del problema requiere la definición de una distancia o medida de similaridad entre textos, problema que ha sido ampliamente discutido en la literatura [6] [20]. Considerando que una de las medidas más utilizadas para comparar documentos en la práctica es la similaridad de coseno (definida en la sección anterior), dicha medida sería la opción natural a considerar en esta memoria. Sin embargo, un aspecto distintivo de las aplicaciones de este trabajo con respecto otros problemas de búsqueda por similaridad, es que la distancia coseno es típicamente sólo un “proxy” hacia la búsqueda de documentos que coincidan en términos de categorías, temas o tópicos, con un documento de consulta que se desea caracterizar (e.g. clasificar). Si se adopta esta medida, es un problema en sí mismo encontrar una representación adecuada para los documentos tal que la distancia coseno refleje la similitud entre ellos, de modo que sea posible identificar aquellos que pertenecen al mismo tipo.

Considerando este problema, en este trabajo se estudiará también la definición de una medida de similaridad que denominaremos *topic similarity* y que permita cuantificar el grado de *afinidad semántica* entre dos documentos en términos de tópicos. Para ello se asumirá que cada documento x está asociado a un conjunto de etiquetas $t(x)$, de modo que el grado de similaridad entre $x \in X$ e $y \in X$ se pueda medir comparando $t(x)$ y $t(y)$. En la práctica estas etiquetas existen porque las colecciones han sido explícitamente anotadas o podrían corresponder a los tópicos obtenidos después de procesar la colección de documentos mediante una técnica de análisis de documentos. Se discutirá que la noción de similaridad adoptada da como resultado una pseudo-métrica en vez de una métrica, pero la definición de búsqueda por similitud puede adaptarse fácilmente a este caso.

1.2. Estrategia de Solución Adoptada

Una técnica introducida recientemente para reducir costos de una búsqueda por similitud es *similarity preserving hashing* [40]. En general, una técnica de *hashing* permite crear

una representación de los elementos a modo de índice. Así, para buscar un elemento en particular dentro de un dataset no es necesario revisar todos los elementos candidatos sino que basta utilizar la función mediante la cual se creó la representación del documento y acceder a ella de forma directa. Lamentablemente, los métodos tradicionales de *hashing* no permiten crear una representación que refleje la similitud entre los elementos, por ende el índice no permite realizar una recuperación directa de dichos elementos para una consulta q_i y será necesario revisar el dataset por completo. Es por esto que, recientemente, se han propuesto técnicas de hashing capaces de generar representaciones similares para elementos similares. Nos referiremos a estas técnicas como *similarity preserving hashing* o simplemente *hashing*, cuando esto no cause confusión.

Puesto que es difícil generar un método único para la representación de datos que refleje la similitud se opta por utilizar *ANN* para generar una *función de hashing* que garantice que documentos semánticamente similares, es decir, del mismo tipo, tópico o clase, sean mapeados al mismo índice o uno similar. La *ANN* corresponde una estructura inspirada en el sistema nervioso humano, el cuál está encargado de recibir, procesar e interpretar los diversos estímulos que el cuerpo recibe. Para esto, se cuenta con diversas capas de neuronas, las cuáles reciben el estímulo y, en caso de ser lo suficientemente fuerte, lo procesan y comparten a otras neuronas mediante un proceso llamado *sinapsis* hasta que la información sea recibida por el cerebro, se determine una respuesta y esta sea enviada por las neuronas para ser ejecutada. La *ANN* opera del mismo modo: mediante neuronas artificiales distribuidas en distintas capas se recibe, procesa y transmite la información recibida de modo que sea de utilidad para un fin a definir. Para esto, la *ANN* debe pasar previamente por un proceso de aprendizaje, donde recibe información del tipo que debe analizar, la procesa y realiza las correcciones necesarias a sus parámetros durante un cierto tiempo para reducir el error en su desempeño.

De este modo, es posible utilizar una *ANN* para determinar la *función de hashing* que genere una representación de cada documento que permita medir la similaridad semántica, así al ser utilizado como índice se pueda reducir el tiempo de búsqueda de $O(n)$, es decir, no sea necesario recorrer el dataset completo para encontrar los elementos más cercanos [28].

1.3. Objetivos

El objetivo de este trabajo es implementar y evaluar una *ANN* como método de *hashing* de modo que sea posible generar una representación que permita realizar *BSPD*. Para esto, será necesario:

1. Implementar computacionalmente un modelo de *ANN* cuyo objetivo de aprendizaje sea determinar una *función de hashing* para *BPSD*.
2. Evaluar el desempeño del modelo comparándolo con baseline bien conocidos en la literatura.
3. Evaluar la sensibilidad de los resultados obtenidos a la cantidad de capas de ocultas del modelo.
4. Evaluar la sensibilidad de los resultados obtenidos a la cantidad de neuronas en las capas ocultas.

1.4. Contribuciones de esta memoria

Este trabajo está inspirado en la técnica *semantic hashing* [28], en donde se define una *ANN* para generar una representación binaria de documentos de texto que refleje la similitud semántica entre dichos elementos. De este modo, es posible identificar y recuperar los elementos pertenecientes a una misma clase respecto a una consulta q comparando la similitud entre sus representaciones.

Para este trabajo, se implementó dicho modelo en Python, utilizando la librería *NumPy* y *Scipy*. A diferencia de la implementación en Matlab, esta nueva implementación permite modificar la profundidad de la red y el tamaño de las capas ocultas, lo cuál permite modificar el modelo a conveniencia. Además, se suprimió el ruido en la capa de salida pues no mejora el desempeño del modelo. Luego, se experimentó utilizando distintas arquitecturas de red, ya sea variando la cantidad de capas ocultas y la cantidad de neuronas por capa oculta, con

el objetivo de evaluar su comportamiento y definir qué modelo es más apropiado para esta tarea.

Capítulo 2

Estado del Arte

2.1. Hashing

La búsqueda por similitud requiere de métodos que permitan comparar los elementos de forma veloz y apropiada, para ello existen diversos métodos de búsqueda. Uno de ellos es utilizar métodos para transformar los elementos de modo que se obtenga una representación compacta y precisa de ellos. Esta técnica se conoce como *hashing* [14].

Para generar una representación adecuada de los elementos es necesario definir las operaciones que lo permitirán, esto se denomina función de *hashing*.

$$x \rightarrow hash(x). \quad (2.1)$$

La función de *hashing* tiene por objetivo transformar un elemento x desde su espacio original al espacio de Hamming, es decir, al espacio de secuencias binarias, un espacio que posee su propia métrica, la distancia de Hamming. Mediante una función de *hashing* es posible generar una tabla con secuencias binarias que representan los elementos, de modo que en un proceso de recuperación de un elemento x se utilice dicha clave para identificar con mayor facilidad dicho elemento, agilizando su recuperación. No obstante, es posible que para distintos elementos se genere la misma clave, esto se denomina colisión. El enfoque tradicional

de *hashing* busca evitar colisiones pues entorpecen la diferenciación de dicho elemento, sin embargo en la búsqueda por similitud la colisión permite identificar qué tan cercanos son los distintos elementos y así decidir cuales recuperar, siendo un factor importante y deseable.

2.1.1. Búsqueda mediante hashing

Para realizar búsqueda mediante *hashing* existen básicamente tres etapas: definir una función de *hashing*, generar la representación binaria de los elementos en la base de datos e indexar respecto a ellos y realizar consultas a la base de datos mediante esta representación.

2.1.1.1. Definir una función de hashing

Definir una función de *hashing* es un proceso complejo y no exento de problemas, ya que existen múltiples factores que determinan el modo en que se define la función y su comportamiento, por ejemplo: [40]

- **Métodos adaptativos versus No-adaptativos:** Una función de hashing se dice adaptativa si se define tomando en cuenta información acerca de la base de datos donde se aplicará. Una función de hashing no-adaptativa por el contrario, se define a-priori, usando típicamente proyecciones definidas sin tomar en cuenta información estadística de los elementos donde será aplicada. Si bien la calibración de técnicas adaptativas puede resultar más costosa, con frecuencia es posible obtener un mejor rendimiento.
- **Análisis No-Supervisado, Semi-Supervisado y Supervisado:** Se denomina análisis no-supervisado cuando se utilizan las propiedades de los elementos para determinar la función de *hashing* sin considerar las etiquetas que posean, por ejemplo, *spectral hashing* se basa en la estructura de los datos para generar su representación. En cambio, para el análisis supervisado se utilizan las etiquetas de los elementos para determinar la función que genere su representación [12]. En análisis semi-supervisado se utilizan ambos enfoques, es decir, se aprovechan las propiedades de los elementos para determinar la función y su resultado es guiado por sus etiquetas.

- **Función Lineal vs No Lineal:** Una función de hashing puede ser lineal o no lineal. Si se utiliza una función lineal, los atributos que describen un elemento serán tratados de modo completamente independiente, mientras que si se utiliza una función no-lineal es posible tener en cuenta dependencias entre los atributos de entrada. Si bien las primeras son menos costosas, los métodos no lineales podrían permitir un mejor desempeño.
- **Con o sin peso:** Se dice *hashing* con peso cuando se asigna distinta ponderación a los bits de la representación generada que refleje su relevancia. De este modo, se reduce la probabilidad de colisión para elementos no vecinos y se aumenta en elementos vecinos.

2.1.1.2. Indexar de la base de datos

Tras definir una función de *hashing*, se utiliza dicha función para transformar todos los elementos en la base de datos desde el espacio al que pertenecen al espacio de Hamming, esto reduce considerablemente la dimensionalidad.

Las representaciones son registradas en una tabla (llamada *hash map*), de modo que según esta tabla se pueden recuperar todos los elementos correspondientes a una secuencia de datos en tiempo constante. Entre menos bits se utilicen más elementos estarán asociados a una misma representación, por ende mediante una consulta se recuperarán más documentos.

2.1.1.3. Consultar a la base de datos

Siendo el objetivo recuperar los elementos más cercanos, las consultas son transformadas mediante la función de *hashing* aprendida con anterioridad y utilizada en la base de datos. Para realizar la búsqueda por similitud existen dos alternativas:

1. Medir la distancia de Hamming entre la representación de la consulta q y todos los elementos de la base de datos y luego recuperar los k -elementos más cercanos o todos los que se encuentren a una distancia igual o menor a r . Si bien se utiliza una operación menos costosa que comparar los elementos en su espacio original su complejidad es lineal respecto a la base de datos.

2. Medir la distancia de Hamming entre la representación de la consulta q y la *hash map*, recuperar las secuencias que se encuentren dentro de la esfera de Hamming de radio r y recuperar directamente de la base de datos los elementos que poseen dichas representaciones. Esto reduce considerablemente la complejidad en la búsqueda pues se recorre una tabla mucho más pequeña que la base de datos [39].

2.1.2. Aplicaciones

Mediante *hashing* se genera una representación de los elementos que refleja su similitud pero disminuyendo considerablemente la dimensionalidad al transformarlos en secuencias binarias, reduciendo el espacio de búsqueda y permitiendo utilizar operadores menos costosos para comparar entre elementos. De este modo, se puede utilizar para indexar una base de datos, así para recuperar los elementos similares a una consulta basta con generar la representación del elemento consultado y recuperar aquellos cuya representación sea más cercanas. De este modo, existen aplicaciones en múltiples áreas, por ejemplo:

- Búsqueda y recuperación de imágenes [9] [16]
- Reconocimiento facial [34] [22]
- Identificar posición y orientación de elementos [33] [31]
- Sistemas recomendadores [44] [32]
- Recuperación de documentos de texto [28] [35]
- Detección de duplicados y copia [18] [13]

2.2. Representación de Texto

Los documentos de texto no necesariamente poseen una estructura similar al compartir temas, lo cuál dificulta el cálculo de similitud pues:

- Se requieren métodos particulares para comparar documentos según la estructura que posean.
- Los símbolos, modismos, pronombres, artículos y cualquier palabra que no provea información significativa del documento de texto, llamadas *stop word*, entorpecen la diferenciación de documentos de texto [26].

Es por esto que se requiere dar una representación de los documentos que contenga solamente la información necesaria para un buen análisis y permita un desempeño más eficiente y computacionalmente factible sin importar el tamaño de los textos para el proceso de la búsqueda por similitud. Al momento de diseñar una representación para los textos es necesario definir el Vocabulario a utilizar, la estructura de la representación y su contenido. En [29] se describen distintos tipos de representación basadas en vectores indicando la ocurrencia de las palabras.

2.2.1. Vocabulario

El vocabulario corresponde al listado de palabras existentes en el corpus, es decir, en el conjunto de documentos. En [19] se indican diversos métodos para filtrar los documentos y así indicar el contenido del vocabulario.

- **Tokenización:** Las palabras son separadas entre sí, manteniendo solo una copia de cada una existente en el corpus. Se suele aprovechar de eliminar elementos de puntuación.
- **Eliminar *stop word*:** Se eliminan las palabras que no presentan significado por si mismas, como los artículos o pronombres, puesto que no contribuyen a la diferenciación de los documentos de texto.
- **Normalización:** Todas las palabras se pasan a minúsculas de modo que se descartan las palabras repetidas y se eliminan caracteres no deseados.

- **Stemming y Lemanización:** Se reducen las palabras a su raíz, de modo que palabras derivadas de una misma se consideran como igual sin importar su tiempo verbal o si corresponden al singular o plural.

Además, el vocabulario contiene sólo palabras que aparecen al menos 1 vez en algún texto del corpus y que sean pertinentes para un buen análisis.

2.2.2. Word Count Vector

El *word count vector* es una estructura a modo de vector utilizada en la representación de textos para facilitar la recuperación de los documentos. Su estructura corresponde a un vector donde cada casilla corresponde a una palabra del diccionario y su contenido depende del modo que se desee representar; puede ser del mismo tamaño que el diccionario o estar limitado a un subconjunto de este basado en su frecuencia en el corpus o su relevancia. Existen diversos criterios para fijar el contenido del vector, tanto según el tipo de dato como en la frecuencia de las palabras [30].

- **Binario:** En la representación binaria cada casilla acepta solo dos valores, 0 o 1. Se asigna 1 a una casilla cuando la palabra asociada se encuentra al menos una vez en el texto; en caso que no exista se le asigna 0.
- **TF:** La representación *TF* indica la frecuencia de cada término en el texto. Para ello cada casilla corresponde a una palabra del vocabulario y su valor corresponde a la ocurrencia de dicha palabra en el texto representado.

$$TF(t, d) = fr(t, d). \quad (2.2)$$

Una alternativa es indicar la frecuencia relativa de las palabras, es decir, el *TF* se normaliza dividiendo por el tamaño del texto representado. De este modo, se puede utilizar a modo de probabilidad de ocurrencia de las palabras en el texto.

$$TFN(t, d) = \frac{fr(t, d)}{|d|}. \quad (2.3)$$

- **TF-IDF**: La representación *TF-IDF* es una alternativa que busca eliminar la sobre representación de palabras de alta frecuencia en una representación *TF*. Para ello, se generan el vector *TF*, previamente descrito, y el vector *IDF*. Este último indica la relevancia de las palabras en el corpus según la proporción de textos en los que se puede encontrar. Matemáticamente corresponde al inverso del cociente entre la cantidad de documentos donde ocurre dicha palabra y el total de documentos. De este modo, el valor indicado en el *TF* se reduce de forma proporcional a los textos en que existe, así las palabras de alta frecuencia pero que existen en pocos textos del corpus ven su valor considerablemente reducido.

$$\text{IDF}(t, D) = \frac{|D|}{|d \in D : t \in d|}, \quad (2.4)$$

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D). \quad (2.5)$$

2.3. Machine Learning

Existen problemas donde diseñar un algoritmo que permita resolverlos se vuelve una tarea demasiado compleja. Producto de esto surge el interés por crear programas que mediante la experiencia puedan mejorar su desempeño al realizar una tarea, esto es la motivación de los algoritmos de aprendizaje en *machine learning*.

2.3.1. Algoritmo de aprendizaje

Se dice que un programa es capaz de aprender de la experiencia respecto a una tarea y una determinada medida de desempeño si es capaz de mejorar en la realización de dicha tarea a partir de la experiencia recibida [21]. Por ejemplo, para un programa de reconocimiento facial se tiene:

- Tarea: Reconocer y clasificar un rostro en base a una imagen.

- Desempeño: Cantidad de rostros clasificados de forma correcta
- Experiencia: Un conjunto de rostros previamente clasificados.

En *machine learning* se busca implementar un modelo que tenga un buen desempeño en la tarea asignada para elementos nuevos, es decir, que sea posible de generalizar a partir de la experiencia recibida. Por esto, a partir del conjunto de elementos que se poseen se generan dos subconjuntos independientes, el conjunto de entrenamiento y el conjunto de prueba. El conjunto de entrenamiento corresponde al subconjunto de elementos utilizados en el proceso de aprendizaje de un programa de modo que pueda desarrollar una tarea; el conjunto de prueba corresponde al subconjunto de elementos cuyo fin es corroborar el desempeño del programa al realizar la tarea asignada con elementos completamente nuevos, es decir, muestra el desempeño ante una generalización del problema.

Al medir el desempeño del programa según el conjunto de entrenamiento y de prueba se espera que el error del segundo sea igual o mayor al primero pues son elementos nuevos y desconocidos. Además, se desea:

- Reducir el error del conjunto de entrenamiento.
- Reducir la brecha entre el error del conjunto de entrenamiento con el error respecto del conjunto de pruebas.

No lograr disminuir el error del conjunto de entrenamiento implica que el modelo no está aprendiendo la tarea que se espera pueda realizar, esto se conoce como *underfitting*. En cambio, una brecha muy grande entre el error del conjunto de entrenamiento con el error del conjunto de pruebas significa que no fue posible generalizar el problema de modo que sea posible resolver la tarea para situaciones nuevas, esto se conoce como *overfitting*. Para evitar esto se debe definir una representación adecuada del problema, ya sea con la que ingrese, como se represente de forma interna o en el resultado que entrega.

2.3.2. Aplicaciones

Existen múltiples tipos de problemas en la informática donde es necesario *Machine Learning*, algunos de ellos son [8]:

- **Reconocimiento:** A partir de un conjunto de elementos, determinar los patrones que lo identifican y recuperar aquellos que cumplen con los patrones deseados. Por ejemplo, es posible determinar si una situación financiera presenta características similares a los de una crisis pasada.
- **Clasificación:** Para esta labor se desea determinar a qué categoría corresponde un elemento dado, para ello se define una función que genere una representación del elemento que refleje a cuál pertenece. Un ejemplo de esto es el reconocimiento de objetos, en donde ingresan imágenes, pre-procesadas para ser ingresadas al modelo, y se genera una secuencia numérica que identifica al elemento.
- **Regresión:** Se desea predecir un valor numérico dado un elemento considerando distintos factores. Por ejemplo, predecir el precio de una propiedad en función de un conjunto de características reunidas con anterioridad, como crimen del sector, proporción estudiante-profesor, entre otros.
- **Transcripción:** Se desea observar elementos relativamente no estructurado para transcribirlos en texto. Por ejemplo, mediante una imagen que contenga palabras, identificar y entregar dichas palabras como texto. También es visible posible en reconocimiento de voz, generar texto en función de un sonido emitido.
- **Optimización:** Existen modelos basados en minimizar una función de energía, esto se puede utilizar para optimizar procesos que de forma independiente serían costosos.

2.4. Red Neuronal Artificial

El sistema nervioso es una red de órganos y células que tiene como objetivo captar la información del medio, procesarla y generar una respuesta que será transmitida al resto

del organismo. Para esto, el sistema nervioso se encuentra formado por capas de neuronas, las cuáles son células conductoras que, tras ser suficientemente estimuladas, transmiten la información recibida a la siguiente capa de neuronas hasta ser recibida por el cerebro, ser procesada y se genera una respuesta.

Inspirado en el sistema nervioso humano surge el concepto de red neuronal (en adelante *ANN*). Este modelo consiste en emular el funcionamiento del sistema nervioso de modo que al recibir cierto tipo de dato se genere una respuesta adecuada para ser utilizada de un modo deseado. Del mismo modo, se definen neuronas artificiales, las cuáles corresponden a nodos interconectados entre sí encargados de recibir información y darle una interpretación, para luego compartirla con las neuronas con las que se encuentre conectada. Las neuronas se organizan en capas, de modo que la información suele ser compartida de capa en capa, sin embargo el tipo de conexión entre neuronas puede variar según el tipo de modelo en que se base la red.

Es posible separar una red neuronal en 3 componentes según el trato dado a la información: la información que ingresa es almacenada en la capa de entrada o capa visible, es procesada en las capas intermedias, llamadas capas ocultas y entrega el resultado a través de la capa de salida.

Para procesar la información se definen 2 funciones: función de pre-activación y la función de activación:

- **Función de pre-activación:** En esta función se combina el elemento de entrada con los pesos que indica la relación entre las neuronas que reciben el elemento de entrada y la neurona j que busca ser activada. Un método común es sumar los estímulos recibidos:

$$a(x)_j = b_j + \sum_i x_i w_{ij} = W^T x + b_j. \quad (2.6)$$

- **Función de activación:** Es la función que permite determinar la activación de una neurona tras recibir los estímulos de las neuronas a las que se encuentre conectada. Para ello existen diversos métodos:

- *Lineal:* La función lineal entrega como resultado el mismo elemento de entrada.

Reutilizando la ecuación 2.6 se tiene:

$$h(x) = g(a(x)_j) = W^T x + b_j. \quad (2.7)$$

Suele ser utilizada en la capa de salida y para problemas de regresión.

- *Sigmoid*: Otra función muy utilizada es la función sigmoid, la cuál permite que la activación de la neurona se encuentre limitada entre 0 y 1, sea siempre positiva y aumente mientras la función de pre-activación aumente

$$h(x) = \sigma(a(x)_j) = \frac{1}{1 + \exp(-a(x)_j)}. \quad (2.8)$$

- *Tangente hiperbólica*: La función tangente hiperbólicaS (en adelante función tanh) permite que la activación de la neurona se encuentre limitada entre -1 y 1, puede tomar valores positivos o negativos y aumenta mientras la función de pre-activación aumente

$$h(x) = \tanh(a(x)_j) = \frac{\exp(2a(x)_j) - 1}{\exp(2a(x)_j) + 1}. \quad (2.9)$$

Posterior a la función de activación es posible fijar una condición adicional a la activación de la neurona, por ejemplo, mediante una *sigmoid* es posible utilizar el resultado como una probabilidad de activación, de modo que al superar un valor mínimo la neurona receptora del estímulo posea valor 1; caso contrario, 0

$$h(x) = \begin{cases} 1, & a(x)_j \geq k. \\ 0, & a(x)_j < k. \end{cases} \quad (2.10)$$

2.4.1. Entrenamiento

Para que una *ANN* pueda ser utilizada de forma apropiada debe pasar por un proceso de entrenamiento, esto es, el proceso de aprendizaje donde se calibran los pesos y sesgo de la red

de modo que genere una representación adecuada de los elementos que ingresarán y refleje el objetivo deseado. El algoritmo básico de entrenamiento de *ANN* es *backpropagation*.

Backpropagation es un algoritmo que busca modificar los pesos de la red de modo que mejore su desempeño. Para esto se define una función de error para medir la diferencia entre el resultado obtenido y el resultado deseado, la cuál busca ser minimizada en cada ejecución del algoritmo:

1. Se calcula el gradiente de los errores según la capa k

$$\nabla_{a^k(x)} l(f(x), z). \quad (2.11)$$

2. Se calcula el gradiente para los pesos de la capa k

$$\nabla_{W^k} l(f(x), z) = \nabla_{a^k(x)} l(f(x), z) h^{k-1}(x)^T. \quad (2.12)$$

3. Se calcula el gradiente para los sesgos de la capa k

$$\nabla_{b^k} l(f(x), z) = \nabla_{a^k(x)} l(f(x), z). \quad (2.13)$$

4. Se calcula el gradiente para la capa k-1

$$\nabla_{h^{k-1}(x)} l(f(x), z) = (W^k)^T \nabla_{a^k(x)} l(f(x), z). \quad (2.14)$$

5. Se calcula el gradiente para la capa k-1 antes de ser activada

$$\nabla_{a^{k-1}(x)} l(f(x), z) = \nabla_{h^{k-1}(x)} l(f(x), z) \odot h'(k(x)). \quad (2.15)$$

6. Se repite desde el paso 2 utilizando el gradiente medido en el paso 5 hasta llegar a la primera capa.

Finalmente, cada gradiente de peso y sesgo indica la corrección que necesitan estos parámetros para reducir el error y, con ello, mejorar su desempeño. Se suele repetir una cantidad de veces (epochs) o hasta alcanzar un error aceptable, previamente definido.

Para mejorar el desempeño del algoritmo existen distintos métodos, como el uso de *mini-batch*, *momentum* y *weight decay*:

- **Mini-batch:** Se dice *mini-batch* cuando el total de elementos se divide en subconjuntos, de modo que en vez de usar el error según un ejemplo se promedia el error de todos los elementos del *mini-batch*, lo cuál permite encontrar un gradiente más acertado.
- **Momentum:** El *momentum* corresponde a añadir una fracción de la actualización de los pesos de la capa recientemente modificada en la capa actual, de modo que se evite converger en óptimos locales. Un valor muy alto puede evitar que evite la convergencia, en cambio un valor muy pequeño no evitará los óptimos locales.
- **Weight decay:** El *weight decay* corresponde a penalizar los pesos, de modo que mantengan un valor pequeño, evitando sobre entrenamiento.

2.4.2. Restricted Boltzman Machine

Una *RBM* es una *ANN* no dirigida en donde todas las neuronas de una capa interactúan con todas las neuronas de otras capas y no interactúan entre neuronas pertenecientes a la misma capa, es decir, se permite solamente la interacción inter-capa pero no intra-capa. Es una red estocástica, es decir, las neuronas son representadas mediante una distribución de probabilidad, de modo que al calcular su función de activación se determina la probabilidad que cada neurona se vea activada tras recibir los estímulos de otras neuronas.

Las *RBM* son modelos basados en energía (*EBM*), es decir, se asocia una energía al estado de las neuronas neuronas visibles y ocultas, la cuál busca ser modificada según el objetivo del aprendizaje [37],

$$E(v, h) = - \sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{oculta}} b_j h_j - \sum_{i, j} v_i w_{ij} h_j, \quad (2.16)$$

en donde v_i y h_j son un valor binario que indican si la neurona i y la neurona j están activas, a_i y b_j son el sesgo de la neurona i y de la neurona j , respectivamente; w_{ij} corresponde al peso de la relación entre la neurona i y j . Mediante esta energía, se define la distribución de

probabilidad de la capa v :

$$P(v, h) = \frac{e^{-E(v, h)}}{\sum_{v, h} e^{-E(v, h)}}. \quad (2.17)$$

Puesto que las neuronas de cada capa en una *RBM* indican su probabilidad de activación, se utiliza la *sigmoid* como función de activación

$$P(h_j = 1, v) = \sigma \left(\sum_i W_{ij} v_i + b_j \right), \quad (2.18)$$

$$P(v_i = 1, h) = \sigma \left(\sum_j W_{ji} h_j + a_i \right). \quad (2.19)$$

2.4.2.1. Entrenamiento

El entrenamiento de una *RBM* esta basado en el algoritmo *contrastive divergence (CD)*, el cual busca generar una aproximación del gradiente de la energía, de modo que es posible modificar los parámetros de la red sin necesidad de calcularla. Puede ser ejecutada hasta la convergencia, sin embargo, según el objetivo deseado es posible obtener un buen ajuste de parámetros iterando este método sin iterar demasiado [11]. El entrenamiento se puede dividir en dos fases: positiva y negativa.

1. Fase positiva

- Dada la capa v , se genera la distribución de la capa h según $P(h|v)$.
- Mediante la función de activación se determina el estado de cada neurona h_j , así determinar qué neurona se activa.

2. Fase negativa

- Dada la distribución h generada en la fase positiva se reconstruir la capa v según $P(v|h)$. Esta reconstrucción se denomina v' .

- Mediante v' , generar nuevamente la distribución, de h' .

Luego, el peso y sesgo de las neuronas se modifica mediante:

$$\delta w_{ij} = \epsilon(vh - v'h'), \quad (2.20)$$

$$\delta a_i = \epsilon(v - v'), \quad (2.21)$$

$$\delta b_j = \epsilon(h - h'), \quad (2.22)$$

en donde ϵ es la tasa de aprendizaje (en adelante *learning rate*), un factor que permite guía la convergencia del entrenamiento.

2.4.2.2. Constrained Poisson Model

CPM es una variante de las *RBM*. En este caso se utilizan elementos enteros en la capa visible, de modo que en base a ellos se busca la distribución de probabilidad de la capa oculta, por ello para la reconstrucción de la capa visible en la fase negativa de *CD* se utiliza una distribución de poisson [28]

$$P(v_i = n, h) = PS\left(n, \frac{e^{a_i + \sum_j h_j w_{ij}}}{\sum_k e^{a_k + \sum_j h_j w_{kj}}} N\right), \quad (2.23)$$

$$P(h_j = 1, v) = \sigma\left(\sum_i W_{ij} v_i + b_j\right), \quad (2.24)$$

en donde $PS(n, \lambda)$ corresponde a la Distribución de Poisson, w_{ij} es la relación entre las neuronas i y j , N es el tamaño original del elemento, a_i y b_i son el sesgo de la capa v y h .

La normalización de este modelo es significativa pues mantiene la estabilidad del aprendizaje, de modo que es efectiva para elementos de distintos tamaños [28].

2.4.3. Autoencoders

Los Autoencoders (*AE*) son redes neuronales feed-forward de aprendizaje no supervisado cuyo objetivo es aprender la estructura de un cierto tipo de elemento, ya sea generando una representación de menor dimensionalidad o una sobre representación donde se aprendan los atributos que caracterizan los patrones de entrada. Además, mediante esta representación es posible reconstruir el elemento reconstruido. Se conoce como *encoder* al proceso de generar la representación; *decoder* corresponde al proceso de reconstruir el original a través de su representación:

$$\begin{aligned}\text{Encoder} \quad h(x) &= g(a(x)) = \sigma(b + Wx), \\ \text{Decoder} \quad x' &= g(a'(x)) = \sigma(a + W^T h(x)).\end{aligned}$$

En donde x es el dato de entrada, b el sesgo de la capa en el proceso de generar la representación, $h(x)$ corresponde a la representación tras la transformación, a es el sesgo de la capa de reconstrucción y x' corresponde a la reconstrucción de x .

2.4.3.1. Entrenamiento

Para entrenar *AE* se utiliza el gradiente de su función de error, la cuál mide la diferencia entre el input original y su reconstrucción. Luego, mediante el algoritmo *backpropagation* estándar se generan los gradientes de todos los parámetros de la red, desde la capa de salida a la de entrada, y se utilizan para ajustarlos en busca de mejorar el desempeño de la red.

2.4.3.2. Variantes

- **Stacked Autoencoders:** Corresponde a un método que apila varios *AE*, de modo que el elemento de salida de un *AE* se vuelve la entrada del *AE* siguiente. Este modelo

permite utilizar redes profundas de modo que cada *AE* aprenda distintas características del elemento. Por ejemplo, para una imagen el primer *AE* podría comprender los elementos que se encuentran en los bordes, el segundo la forma de estos, y así sucesivamente.

- **Denoising Autoencoders:** Para operar con elementos incompletos o destruidos la definición previa de *AE* no es suficiente. En este caso, para el entrenamiento se distorsionan los elementos a ingresar a la red pero se mide el error según la reconstrucción y el elemento sin corromper. De este modo es posible detectar la estructura de los elementos y utilizar su representación de forma adecuada en procesos de reconstrucción y clasificación supervisada [38].

2.4.4. Deep Belief Network

Una *DBN* es una Red Neuronal profunda, es decir, posee múltiples capas ocultas con el objetivo de encontrar una mejor representación de los elementos; del mismo modo que en la *RBM*, las neuronas corresponden a elementos estocásticos.

El entrenamiento de la *DBN* consta de dos etapas: pre-entrenamiento y fine-tuning.

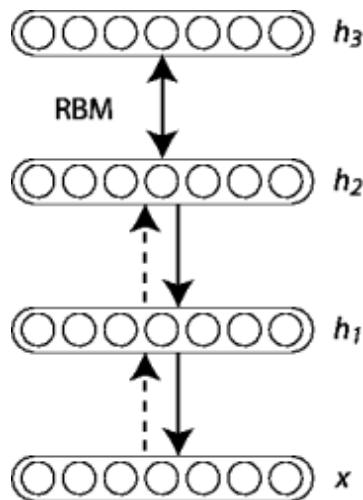


Figura 2.1: DBN [1]

2.4.4.1. Pre-Entrenamiento

El pre-entrenamiento consta de entrenar las capas de a pares utilizando *CD* del mismo modo que en la *RBM*, por esto se dice que las *DBN* corresponden a *RBM* apiladas [11].

1. Para un elemento dado, utilizar el método de entrenamiento de las *RBM* para el primer par de capas.
2. Con los parámetros asignados, obtener la transformación del elemento de entrada.
3. Repetir paso 1 utilizando la segunda capa anteriormente usada y la que le sigue, además transformación del elemento de entrada obtenida en 2 se utiliza como entrada en este nuevo par de capas. Esto se repite con todas las capas siguientes.

Si bien es posible generar una buena calibración de los datos para los pares de capas esto no se refleja en una red profunda, es decir, las capas que no estén conectadas directamente no muestran relación alguna, por ende el desempeño de la red completa no será significativamente mejor.

2.4.4.2. Fine-tuning

Como se indicó con anterioridad, esta configuración inicial permite un muy buen desempeño para las capas de forma independiente, no así con el desempeño de la red completa, la cuál no se ve mayormente beneficiada. No obstante sí permite determinar unos buenos parámetros iniciales para realizar *backpropagation* de modo que sean una guía inicial hacia la convergencia. De este modo, el desempeño de la red mejorará.

2.4.4.3. Semantic Hashing

Corresponde a una método de recuperación de documentos de texto en donde se genera una representación compacta de ellos que refleje su similitud a nivel semántico, de modo que

para una consulta sea posible recuperar los documentos similares según la cercanía de sus representaciones.

Este trabajo está basado en [28], donde se modeló una *ANN* que recibe documento procesado como *TF* y genera una representación binaria. El entrenamiento corresponde al de una *DBN*, con la diferencia que al recibir números enteros en su capa de entrada es necesario que el primer par de capas se entrene con el método descrito para la *CPM*:

$$P(v_i = n|h) = PS \left(n, \frac{e^{\lambda_i + \sum_j h_j w_{ij}}}{\sum_k e^{\lambda_k + \sum_j h_j w_{kj}}} \right), \quad (2.25)$$

$$P(h_j = 1|v) = \sigma \left(b_j + \sum_i w_{ij} v_i \right). \quad (2.26)$$

Para el resto de los pares de capas se utiliza el método de entrenamiento descrito para las *RBM*:

$$P(h_j = 1|v) = \sigma \left(b_j + \sum_i w_{ij} v_i \right), \quad (2.27)$$

$$P(v_i = 1|h) = \sigma \left(b_i + \sum_j w_{ij} h_j \right). \quad (2.28)$$

Para el *fine-tuning*, se realiza *backpropagation* con gradiente conjugado evaluando el error entre la normalización del *TF* y su reconstrucción para corregir los parámetros de la red, de modo similar al *AE* tradicional,

$$\nabla_{a(v)} l(v', v) = v' - v, \quad (2.29)$$

Este método ha mostrado ser una solución eficiente y efectiva en la recuperación de documentos de texto pues es posible realizar consultas rápidas sin aumentar de forma dramática el almacenamiento necesario, además puede ser utilizado tanto en aprendizaje supervisado como no supervisado [42] [43] [10] [24].

2.5. Tipos de Dataset

Los datasets a utilizar se dividen en dos tipos basados en el etiquetado, estos pueden ser *single label* y *multi label*.

- **Single Label Dataset:** Se conocen como *single label dataset* aquellos en donde todos los documentos poseen solamente una etiqueta, de este modo es posible distinguir con facilidad las distintas clases en base a la etiqueta que poseen. *20 Newsgroups* [3] corresponde a un dataset de noticias que poseen sólo una etiqueta.
- **Multi Label Dataset :** Se conocen como *multi label dataset* aquellos en donde los documentos poseen una o más etiquetas a la vez, de este modo distinguir la clase a la que corresponde un documento involucra analizar el conjunto de etiquetas que posee. En el mundo real es fácil encontrar ejemplos de este estilo siendo necesario identificar cuales identifican de forma más acertada clase a la cuál pertenecen, sin embargo esto es más complejo que para el tipo anterior, por ello se requieren métricas distintas basadas en el conjunto de etiquetas. Reuters [2] corresponde a un dataset donde cada documento presenta al menos una etiqueta.

2.5.1. Métricas de Resultados

Para determinar si los documentos recuperados corresponden a la consulta realizada es necesario evaluar la calidad de los resultados de la *ANN*. Para ello, se utilizarán técnicas de recuperación de información que permitan medir la precisión y recall.

2.5.1.1. Coincidencia de Etiquetas

En el caso de un *single label dataset*, se utiliza el criterio *coincidencia de etiquetas* para medir la precisión y recall, en donde basta comparar la etiqueta que posee cada documento para determinar si el documento recuperado coincide con la consulta. Esto es, sea q_i una consulta, $D = \{d_1, \dots, d_n\}$ los elementos del dataset y $c(d)$ la etiqueta del documento d , se

definirá $TS_i = \{d \in D : c(q_i) = c(d)\}$ el conjunto de elementos que poseen la misma etiqueta que q_i ; además, se define $S_i = \{d \in D : Hamming(q_i, d) < r\}$ el conjunto de elementos recuperados mediante q_i cuya distancia de Hamming $Hamming()$ sea menor a un radio r .

Se define p_i como la precisión de la consulta q_i

$$p_i = \frac{|S_i \cap TS_i|}{|S_i|}, \quad (2.30)$$

luego, se define r_i como el recall de la consulta q_i

$$r_i = \frac{|S_i \cap TS_i|}{|TS_i|}. \quad (2.31)$$

Para identificar la precisión y recall del conjunto de pruebas, P y R , se promedia p_i y r_i respecto al total de consultas realizado.

$$P = \frac{\sum_i p_i}{n}, \quad (2.32)$$

$$R = \frac{\sum_i r_i}{n}. \quad (2.33)$$

En un *multi label dataset* existen distintos métodos para corroborar la similitud entre los documentos; ya sea definiendo la similitud entre documentos mientras se comparta al menos una etiqueta o medir la fracción de coincidencia de etiquetas, también se puede medir la similitud por etiqueta y luego promediar o medir directamente sin hacer dicha separación.

2.5.1.2. Coincidencia Mínima de Etiquetas

En [17], [41], se propone considerar como elementos similares a una consulta aquellos que comparten al menos una etiqueta con la consulta, es decir, $TM_i = \{d \in D : c(q_i) \cap c(d) \neq \emptyset\}$, manteniendo la notación anterior. De este modo, se mantiene la definición de precisión y recall de *coincidencia de etiquetas* mencionada con anterioridad.

2.5.1.3. Jaccard Score

El método anteriormente descrito determina la similitud entre documentos sin importar la cantidad de etiquetas en común y dispares, de modo que documentos que no sean realmente relevantes a la consulta realizada pueden ser considerados correctos. Una técnica para evaluar un clasificador considerando el caso de múltiples etiquetas es utilizar la *similitud de jaccard* entre el conjunto de etiquetas de la consulta con las etiquetas del documento recuperado [40]. Esto es, sea $s_k \in S_i$ un documento recuperado mediante q_i :

$$\text{Jaccard}(q_i, s_k) = \frac{|c(q_i) \cap c(s_k)|}{|c(q_i) \cup c(s_k)|}, \quad (2.34)$$

luego, se define *jaccard score* de la consulta q_i como el promedio de $\text{Jaccard}(q_i, s_k)$ respecto al total de elementos recuperados y la global respecto a las consultas realizadas

$$\text{SJ}_i = \frac{\sum_k \text{Jaccard}(q_i, s_k)}{|S_i|}, \quad (2.35)$$

$$\text{SJ}_i = \frac{\sum_i \text{SJ}_i}{n}. \quad (2.36)$$

2.5.1.4. Criterio predictivo con promedio Macro

El *criterio predictivo con promedio macro* corresponde al análisis de precision y recall de forma independiente de cada etiqueta, es decir, se evaluará cada etiqueta t_j de la query q_i , en donde, $t_j \in c(q_i)$. Para ello se definen los verdaderos positivos, falsos positivos y falsos negativos.

- *Verdaderos Positivos* Corresponde a un contador de documentos recuperados para una etiqueta t_j en donde ésta coincide tanto en la consulta como en el documento recuperado, es decir, $t_j \in c(q_i)$ y $t_j \in c(s_k)$. Será definido como n_{j++} .
- *Falsos Positivos*: Corresponde a un contador de documentos recuperados para una etiqueta t_j en donde ésta se encuentra en el documento recuperado pero no en la consulta,

es decir, $t_j \in c(s_k)$ y $t_j \notin c(q_k)$. Será definido como n_{j+-} .

- *Falsos Negativos*: Corresponde a un contador de documentos recuperados donde la etiqueta t_j se encuentra en la consulta pero no en el documento recuperado, es decir, $t_j \in c(q_i)$ y $t_j \notin c(s_k)$. Será definido como n_{j-+} .

Mediante estos contadores se define la precisión y recall de etiqueta de la consulta:

$$p_j = \frac{n_{j++}}{n_{j++} + n_{j+-}}, \quad (2.37)$$

$$r_j = \frac{n_{j++}}{n_{j++} + n_{j-+}}, \quad (2.38)$$

se define la Precisión y Recall de la consulta mediante el promedio de la Precisión y Recall de sus etiquetas.

$$p_i = \frac{\sum_j p_j}{|c(q_i)|}, \quad (2.39)$$

$$r_i = \frac{\sum_j r_j}{|c(q_i)|}. \quad (2.40)$$

Finalmente, la evaluación global corresponde a promediar las métricas respecto al total de consultas realizadas

$$P = \frac{\sum_i p_i}{n}, \quad (2.41)$$

$$R = \frac{\sum_i r_i}{n}. \quad (2.42)$$

2.5.1.5. Criterio predictivo con promedio Micro

El *criterio predictivo con promedio micro* es similar al anterior, con la diferencia que evalúa la precision y recall de las consultas en conjunto, es decir, se ignora la división por clase para los contadores de verdaderos positivos, falsos positivos y falsos negativos.

- *Verdaderos Positivos* Corresponde a un contador de documentos recuperados para una etiqueta t_j en donde ésta coincide tanto en la consulta como en el documento recuperado, es decir, $|c(q_i) \cap c(s_k)|$. Será definido como n_{j++} .
- *Falsos Positivos*: Corresponde a un contador de documentos recuperados para una etiqueta t_j en donde ésta se encuentra en el documento recuperado pero no en la consulta, es decir, $|c(s_k) - c(q_i)|$. Será definido como n_{j+-} .
- *Falsos Negativos*: Corresponde a un contador de documentos recuperados donde la etiqueta t_j se encuentra en la consulta pero no en el documento recuperado, es decir, $|c(q_i) - c(s_k)|$. Será definido como n_{j-+} .

Mediante estos contadores se define la precisión y recall de cada consulta

$$p_i = \frac{n_{++}}{n_{++} + n_{+-}}, \quad (2.43)$$

$$r_i = \frac{n_{++}}{n_{++} + n_{-+}}. \quad (2.44)$$

Finalmente, la evaluación global corresponde a promediar las métricas respecto al total de consultas realizadas

$$P = \frac{\sum_i p_i}{n}, \quad (2.45)$$

$$R = \frac{\sum_i r_i}{n}. \quad (2.46)$$

Mediante la precisión es posible identificar el porcentaje de elementos correctamente recuperados respecto a la cantidad de recuperados con la consulta; en cambio, recall indica el porcentaje de elementos correctamente recuperados respecto a la cantidad de elementos deseados.

Capítulo 3

Propuesta e implementación de solución

3.1. Red Neuronal

El objetivo es implementar una *ANN* que transforme la representación *TF* con valores enteros de los documentos en una representación binaria que refleje su similitud semántica, para esto es necesario definir la arquitectura de la red y su método de entrenamiento. Se construyeron dos tipos de redes: La primera corresponde a un *CPM*, el cuál es una variante de la *RBM*; la segunda corresponde a una *DBN*.

La implementación de la *DBN* fue basandose en la implementación en *Matlab* realizada por *Ruslan Salakhutdinov* y *Geoff Hinton* para resolver el problema tradicional de MNIST [27]. Dicha implementación contiene ciertas diferencias respecto a la que se implementará para este trabajo:

- Si bien permite cambiar la cantidad de neuronas, para cambiar la cantidad de capas ocultas se debe reescribir gran parte del código.
- La capa de entrada recibe elementos binarios, por ende no requiere realizar el *CPM* para trabajar con enteros.
- Utiliza una función de activación lineal para la capa de salida, en cambio el modelo

propuesto utiliza la función *sigmoid* de modo que la representación posea valores entre 0 y 1.

Sin embargo, el algoritmo de pre-entrenamiento y fine-tuning corresponden al modelo original. Además, posee la implementación de Carl Edward Rasmussen para realizar el método del gradiente conjugado con linesearch [23].

Como se mencionó con anterioridad, para este trabajo se implementó la *DBN* en *Python* con *NumPy* realizando las modificaciones necesarias y con la flexibilidad para fijar la cantidad de capas ocultas y la cantidad de neuronas que se requiera.

Antes de iniciar el entrenamiento es necesario asignar el tamaño de los *mini-batches*, el *learning rate* para los pesos y sesgos de la capa de entrada y salida, el tamaño de la capa de salida, el *momentum*, *weight decay* y la cantidad de *epoch* a realizar. Además, se inicializa la matriz de pesos con valores aleatorios en una distribución normal con media 0 y varianza 0.01, los sesgos de las neuronas de ambas capas se inicializan en 0.

3.1.1. CPM

Los *TF* de los documentos de entrenamiento son separados en *mini-batches*, de modo que tras cada *mini-batches* se corrige la matriz de pesos y los sesgos de las neuronas. A continuación, se expone el método *CD* con cada minibatch para entrenar la red, esto se repite según la cantidad de *epochs* asignados:

1. Fase Positiva

- Ingresa una matriz en la capa visible, donde las filas corresponden a distintos documentos y las columnas indican la frecuencia de las palabras.
- Se modela la distribución de probabilidad de la capa oculta mediante la función *sigmoid* a partir de la matriz de *TF*, la matriz de pesos y el sesgo de las neuronas de la capa oculta.
- Para cada neurona se suma su probabilidad de activación.

- Se determina la activación de la capa oculta.

```

1 num_v = Cantidad neuronas visibles
2 num_h = Cantidad neuronas ocultas
3 pos_h= expit(data*vishid + hidbiases)
4 batch_h = pos_h
5 pos_prod = transpose(data)*pos_h
6 pos_dist_h = sum(pos_h, axis=0)
7 pos_dist_v = sum(minibatch, axis=0)
8 pos_act = pos_h > random

```

2. Fase Negativa

- Se reconstruye la distribución de la capa visible mediante el *CPM*.
- Se modela la distribución de probabilidad de la capa oculta mediante la función *sigmoid* a partir de la reconstrucción de los *TF*, la matriz de pesos y el sesgo de las neuronas de la capa oculta.
- Para cada neurona se suma su probabilidad de activación.

```

1 size_doc = sum(data,axis=1)
2 lambda_=size_doc*softmax(pos_act*transpose(vishid) + visbiases))
3 neg_data = lambda_
4 neg_h= expit(neg_data*vishid + hidbiases)
5 neg_prod = transpose(neg_data)*neg_h
6 neg_dist_h = sum(neg_h, axis=0)
7 neg_dist_v = sum(neg_data, axis=0)

```

3. Actualizar Pesos y sesgo

- Se mide la variación de los pesos y sesgos mediante el algoritmo explicado en el capítulo anterior.

```

1 vishid_inc = lr_w*((pos_prod-neg_prod)/num_cases
2               - weightcost*vishid) + momentum*vishid_inc
3 visbiases_inc = (lr_v/num_cases)*(pos_dist_v-neg_dist_v)

```

```

4         + momentum*visbiases_inc
5 hidbiases_inc = (lr_h/num_cases)*(pos_dist_h-neg_dist_h)
6         + momentum*hidbiases_inc
7 vishid = vishid + vishid_inc
8 visbiases= visbiases + visbiases_inc
9 hidbiases = hidbiases + hidbiases_inc

```

3.1.2. DBN

Puesto que este modelo posee capas ocultas se debe indicar, además de los parámetros ya mencionados, la cantidad de neuronas para cada capa oculta. El entrenamiento se divide en dos etapas: pre-entrenamiento, donde se busca una buena inicialización de los parámetros trabajando con pares de capas, y *fine-tuning*, donde se corrigen los pesos de la red completa en función al error obtenido y al gradiente de los pesos.

3.1.2.1. Pre-entrenamiento

Como se mencionó, en el pre-entrenamiento se busca una buena inicialización pesos y sesgos trabajando con pares de capas desde la capa de entrada hasta la capa de salida de la ANN. Para este trabajo se utilizó *CPM* en el primer par de capas, para el resto de las capas se entrenaron como *RBM*.

1. *CPM*: Se utiliza del modo descrito en la sección 3.1.1 para actualizar los pesos y sesgos del primer par de capas.
2. *RBM*: La capa de entrada corresponde a la capa de salida del par de capas anterior, para ello recibe su distribución de probabilidad. Difiere de *CPM* en la función que utiliza para la reconstrucción de la distribución de probabilidad en la capa visible, no busca reconstruir elementos enteros sino que binarios.

```

1 num_v = Cantidad neuronas visibles
2 num_h = Cantidad neuronas ocultas
3 pos_h= expit(data*vishid + hidbiases)
4 batch_h = pos_h
5 pos_prod = transpose(data)*pos_h
6 pos_dist_h = sum(pos_h, axis=0)
7 pos_dist_v = sum(minibatch, axis=0)
8 pos_act = pos_h > random

```

```

1 neg_data = expit(pos_act*transpose(vishid) + visbiases))
2 neg_h= expit(neg_data*vishid + hidbiases)
3 neg_prod = transpose(neg_data)*neg_h
4 neg_dist_h = sum(neg_h, axis=0)
5 neg_dist_v = sum(neg_data, axis=0)

```

```

1 vishid_inc = lr_w*((pos_prod-neg_prod)/num_cases
2               - weightcost*vishid) + momentum*vishid_inc
3 visbiases_inc = (lr_v/num_cases)*(pos_dist_v-neg_dist_v)
4               + momentum*visbiases_inc
5 hidbiases_inc = (lr_h/num_cases)*(pos_dist_h-neg_dist_h)
6               + momentum*hidbiases_inc
7 vishid = vishid + vishid_inc
8 visbiases= visbiases + visbiases_inc
9 hidbiases = hidbiases + hidbiases_inc

```

3.1.2.2. Fine-tuning

El objetivo del pre-entrenamiento es encontrar los valores iniciales apropiados para los parámetros de la ANN, de modo que al realizar el fine-tuning los pesos sean corregidos de forma apropiada para mejorar el desempeño en la labor a realizar. El pre-entrenamiento por sí solo no considera la interacción entre todas las capas de la red. A continuación, se utiliza la técnica *backpropagation* para modificar todos los parámetros de la red y mejorar su rendimiento. Se utiliza *cross-entropy error* con *softmax* en la capa de salida como *loss*

function y el método de gradiente conjugado con 3 lineasearch para determinar la variación de los pesos, para esto se realiza:

1. Normalizar el *TF*, de modo que sea una de distribución de probabilidad.
2. Se desenrolla la red, de modo que se presenta como un *AE*.
3. Se utiliza un *softmax* en la capa de salida.
4. Cada *TF* normalizado recorre el *AE* para reconstruir el elemento.
5. Se mide el error de la red mediante *cross-entropy* según el *TF* original y su reconstrucción.

```
1 for w in W[-1]:
2     batches_prob.insert(0, batch)
3     batch = sigmoid(batch*w)
4 batch = softmax(batch*W[-1])
5 cross_entropy_softmax = (-sum(data*log(batch)))/num_cases
```

6. Se mide el gradiente de la *loss function* respecto los pesos, estos gradientes modificarán los parámetros de la red en busca de minimizar el error en la siguiente iteración. Esto se realiza desde la capa de salida del *AE* hasta la capa de entrada, sea *K* la capa actual:
 - a) Se obtiene el gradiente de la matriz de pesos *K* al multiplicar el gradiente de la capa de salida pre-activada con la distribución de probabilidad de la capa *K-1* traspuesta.
 - b) Se mide el gradiente de la capa *K-1* como el producto entre la matriz de pesos *K* traspuesta y el gradiente de la capa de salida.
 - c) Se mide el gradiente de la capa *K-1* pre-activada como el producto *wise-element* entre el gradiente de la capa *K-1* con la activación de la capa *K*.
 - d) Utilizando el gradiente de la capa *K-1* en lugar del obtenido de la capa de salida, se repite desde el paso b en adelante hasta obtener el gradiente de cada matriz de pesos.

```

1 pre_act = (batch-data)/num_cases
2 for b_p, w in batches_prob, W:
3     dw = transpose(b_p)*pre_act
4     dW.insert(0,dw)
5     pre_act = (pre_act*transpose(w))*b_p*(1-b_p)
6 return dW

```

3.2. Dataset

La primera decisión a tomar para validar una *ANN* como solución a un problema es la elección de los dataset que se utilizarán durante la experimentación y separar sus datos entre los que se utilizarán para entrenar la red y los que se utilizarán para hacer pruebas. Es importante hacer esta separación de modo que:

- El conjunto de datos de entrenamiento sea representativo del dataset, de modo que todos los tipos de datos se encuentren en él. Así, durante el entrenamiento, la red recibirá la información necesaria para encontrar una configuración adecuada de sus parámetros y clasificar de forma adecuada la información futura.
- El conjunto de datos de pruebas no debe contener ningún dato de entrenamiento, así las métricas de rendimiento de la *ANN* corresponden a información nueva y son una muestra verdadera del desempeño de la red.

3.2.1. 20 Newsgroups

Corresponde a una colección de aproximadamente 20.000 documentos correspondientes a 20 tipos distintos de noticias. Se considera *single label dataset* pues cada documento posee solamente una etiqueta. Para este trabajo se utilizó la versión *20news-bydate* [3], la cuál posee los datos ordenados por fecha y se han eliminado previamente los documentos replicados, en total son 18.846 documentos divididos en 11.308 documentos para el entrenamiento de la red; los restantes, correspondiente a 7.538 documentos, se utilizan para probar

la red ya entrenada y realizar mediciones de precisión, recall y *jaccard score*.

3.2.2. Reuters

Corresponde a una colección de aproximadamente 810.000 documentos, los cuáles poseen al menos una etiqueta dentro de las 103 existentes. Se considera *multi label dataset* pues las etiquetas se encuentran organizadas a modo de árbol, típicamente con profundidad 3, de modo que una misma noticia posee, al menos, una etiqueta. Para este trabajo se utilizó la versión *RCV1-v2* [2], la cuál corresponde a los documentos utilizados en LYRL2004. En total son 804.414 documentos divididos en 23.491 documentos para el entrenamiento de la red; los 781.265 documentos de pruebas que se utilizarán para realizar el *criterio predictivo con promedio micro*, *criterio predictivo con promedio macro* y *jaccard score* para medir el desempeño de la red ya entrenada.

3.3. Pre-Procesamiento

Luego de elegir los datasets, el siguiente paso es dar un mismo formato a los documentos. Esto es necesario para transformar los documentos a un formato comprensible, eliminando información que no sea relevante, de modo que permita un buen desempeño en la ANN. Puesto que los dataset *20 newsgroups* y *reuters* están presentados de forma distinta se utilizan distintos métodos para transformar los documentos al tipo de dato que requiere como entrada la ANN.

Luego del pre-procesamiento, los documentos son almacenados en *MongoDB*, una base de datos no relacional, de modo que se pueda acceder a ellos de forma más simple y, posteriormente, se les añadan directamente la representación generada por la ANN.

1. 20 Newsgroups

Los documentos de este dataset no han sido pre-procesados, contienen mucha información que no será de utilidad, por ello se realiza:

- Eliminar números.
- Eliminar caracteres especiales, por ejemplo, unidades monetarias.
- Eliminar elementos de puntuación.
- Reducir todo el texto a minúscula.

```

1  for archivo in archivos:
2      text = documento_a_pre_procesar
3      # Eliminar numeros
4      text = ''.join([j for j in text if not j.isdigit()])
5      # Eliminar caracteres no letras
6      text = ''.join([x if 31 < ord(x) < 128 else '?'
7                      for x in text])
8      # Eliminar puntuacion
9      for c in string.punctuation:
10         text= text.replace(c," ")
11     # Convertir texto a minuscula
12     text = text.lower()

```

2. Reuters

Los documentos de este dataset se encuentran pre-procesados, de modo que entrega las palabras del modo deseado y no es necesario realizar otra modificación.

3.4. Representación

A partir de los documentos pre-procesados se transforman los documentos al tipo de dato que recibirá la *ANN* como entrada, para ello existen dos etapas: Generar el vocabulario del dataset y generar la representación *TF* que contiene la información relevante del documento.

1. Vocabulario

Contiene solo una copia de cada palabra del dataset ordenada por frecuencia. Para ello:

- Separar cada palabra del texto.

- Eliminar palabras repetidas.
- Indicar la frecuencia de cada palabra en el dataset.
- Ordenar por frecuencia de menor a mayor.

```

1 class word_count_vocab:
2     def __init__(self, word, count=0):
3         self.word = word
4         self.count = count
5 words = []
6 for word in train in all_train:
7     words = words + train['Texto'].split()
8 for test in all_test:
9     words = words + test['Texto'].split()
10 # Elimina palabras repetidas
11 uniquewords = set(words)
12 for word in uniquewords:
13     word_count.append(word_count_vocab(word, words.count(word)))
14 # Ordenar por frecuencia
15 word_count.sort(key=lambda l:l.count)

```

2. *TF*

Los documentos fueron transformados a una representación *TF* para ingresar en la ANN, en donde cada casilla corresponde a una palabra y su contenido indica su frecuencia en el documento. El tamaño del vector depende del experimento a realizar, no obstante para este trabajo se eligen las 2000 palabras con mayor frecuencia en el dataset.

```

1 for doc in docs:
2     tf = doc['Texto'].count(w) for w in vocab_freq
3     id_doc = int(train['ID'])

```

3.5. Métricas de Resultados

Para medir el desempeño de la implementación se utilizó la definición de precisión y recall del modo descrito en [28].

$$precision = \frac{Cantidad\ documentos\ relevantes\ recuperados}{Cantidad\ documentos\ recuperados}. \quad (3.1)$$

$$recall = \frac{Cantidad\ documentos\ relevantes\ recuperados}{Cantidad\ documentos\ relevantes}. \quad (3.2)$$

```
1 Recuperados_misma_etiqueta =  
2     Etiquetas[:,idx_recuperados]  
3 Cantidad_documentos_relevantes_recuperados =  
4     sum(Recuperados_misma_etiqueta,axis=1)  
5 Cantidad_documentos_recuperados = Total_recuperados  
6 Cantidad_documentos_relevantes = sum(Etiquetas,axis=1)
```

Para un *single label dataset* se considera que un documento recuperado s_k es relevante si posee la misma etiqueta que el documento query q_i . En cambio, para el *multi label dataset* se mide la precisión y recall para cada etiqueta t_j de la consulta q_i , para luego utilizar el criterio predictivo con promedio micro y macro.

```
1 macro_relevantes_recuperado =  
2     Cantidad_documentos_relevantes_recuperados  
3 macro_cantidad_recuperados =  
4     Cantidad_documentos_recuperados  
5 macro_cantidad_relevantes =  
6     Cantidad_documentos_relevantes  
7  
8 micro_relevantes_recuperado =  
9     sum(macro_relevantes_recuperado)  
10 micro_cantidad_recuperados =  
11     sum(macro_cantidad_recuperados)  
12 micro_cantidad_relevantes =  
13     sum(macro_cantidad_relevantes)
```

```
14
15 macro_precision =
16     mean(divide(macro_relevantes_recuperado,macro_cantidad_recuperados))
17 macro_recall =
18     mean(divide(macro_relevantes_recuperado,macro_cantidad_relevantes))
19
20 micro_precision =
21     (micro_relevantes_recuperado/micro_cantidad_recuperados)
22 micro_recall =
23     (micro_relevantes_recuperado/micro_cantidad_relevantes)
```

Cabe destacar que en *single lable dataset* el criterio predictivo con promedio micro y macro entrega los mismos resultados, por ello el método descrito con anterioridad fue utilizado para ambos tipos de dataset.

Capítulo 4

Resultados

En este capítulo se muestran los resultados del trabajo realizado, en donde se utilizaron los datasets mencionados con anterioridad: *20 Newsgroups* y *Reuters*.

4.1. Comparación de desempeño con baseline

Para validar la implementación realizada es necesario comparar su desempeño con el desempeño presentado en [28], de modo que se obtengan resultados similares o superiores. Para ello es necesario utilizar la arquitectura de red y tamaño de conjuntos de entrenamiento y de prueba descritos en [28], tanto para *20 Newsgroups* y *Reuters*.

4.1.1. 20 Newsgroups

La figura 4.1 compara el desempeño entre la implementación en *NumPy* con los resultados en [28] utilizando el dataset *20 Newsgroups*.

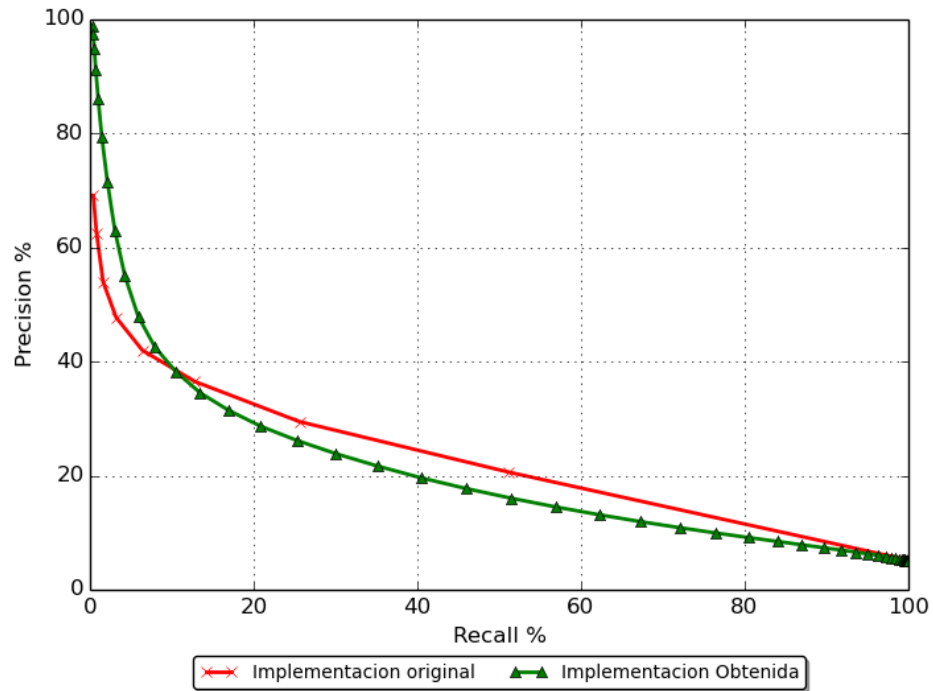


Figura 4.1: Precisión vs recall entre la implementación obtenida y la implementación original [28].

Se observa que el desempeño es similar, siendo levemente superior hasta el 10% de recall y levemente inferior desde dicho valor. Esta diferencia se explica pues en el pre-procesamiento de los documentos no se realizó *stemming* pues disminuye de forma drástica el desempeño del algoritmo, además se eliminó el ruido pues no mejora considerablemente los resultados obtenidos [15]. También, la inicialización aleatoria de los subconjuntos puede influir en el desempeño.

4.1.2. Reuters

A diferencia de *20 Newsgroups*, fue necesario cambiar el tamaño del conjunto de entrenamiento y de prueba descrito con anterioridad para que coincidan con el baseline, por esto se utilizaron 402.207 documentos en el documentos de entrenamiento y 402.207 para el conjunto de prueba, separados de forma aleatoria.

La figura 4.2 compara el desempeño entre la implementación en *NumPy*, según el criterio micro y macro, con el resultado presentado en [28] utilizando el dataset *Reuters*.

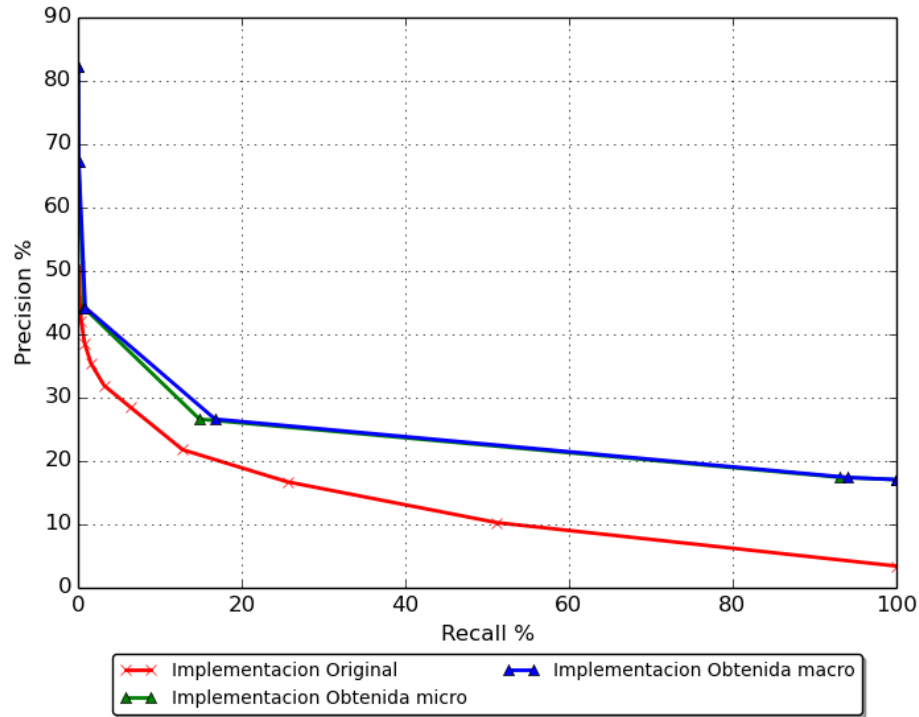


Figura 4.2: Comparación de precisión vs recall entre la implementación obtenida y la implementación original, se muestra el desempeño según el criterio micro y el criterio macro.

Se observa que el desempeño de la implementación en *NumPy* inicia de forma similar pero al aumentar el recall se muestra superior. Del mismo modo que para *20 Newsgroups*, al eliminar el ruido el desempeño alcanzado fue superior.

4.2. Evaluación de sensibilidad de resultados

Tras comprobar que la *ANN* alcanza un desempeño comparable al baseline, se evaluará la sensibilidad de los resultados alterando la arquitectura utilizada. Para esto se definieron 16 arquitecturas distintas, cambiando la cantidad de capas ocultas y el tamaño de las capas

ocultas con el objetivo de determinar qué parámetros o elecciones mejoran el desempeño para el problema propuesto. Se utilizarán arquitecturas de 1 a 5 capas ocultas, donde cada una tendrá 250, 500 o 1000 neuronas. La notación a utilizar será 2000- H^1 -...- H^i -128, en donde H^i corresponde a la cantidad de neuronas de la capa i . Por ejemplo, la arquitectura 2000-500-500-128 tendrá 2 capas ocultas, cada una con 500 neuronas.

Además, se entrenará una red sin capas ocultas utilizando el método de entrenamiento de la *CPM*, definido en la sección 3.4.1.; en cambio, las demás arquitecturas si poseen capas ocultas y serán entrenadas según el método de entrenamiento de la *DBN*, definido en la sección 3.4.2.

Para cada experimento se midió la precisión y recall a distintos radios de búsqueda, los resultados fueron observados hasta el 100 %, 10 % y 1 % de recall con el objetivo de determinar qué arquitectura presenta un mejor desempeño dependiendo del volumen de documentos que se busque recuperar.

Para cada dataset se evaluó la sensibilidad de modelos con la misma cantidad de capas ocultas variando la cantidad de neuronas por capa oculta, modelos con la misma cantidad de neuronas por capa oculta variando la cantidad de capas ocultas y se comparó las arquitecturas con mejor desempeño según quienes comparten la cantidad de neuronas por capa oculta con la arquitectura sin capas ocultas 2000-128.

4.2.1. 20 Newsgroups

4.2.1.1. Sensibilidad con igual cantidad de capas oculta

4.2.1.1.1 1 capa oculta

Se compara la precisión y recall para las arquitecturas con 1 capa oculta.

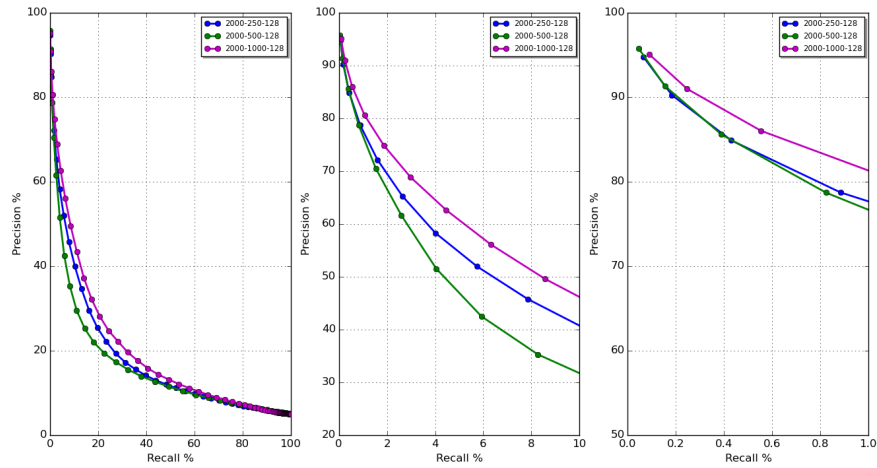


Figura 4.3: Precisión vs recall de arquitecturas con 1 capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo se observa que para un recall superior al 60 % las arquitecturas presentan un desempeño similar; para recall inferior la arquitectura 2000-1000-128 presenta el mejor desempeño, alcanzando una precisión superior en hasta 5 % respecto a 2000-250-128 y hasta un 20 % respecto a 2000-500-128. Además, la arquitectura 2000-500-128 presenta el peor desempeño para recall inferior al 20 %, no obstante desde el 30 % de recall su desempeño no es significativamente inferior al de la arquitectura 2000-250-128.

En el panel central se observa un comportamiento similar, se mantiene la arquitectura 2000-1000-128 con el mejor desempeño, seguido por 2000-250-128 y 200-500-128, cuyo desempeño es el único que baja del 40 %. Sin embargo, para recall cercano al 1 % el desempeño de la arquitectura 2000-500-128 alcanza al desempeño de 2000-250-128.

Finalmente, en el panel derecho se corrobora que la arquitectura 2000-1000-128 presenta el mejor desempeño en estos experimentos siendo el único caso donde la precisión no baja del 80 %. Además, las arquitecturas 2000-250-128 y 2000-500-128 presentan un desempeño casi idéntico. Finalmente, las 3 arquitecturas superan el 95 % de precisión.

4.2.1.1.2 2 capas ocultas

Se compara la precisión y recall para las arquitecturas con 2 capas ocultas.

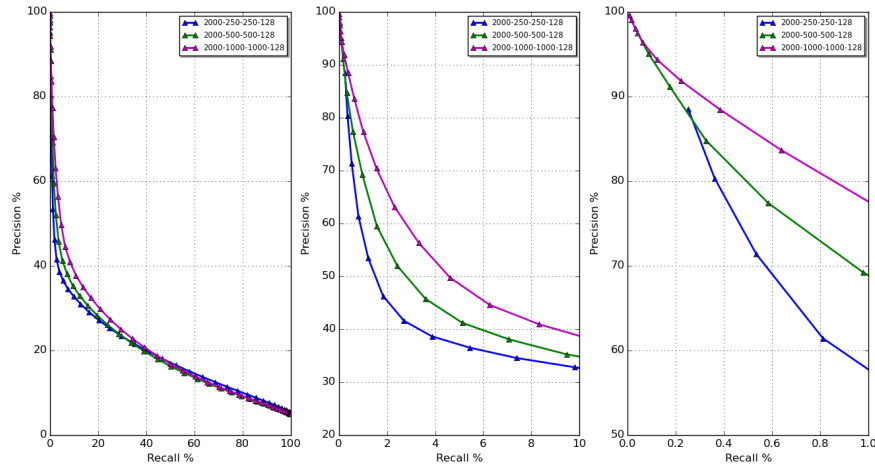


Figura 4.4: Precisión vs recall de arquitecturas con 2 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

El panel izquierdo muestra un comportamiento similar entre las distintas arquitecturas, siendo levemente superior 2000-1000-1000-128 y levemente inferior 2000-250-250-128, incluso para recall superior al 40 % no se observan mayores diferencias en el desempeño de las arquitecturas.

El panel central permite descubrir mayores diferencias en el desempeño de las arquitecturas. La arquitectura 2000-1000-100-128 alcanza una precisión superior al 10 % respecto a 2000-500-500-128 y superior al 15 % respecto a 2000-250-250-128 para recall inferior al 3 %. Además, 2000-250-250-128 presenta el peor desempeño durante todo el intervalo observado.

En el panel derecho confirma a la arquitectura 2000-1000-1000-128 como la de mejor desempeño, siendo la única que supera el 75 % de precisión al 1 % de recall, y a la arquitectura 2000-250-250-128 como la de peor desempeño, siendo la única cuya precisión baja del

60 % y que no supera el 90 %. Además, si bien el desempeño de la arquitectura 2000-500-500-128 se mantuvo constantemente entre las otras arquitecturas las otras, hacia el 0,1 % de recall su desempeño es equivalente al de la arquitectura 2000-1000-1000-128.

4.2.1.1.3 3 capas ocultas

Se compara la precisión y recall para las arquitecturas con 3 capas ocultas.

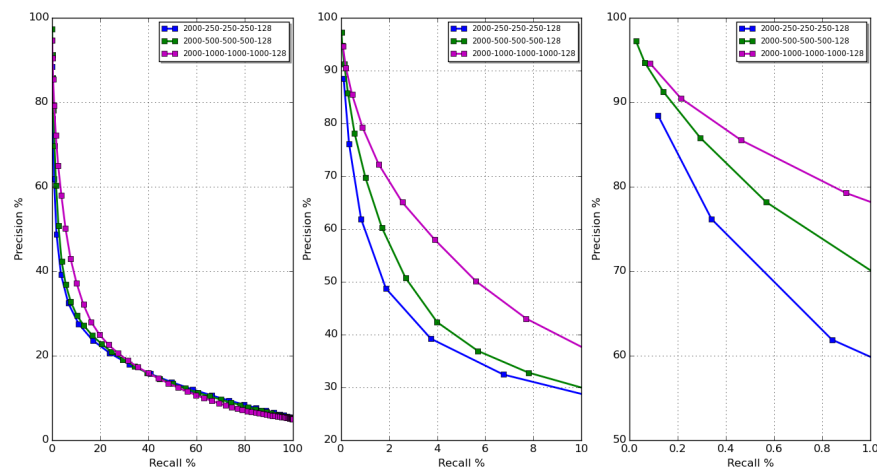


Figura 4.5: Precisión vs recall de arquitecturas con 3 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo se observa que para recall superior al 20 % las arquitecturas presentan un desempeño similar; a menor recall la arquitectura 2000-1000-1000-1000-128 alcanza una precisión levemente superior mientras las demás arquitecturas no muestran diferencias.

En el panel central se observa una mayor diferencia entre el comportamiento de la arquitectura 2000-1000-1000-1000-128 y el resto, alcanzando casi un 20 % más de precisión con recall cercano al 4 %. Además, la arquitectura 2000-500-500-500-128 se diferencia de la arquitectura 2000-250-250-250-128, alcanzando un 10 % más de precisión para recall 2 %.

El panel derecho muestra que la arquitectura 2000-1000-1000-1000-128 se mantiene

con un desempeño superior, sin embargo la mayor precisión que registra es inferior a la mayor que presenta la arquitectura 2000-500-500-500-128, siendo el único momento en toda la medición donde muestra un mejor desempeño. Finalmente, 2000-250-250-250-128 se mantiene con el peor desempeño, siendo la única que no supera el 90 % de precisión.

4.2.1.1.4 4 capas ocultas

Se compara la precisión y recall para las arquitecturas con 4 capas ocultas.

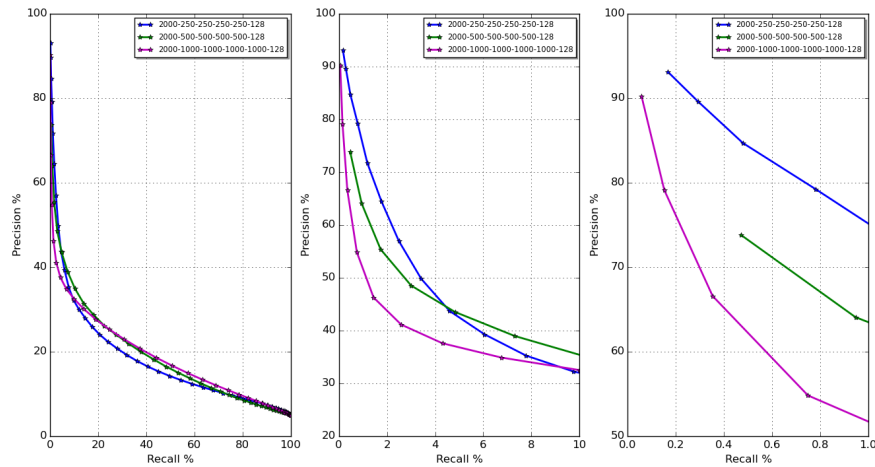


Figura 4.6: Precisión vs recall de arquitecturas con 4 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo se observa un comportamiento muy similar entre las arquitecturas 2000-1000-1000-1000-1000-128 y 2000-500-500-500-500-128; la primera muestra un mejor desempeño para recall superior al 40 %, sin embargo bajo el 20 % es la segunda quien alcanza el mejor desempeño. Además, 2000-250-250-250-250-128 muestra un peor desempeño para recall superior al 10 %.

El panel central muestra una mejora de la arquitectura 2000-250-250-250-250-128, de modo que para recall inferior al 5 % presenta el mejor desempeño de los experimentos. En cambio, la arquitectura 2000-1000-1000-1000-1000-128 baja su desempeño considerablemente, siendo la peor para recall inferior al 10 %, incluso para recall cercano al 2 % su

precisión alcanza un 10 % menos que 2000-500-500-500-500-128 y 20 % respecto a 2000-250-250-250-250-128.

El panel derecho confirma lo anteriormente mencionado, la arquitectura 2000-250-250-250-250-128 posee un desempeño considerablemente superior hasta el 1 % de recall. La arquitectura 2000-500-500-500-500-500-128 presenta un desempeño hasta el 0,5 % recall, donde deja de presentar resultados, por ello es la única arquitectura que no supera el 75 % de precisión. Finalmente, la arquitectura 2000-1000-1000-1000-1000-128 se mantiene con el por desempeño hasta el 0,5 % de recall, pero de igual modo alcanza una precisión máxima cercana al 90 %.

4.2.1.1.5 5 capas ocultas

Se compara la precisión y recall para las arquitecturas con 5 capas ocultas.

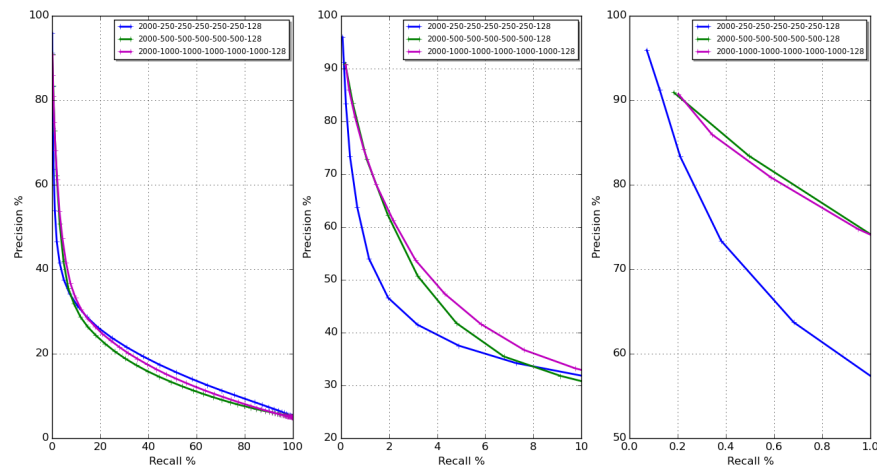


Figura 4.7: Precisión vs recall de arquitecturas con 5 capas ocultas. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo se observa que las arquitecturas presentan un desempeño muy similar, 2000-250-250-250-250-250-128 alcanza una precisión levemente superior para recall superior al 20 %, además 2000-500-500-500-500-500-128 presenta el peor resultado hasta

dicho recall. Para recall inferior al 20 %, la arquitectura con 250 neuronas por capa oculta muestra el peor desempeño, en cambio para las demás no se observan mayores diferencias.

El panel central muestra que la arquitectura 2000-250-250-250-250-128 mantiene el peor desempeño, para recall 2 % es casi un 15 % menor a las demás. Además, para recall superior al 2 % es la arquitectura 2000-1000-1000-1000-1000-128 la que muestra un mejor desempeño, no obstante no logra superar en más del 5 % al mostrado en 2000-500-500-500-500-128 y para recall inferior no se observan diferencias.

En el panel derecho se observa que bajo el 1 % de recall el desempeño de la arquitectura 2000-500-500-500-500-128 es levemente superior al de la arquitectura 2000-1000-1000-1000-1000-128, de modo que no distan en más de un 2 % de precisión. Si bien 2000-250-250-250-250-128 se mantiene con el peor desempeño es la única que muestra resultados para recall inferior al 0,2 %, superando el 95 % de precisión.

Comparando las figuras 4.3, 4.4, 4.5, 4.6 y 4.7 no se observan grandes diferencias de desempeño cuando el objetivo es recuperar una gran cantidad de documentos. No obstante, a medida que se observan recall inferiores, las arquitecturas con más neuronas por capa oculta tienden a mostrar un desempeño considerablemente superior, con excepción del caso con 4 capas ocultas, donde la arquitectura 2000-250-250-250-128 fue superior.

4.2.1.2. Sensibilidad con igual cantidad de neuronas por capa oculta

4.2.1.2.1 250 neuronas

Se compara la precisión y recall para las arquitecturas con 250 neuronas en sus capas ocultas.

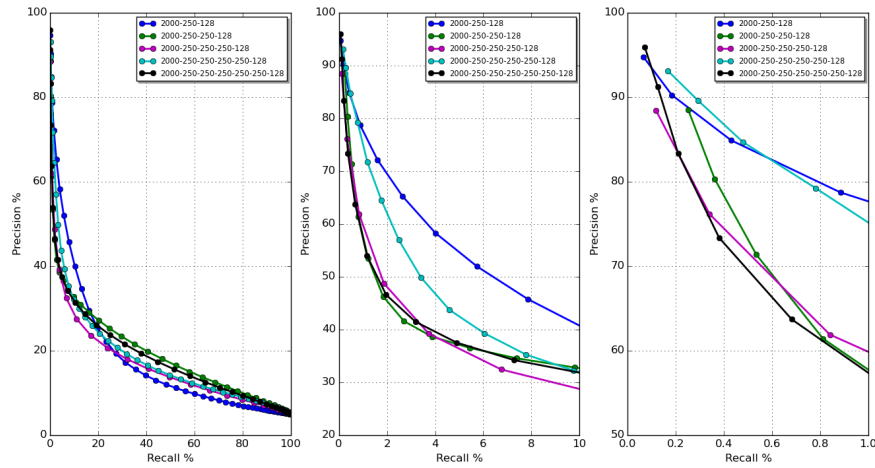


Figura 4.8: Precisión vs recall de arquitecturas con 250 neuronas por capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

Se observa en el panel izquierdo que la arquitectura 2000-250-128 presenta una mayor precisión que el resto de las arquitecturas hasta, aproximadamente, el 10 % de recall, luego su desempeño disminuye considerablemente, de modo que para el 30 % de recall su rendimiento es el peor registrado. De modo opuesto, la arquitectura 2000-250-250-128 muestra un desempeño inicial similar a las arquitecturas 2000-250-250-250-128 y 2000-250-250-250-250-250-128, pero sobre el 20 % alcanza la una precisión mayor.

Respecto a las otras 3 arquitecturas: se observa que 2000-250-250-250-128 y 2000-250-250-250-250-250-128 presentan un desempeño similar para recall superior al 40 %, para valores menores es esta última quien alcanza una precisión mayor. Respecto a la arquitectura 2000-250-250-250-250-250-128, durante toda la medición posee un desempeño similar a 2000-250-250-128, siendo levemente inferior a medida que aumenta el recall.

El panel central corrobora el mayor desempeño de la arquitectura 2000-250-128, sin embargo muestra que para recall inferior al 1 % es alcanzado por 2000-250-250-250-250-128. Las otras 3 arquitecturas muestran desempeños similares, no distan en más de un 5 %.

En el panel derecho la arquitectura 2000-250-250-250-250-128 alcanza una precisión

levemente superior a 2000-250-128 para recall inferior al 0,5 %. Además, para recall inferior al 3 % la arquitectura 2000-250-250-250-128 alcanza el desempeño de 2000-250-128. Respecto a las arquitecturas 2000-250-250-250-128 y 2000-250-250-250-250-128 se observa que la primera es levemente superior en hasta un 2 % para recall superior al 0,3 %, luego no presentan mayores diferencias.

4.2.1.2.2 500 neuronas

Se compara la precisión y recall para las arquitecturas con 500 neuronas en sus capas ocultas.

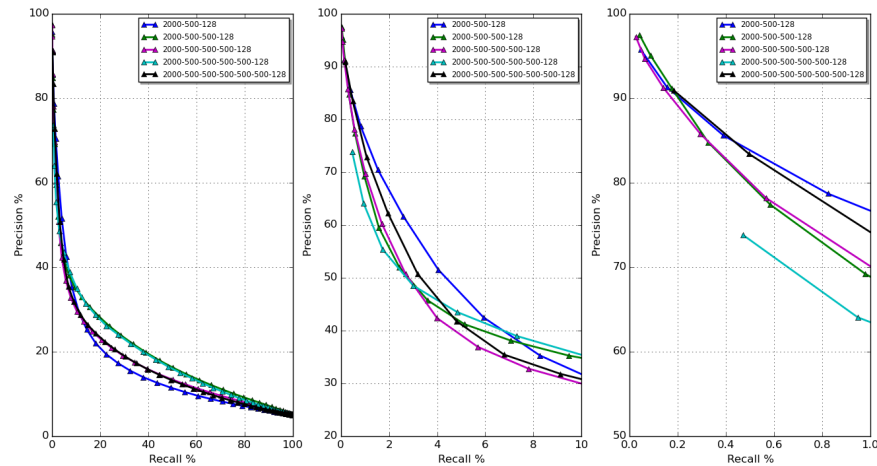


Figura 4.9: Precisión vs recall de arquitecturas con 500 neuronas por capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo se observa que las arquitecturas 2000-500-500-128 y 2000-500-500-500-500-128 presentan un desempeño muy similar y superior al resto de las arquitecturas. Además, se observa un desempeño similar, pero inferior hasta en 5 % respecto a las arquitecturas ya mencionadas, entre las arquitecturas 2000-500-500-500-128 y 2000-500-500-500-500-500-128. Finalmente, para recall superior al 20 % el peor desempeño se observa en 2000-500-128

En el panel central se observan algunas diferencias. En primer lugar, la arquitectura 2000-500-128 pasa de ser la de peor desempeño a la mejor para recall inferior al 6 %. Luego, el desempeño entre las arquitecturas 2000-500-500-500-128 y 2000-500-500-500-500-128 se diferencia, siendo esta última la que alcanza una precisión más alta y bajo el 4 % de recall se vuelve la segunda mejor arquitectura. Además, 2000-500-500-500-128 se presenta el peor desempeño para recall superior al 3 %.

Respecto a la arquitectura 2000-500-500-128, se observa que para recall superior al 3 % su desempeño es levemente inferior a 2000-500-500-500-500-128, bajo este valor supera levemente esta arquitectura y su desempeño es similar al de la arquitectura 2000-500-500-500-128. Finalmente, 2000-500-500-500-500-128 pasa de tener el mejor desempeño, con recall superior al 7 %, a ser el peor, con recall inferior al 3 %.

En el panel derecho se confirma lo observado en el panel central: la arquitectura 2000-500-128 alcanza una mayor precisión, sin embargo la arquitectura 2000-500-500-500-500-128 alcanza una precisión similar para recall menor al 0,5 %. Además, se corrobora un desempeño muy similar entre 2000-500-500-128 y 2000-500-500-500-128. Finalmente, el desempeño de la arquitectura 2000-500-500-500-500-128 es considerablemente inferior al resto, su última medición no supera el 75 % de precisión mientras el resto supera el 90 %.

4.2.1.2.3 1000 neuronas

Se compara la precisión y recall para las arquitecturas con 1000 neuronas en sus capas ocultas.

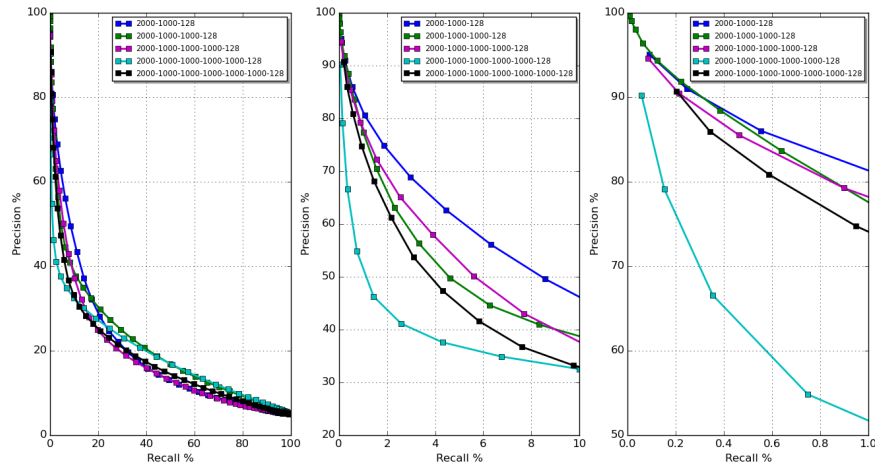


Figura 4.10: Precisión vs recall de arquitecturas con 1000 neuronas por capa oculta. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

El panel izquierdo muestra un comportamiento similar entre las distintas arquitecturas para recall superior al 20 %, en donde la arquitectura 2000-1000-128 supera levemente el desempeño del resto. Además, se observa que el desempeño entre 2000-1000-1000-100-128 y 2000-1000-1000-1000-1000-1000-128 se diferencia solo para recall bajo el 20 %. Respecto a 2000-1000-1000-128 su desempeño es similar al de 2000-1000-1000-1000-1000-128 para recall superior al 40 %, luego el desempeño de la segunda arquitectura disminuye considerablemente. Finalmente, las arquitecturas 2000-1000-1000-1000-128 y 2000-10000-1000-1000-1000-1000-128 presentan un desempeño muy cercano.

En el panel central se observa que la arquitectura 2000-1000-128 se mantiene con mayor precisión respecto al resto, siendo seguida por 2000-1000-1000-1000-128, cuyo desempeño se distingue de 2000-1000-1000-1000-1000-1000-128 superándolo en hasta 10 % de precisión y supera levemente a 2000-1000-1000-128. La arquitectura 2000-1000-1000-1000-1000-128 presenta el peor desempeño, llega a registrar sobre un 30 % de diferencia en precisión respecto a 2000-1000-128 para recall cercano al 1 %.

En el panel derecho se corrobora lo anteriormente observado, el desempeño de la arquitectura 2000-1000-128 se mantiene superior para recall superior al 0,4 %, bajo este valor su

desempeño es alcanzado por 2000-1000-1000-128 y bajo el 0,2 % es igualado por el resto de las arquitecturas, salvo 2000-1000-1000-1000-1000-128. Esta arquitectura mantiene su desempeño considerablemente inferior, siendo la única que baja del 70 % antes del 1 % de recall.

Las figuras 4.8, 4.9 y 4.10 muestran que manteniendo la misma cantidad de neuronas por capa es posible alcanzar una precisión alta utilizando arquitecturas menos profundas si solo es necesario alcanzar el 10 % de recall. Además, las redes con 4 capas ocultas tienden a presentar el peor desempeño a medida que baja el recall.

Para búsqueda con recall alto las arquitecturas con 4 y 5 capas ocultas muestran un mejor desempeño, pero no logra ser superior en un 10 % al mostrado por las redes menos profundas.

4.2.1.3. Comparación de mejores modelos

Se eligen las arquitecturas con mejor desempeño para una misma cantidad de neuronas por capa oculta, es decir, se elige la mejor arquitectura con 250, 500 y 1000 neuronas, y se comparan con el desempeño de la arquitectura 2000-128 para determinar cuál presenta el mejor desempeño dentro de los experimentos realizados. Para elegir las mejores arquitecturas se mide el F1 score de cada punto menor al recall indicado y se promedian.

Arquitectura	Recall 1 %	Recall 10 %	Recall 100 %
2000-250-128	0,7751	4,727	10,85
2000-250-250-128	0,9704	5,330	11,03
2000-250-250-250-128	0,8564	4,029	9,733
2000-250-250-250-250-128	0,8542	5,574	11,12
2000-250-250-250-250-250-128	0,5826	3,556	10,77
2000-500-128	0,7016	4,679	10,37
2000-500-500-128	0,7236	4,559	10,88
2000-500-500-500-128	0,4332	3,850	10,59
2000-500-500-500-500-128	1,400	5,468	11,22
2000-500-500-500-500-500-128	0,6746	5,927	10,94
2000-1000-128	0,5905	5,2839	11,05
2000-1000-1000-128	0,3687	3,467	10,80
2000-1000-1000-1000-128	0,8217	4,635	10,99
2000-1000-1000-1000-1000-128	0,6495	4,953	10,81
2000-1000-1000-1000-1000-1000-128	1,029	5,709	11,23

Cuadro 4.1: F1 score promedio de las arquitecturas para recall hasta el 100 %, 10 % y 1 % en 20 Newsgroups.

De este modo, los experimentos a comparar con la arquitectura 2000-128, separados por recall, son:

1. Recall 100 %:

- 2000-250-250-250-250-128
- 2000-500-500-500-500-128
- 2000-1000-1000-1000-1000-1000-128

2. Recall 10 %:

- 2000-250-250-250-250-128

- 2000-500-500-500-500-128
- 2000-1000-1000-1000-1000-128

3. Recall 1 %:

- 2000-250-250-128
- 2000-500-500-500-500-128
- 2000-1000-1000-1000-1000-128

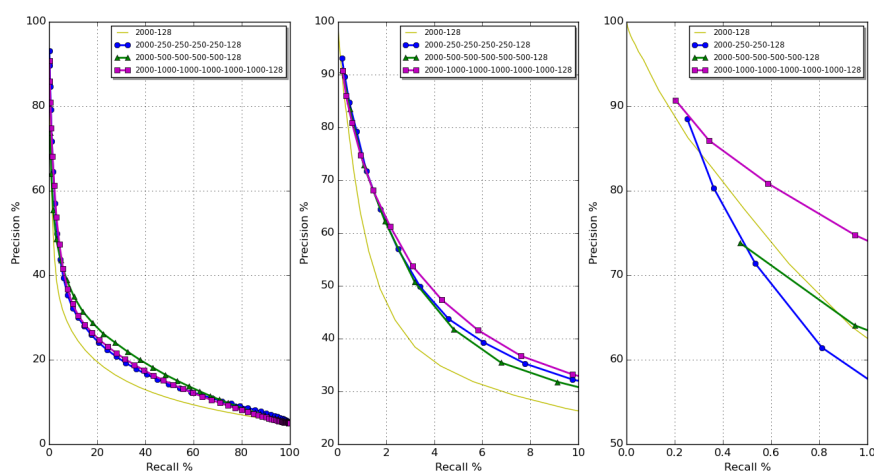


Figura 4.11: Precisión vs recall de las arquitecturas con mejor desempeño. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

El panel izquierdo muestra un comportamiento muy similar entre las arquitecturas con capas ocultas, siendo 2000-500-500-500-500-128 levemente superior en el 10 % y 60 % de recall, esto se condice con la medición del F1 score donde difieren en pocos decimales. Además, la arquitectura sin capas ocultas no alcanza una precisión similar, siendo inferior en hasta un 10 % respecto a la arquitectura con 500 neuronas por capa oculta.

En el panel central se mantienen arquitecturas con 4 o más capas ocultas, en donde la arquitectura 2000-1000-1000-1000-1000-128 alcanza una precisión levemente superior al resto para recall superior al 2 %, sin embargo no les supera en más de 2 % de precisión.

Finalmente, en 2000-128 se mantiene el peor desempeño, siendo la única que baja del 30 % de precisión antes del 10 % de recall.

En el panel derecho se considera por única vez una arquitectura con 2 capas ocultas, 2000-250-250-128, cuya precisión es menor al resto para recall superior al 0,2 %, incluso hacia el 1 % es casi un 20 % menor en precisión. Luego, se mantiene 2000-1000-1000-1000-1000-1000-128 con el desempeño superior, siendo seguido por 2000-128, cuyo desempeño supera levemente a las arquitecturas de 250 y 500 neuronas por capa oculta.

La figura 4.11 muestra que las arquitecturas con capas ocultas presentan un desempeño superior al observado en 2000-128, aunque es posible encontrar obtener resultados mediante esta arquitectura en el intervalo inferior al 1 % de recall. Además, si bien se alcanzó un mayor F1 score en las arquitecturas más profundas este valor no dista en demasía del alcanzado en aquellos con solo 1 capa, esto coincide con la precisión observada en las mediciones hasta el 10 % de recall, en donde se alcanzó precisión similar, incluso superior, a la alcanzada por modelos más profundos.

La tabla 4.2 muestra la precisión de las distintas arquitecturas hasta el 100 %, 10 % y 1 % de recall.

Arquitectura	Recall 1 %	Recall 10 %	Recall 100 %
2000-250-128	78,74	45,78	5,050
2000-250-250-128	61,43	32,82	5,050
2000-250-250-250-128	61,91	32,43	5,050
2000-250-250-250-250-128	79,22	32,24	5,050
2000-250-250-250-250-250-128	63,71	34,21	5,050
2000-500-128	78,71	35,29	5,050
2000-500-500-128	69,23	35,26	5,050
2000-500-500-500-128	78,17	32,80	5,050
2000-500-500-500-500-128	64,08	38,97	5,050
2000-500-500-500-500-500-128	83,44	31,80	5,050
2000-1000-128	86,03	49,58	5,050
2000-1000-1000-128	83,66	40,95	5,050
2000-1000-1000-1000-128	79,27	43,03	5,050
2000-1000-1000-1000-1000-128	54,82	32,52	5,050
2000-1000-1000-1000-1000-1000-128	74,78	33,21	5,050

Cuadro 4.2: Precisión para distintos Recall de las arquitecturas entrenadas en 20 News-groups.

4.2.2. Reuters

Reuters es un *multilabel dataset*, por ello para medir precisión y recall se utilizó el criterio predictivo con promedio micro y macro, estos se explican en la sección [2.5.1.4](#) y [2.5.1.5](#).

4.2.2.1. Sensibilidad con igual cantidad de capas oculta

4.2.2.1.1 1 capa oculta

Se compara la precisión y recall según el criterio predictivo con promedio micro y macro para las arquitecturas con 1 capa oculta.

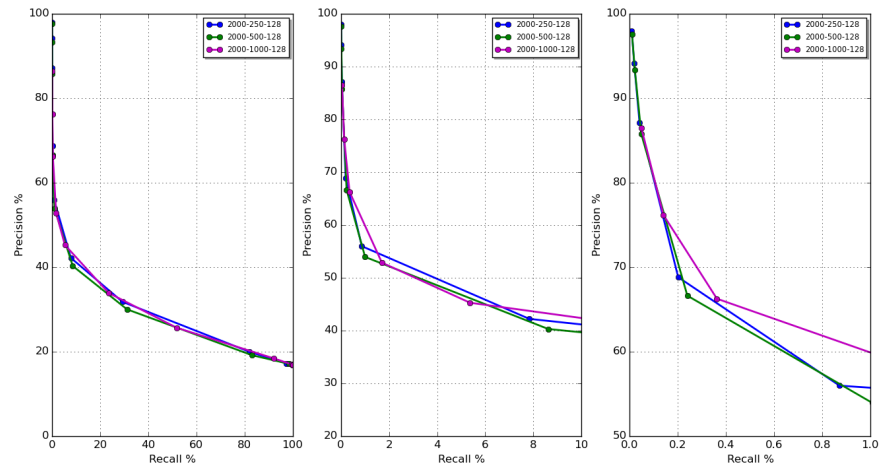


Figura 4.12: Precisión vs recall de arquitecturas con 1 capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

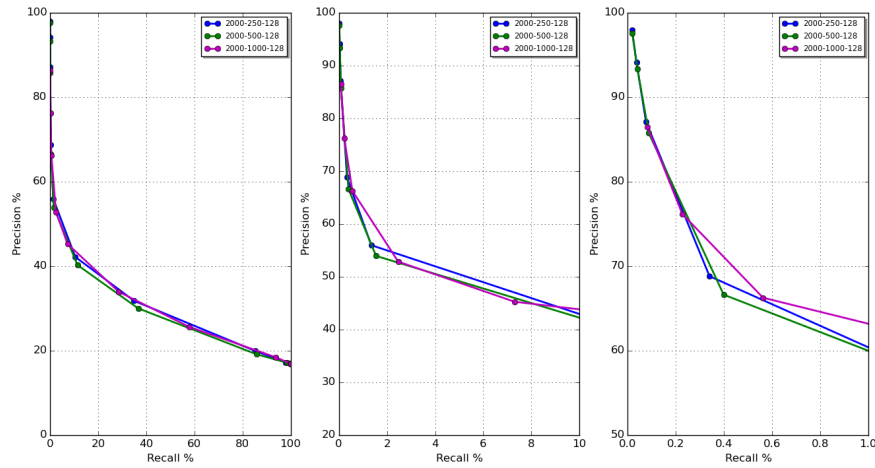


Figura 4.13: Precisión vs recall de arquitecturas con 1 capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo de las figuras 4.12 y 4.13 no se observa una arquitectura cuyo desempeño sea claramente superior, el desempeño de 2000-250-128 y 2000-1000-128 es prácticamente el mismo. En cambio, entre el 10 % y 50 % de recall la arquitectura 2000-500-128 posee un desempeño levemente inferior al resto.

En el panel central de la figura 4.12 se observa una leve superioridad de precisión de la arquitectura 2000-250-128 entre el 2 % y 6 % de recall. En dicho intervalo el desempeño de 2000-1000-128 es similar al de 2000-500-128. En cambio, en la figura 4.13 se observa esto pero para recall entre el 2 % y 8 %.

En el panel derecho de la figura 4.12 se observa que la arquitectura 2000-1000-128 supera al resto de las arquitecturas para recall superior al 0,2 %. Si bien 2000-500-128 se mantuvo con desempeño más bajo, la arquitectura 2000-250-128 bajó su desempeño y se comportó de modo similar. Esto coincide en la figura 4.13 con la única diferencia que 2000-250-128 y 2000-500-128 se comportan similar para recall inferior al 0,4 %.

4.2.2.1.2 2 capas ocultas

Se compara la precisión y recall según el criterio predictivo con promedio micro y macro para las arquitecturas con 2 capas ocultas.

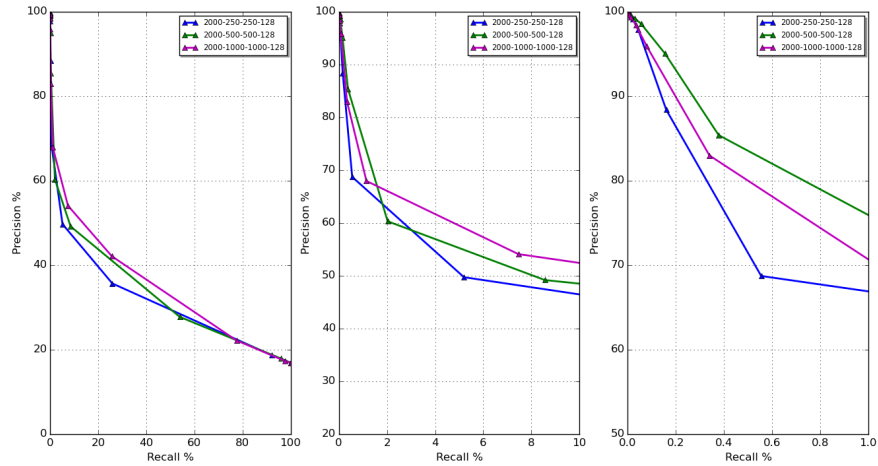


Figura 4.14: Precisión vs recall de arquitecturas con 2 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

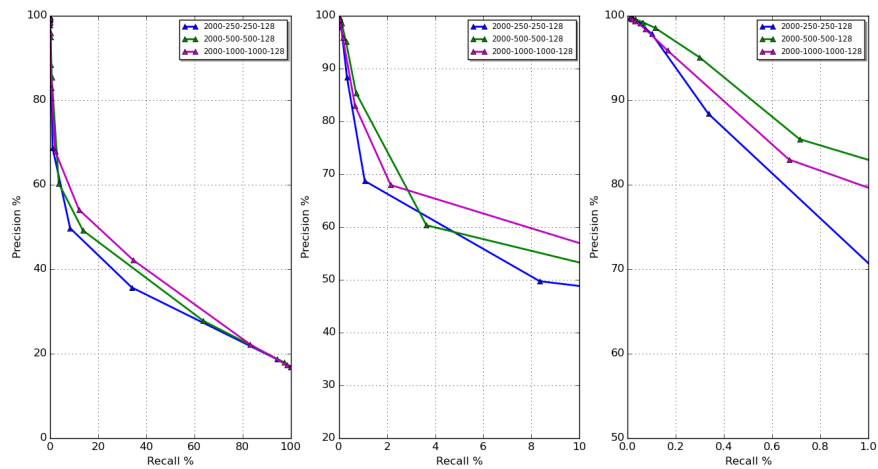


Figura 4.15: Precisión vs recall de arquitecturas con 2 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

El panel izquierdo de las figuras 4.14 y 4.15 muestran que la arquitectura 2000-1000-1000-128 alcanza una precisión mayor. Además, la arquitectura 2000-250-250-128 se mantuvo con la menor precisión entre el 10 % y 80 % de recall.

En el panel central de la figura 4.14 se observa que la arquitectura 2000-1000-1000-128 se mantiene con la mayor precisión para recall superior al 2 %, luego es superado por 2000-500-500-128; además, para este mismo recall se encuentra la única medición en donde la arquitectura 2000-250-250-128 no presenta el peor desempeño. En la figura 4.15 se corrobora el buen desempeño de la arquitectura 2000-1000-1000-128 pero en el 3 % de recall es superado por 2000-500-500-128.

En el panel derecho de las figuras 4.14 y 4.15 se observa de forma clara que la arquitectura 2000-500-500-128 alcanza una precisión más alta para recall inferior al 1 %, seguido por 2000-1000-1000-128 y 2000-250-250-128. Sin embargo, en la primera figura las tres arquitecturas alcanzan una precisión menor al 80 % para el 1 % de recall, en cambio en la segunda figura la arquitectura 2000-500-500-128 supera el 80 % de precisión para dicho recall.

4.2.2.1.3 3 capas ocultas

Se compara la precisión y recall según el criterio predictivo con promedio micro y macro para las arquitecturas con 3 capas ocultas.

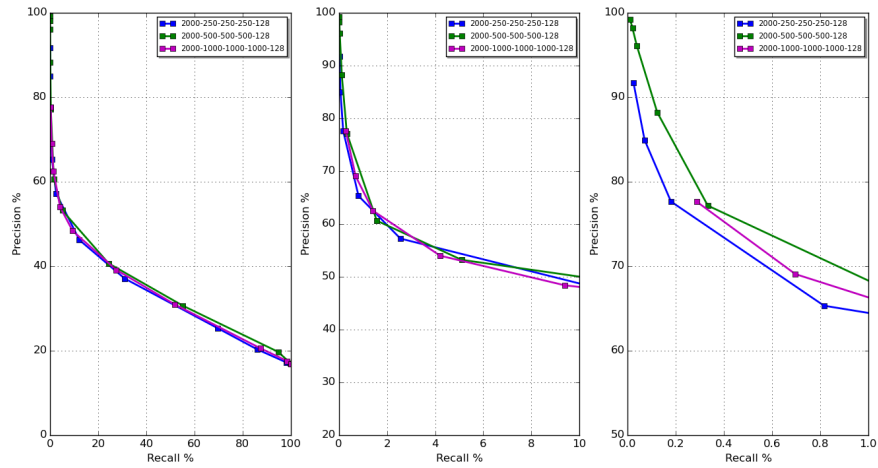


Figura 4.16: Precisión vs recall de arquitecturas con 3 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

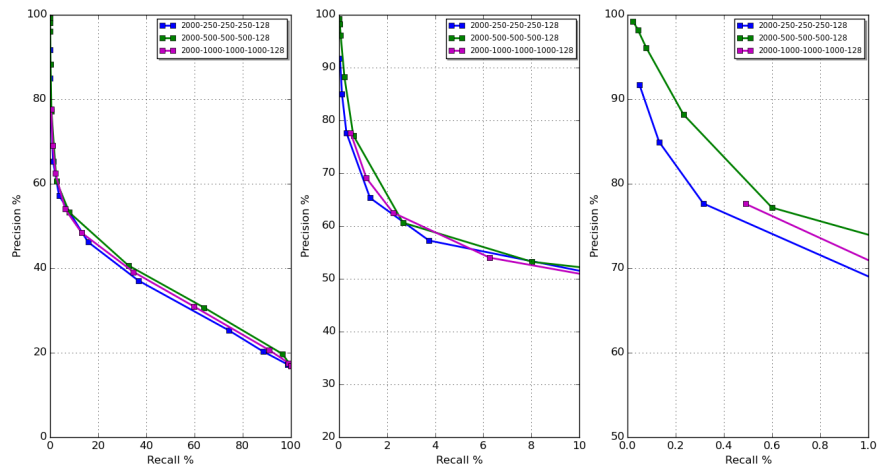


Figura 4.17: Precisión vs recall de arquitecturas con 3 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo de las figuras 4.16 y 4.17 no se distingue una gran diferencia de desempeño, en especial en la primera figura.

En el panel central de la figura 4.16 no se distinguen mayor diferencia con excepción que para recall superior al 3 % la arquitectura 2000-1000-1000-1000-128 alcanza levemente una precisión menor. En cambio, en la figura 4.17 se observa que la arquitectura 2000-500-500-500-128 es levemente superior para recall inferior al 2 % y entre el 4 % y 8 %.

En el panel derecho de la figura 4.16 se observa que la arquitectura 2000-500-500-500-128 alcanza una mayor precisión, seguida por 2000-1000-1000-1000-128 y por 2000-250-250-250-128. Cabe destacar que la arquitectura con 1000 neuronas por capa oculta no supera el 80 % de recall. En la figura 4.17 esto se refleja pero con mayor diferencia entre los desempeños, además solo la arquitectura 2000-250-250-250-128 baja el 70 % de precisión para el 1 % de recall.

4.2.2.1.4 4 capas ocultas

Se compara la precisión y recall según el criterio predictivo con promedio micro y macro para las arquitecturas con 4 capas ocultas.

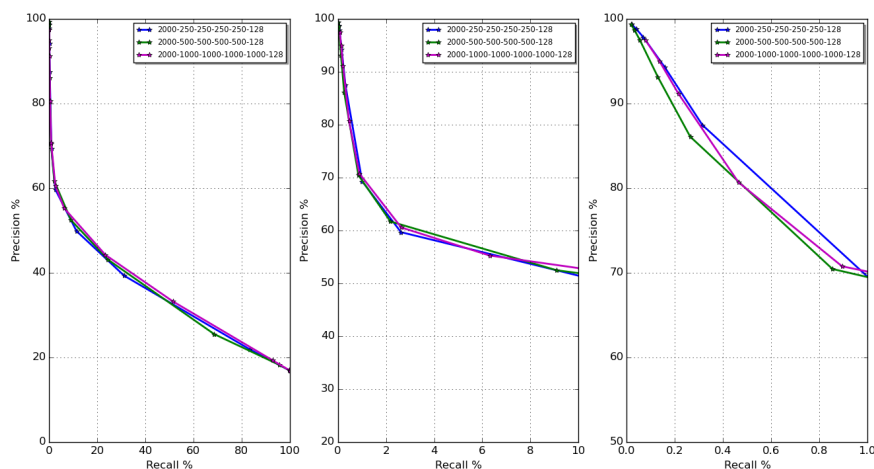


Figura 4.18: Precisión vs recall de arquitecturas con 4 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

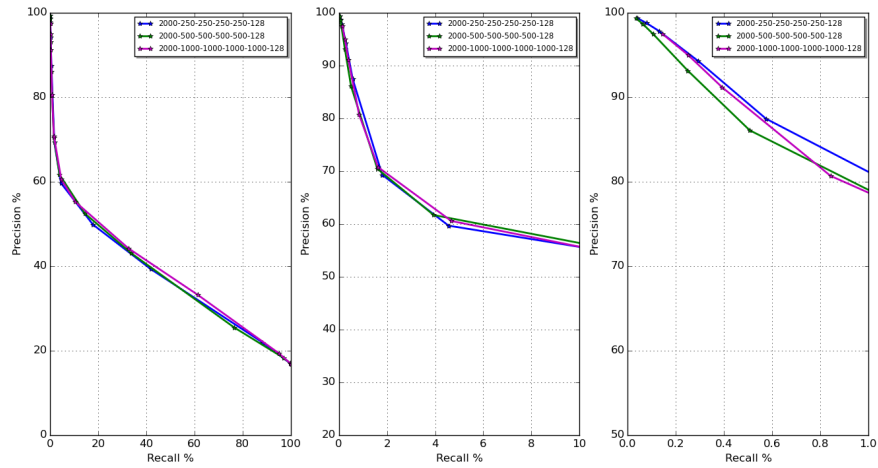


Figura 4.19: Precisión vs recall de arquitecturas con 4 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo de las figuras 4.18 y 4.19 no se observa una gran diferencia en el desempeño de las arquitecturas. En ambas figuras 2000-1000-1000-1000-1000-128 es levemente superior para recall superior al 10 %.

En el panel central de la figura 4.18 se observa que la arquitectura 2000-500-500-500-500-128 es levemente superior entre el 3 % y 7 % de recall, en cambio 2000-250-250-250-250-128 registra menos precisión. En ambos casos la diferencia es mínima. En la figura 4.19 estas diferencias son ligeramente más grandes, de modo que para recall superior al 4 % las arquitecturas 2000-500-500-500-500-128 y 2000-1000-1000-1000-1000-128 alcanzan una precisión similar pero bajando del 4 % la precisión de la primera arquitectura baja, coincidiendo con 2000-250-250-250-250-128.

En el panel derecho de las figuras 4.18 y 4.19 se observa que la arquitectura 2000-250-250-250-250-128 alcanza la mayor precisión, en cambio la arquitectura 2000-500-500-500-500-128 presenta la más baja, siendo levemente inferior a la arquitectura 2000-1000-1000-1000-1000-128. Sin embargo, estas figuras se diferencian en que los resultados en la primera son muy cercanos al 70 %, en cambio en la segunda están cerca del 80 %, para el 1 % de recall.

4.2.2.1.5 5 capas ocultas

Se compara la precisión y recall según el criterio predictivo con promedio micro y macro para las arquitecturas con 5 capas ocultas.

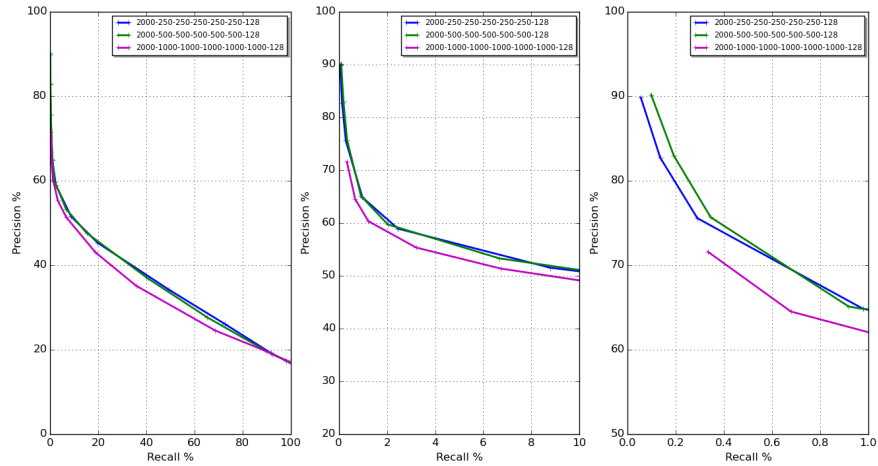


Figura 4.20: Precisión vs recall de arquitecturas con 5 capas ocultas según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

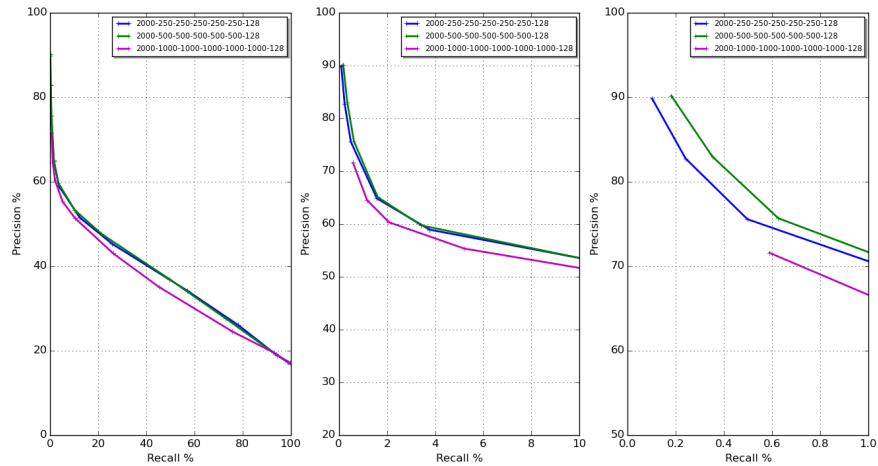


Figura 4.21: Precisión vs recall de arquitecturas con 5 capas ocultas según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo de las figuras 4.20 y 4.21 se observa que la arquitectura 2000-1000-1000-1000-1000-128 presenta el desempeño más bajo en prácticamente toda la medición, en cambio las otras arquitecturas muestran un desempeño similar.

En el panel central de las figuras 4.20 y 4.21 se observa que la precisión de la arquitectura 2000-1000-1000-1000-1000-128 sigue siendo menor, incluso baja en un 3 %, sin embargo las demás arquitecturas mantienen una precisión similar.

Finalmente, en el panel derecho las figuras se diferencian un poco. En la figura 4.20 se observa que para recall menor al 0,6 % la arquitectura 2000-500-500-500-500-128 muestra un desempeño superior a 2000-250-250-250-250-128 y la precisión de ambas es menor al 70 % para recall cercano al 1 %. En cambio, en la figura 4.21 se observa esta diferencia desde el 1 % de recall, incluso hacia el 0,2 % de recall la arquitectura 2000-500-500-500-500-128 alcanza un 5 % más en precisión, además ambas arquitecturas superan el 70 % para recall 1 %.

Respecto a la cantidad de neuronas por capa oculta se observa que para recall hasta el 100 % y 10 % no hay mucha diferencias, con excepción de los experimentos con 2 capas ocultas en donde la diferencia de precisión entre la mejor y peor arquitectura no supera el 5 % y 10 % de recall, respectivamente. En cambio, en la medición hasta el 1 % de recall se observa que en los experimentos con 1 capa oculta utilizar 1000 neuronas otorga hasta un 5 % más de precisión que la arquitectura con 500 neuronas. Luego, en los experimentos con 2, 3 y 5 capas ocultas las arquitecturas con 500 neuronas son superiores, además en el experimento con 4 capas su desempeño levemente menor. Las arquitecturas con 250 neuronas mejoran su desempeño al utilizar 4 o más capas ocultas, alcanzando una precisión muy cercana a la observada con 500 neuronas, si no se requiere la máxima precisión son una buena opción.

4.2.2.2. Sensibilidad con igual cantidad de neuronas por capa oculta

4.2.2.2.1 250 neuronas

Se compara la precisión y recall según el criterio micro y macro para las arquitecturas con 250 neuronas en sus capas ocultas.

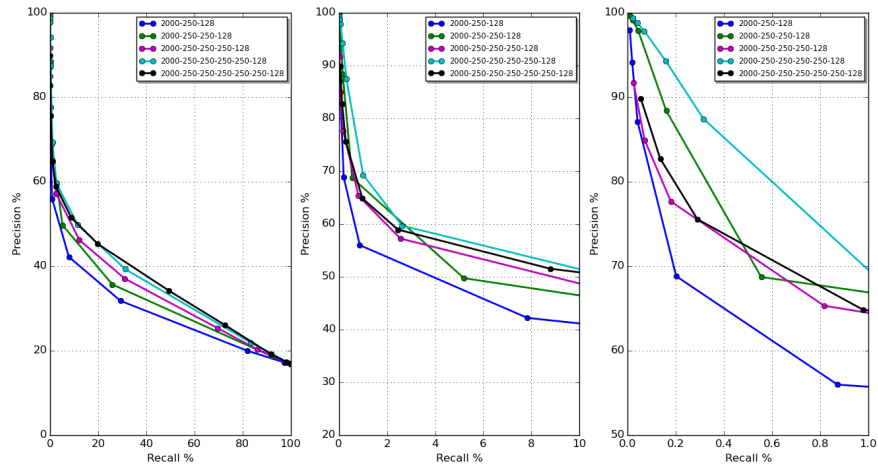


Figura 4.22: Precisión vs recall de arquitecturas con 250 neuronas por capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

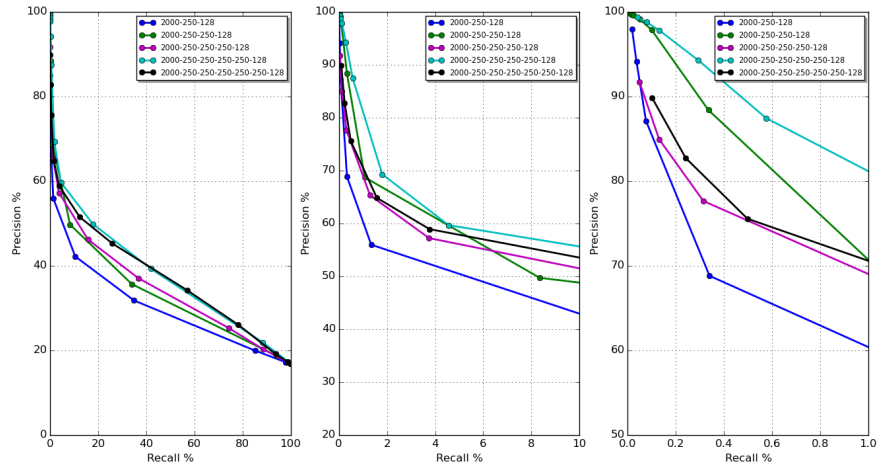


Figura 4.23: Precisión vs recall de arquitecturas con 250 neuronas por capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

En el panel izquierdo de las figuras 4.22 y 4.23 se observa que 2000-250-250-250-250-250-128 y 2000-250-250-250-250-128 poseen un comportamiento similar, siendo la primera quien alcanza una precisión levemente superior desde el 20 % de recall. Respecto al resto

de las arquitecturas se observa que quienes poseen más capas ocultas alcanzan una mayor precisión, es decir, 2000-250-250-250-128 presenta un desempeño levemente superior a 2000-250-250-128 y a 2000-250-128, siendo esta última quien presenta el peor desempeño.

Respecto al panel central de la figura 4.22 se observa que las arquitecturas más profundas presentan la mayor precisión desde el 3 % de recall, para recall inferior tanto 2000-250-250-250-128 como 2000-250-250-128 muestran un mejor resultado, siendo la primera superior. Además, para este mismo recall 2000-250-250-250-128 muestra un desempeño similar a 2000-250-250-250-250-128. Finalmente, 2000-250-128 se mantiene con los peores resultados con precisión al menos 10 % menor a las mejores arquitecturas. Al comparar con la figura 4.23 se observa el mismo comportamiento.

El panel derecho de 4.22 y 4.23 corroboran los resultados observados anteriormente. Nuevamente 2000-250-250-250-250-128 alcanza la mayor precisión, siendo secundado por 2000-250-250-128, mientras que 2000-250-250-250-250-128 y 2000-250-250-128 muestran un desempeño similar. Finalmente, 2000-250-128 muestra los peores resultados, incluso es el único caso donde la precisión baja del 60 % con el 1 % de recall en la figura 4.22.

4.2.2.2.2 500 neuronas

Se compara la precisión y recall según el criterio micro y macro para las arquitecturas con 500 neuronas en sus capas ocultas.

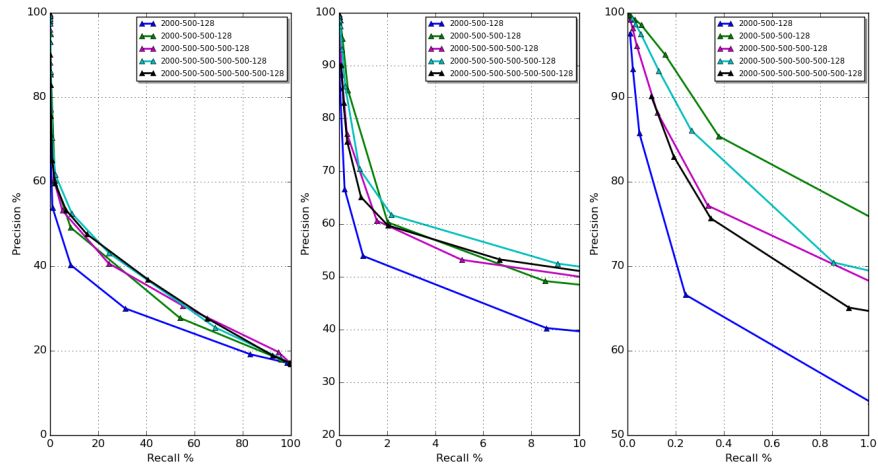


Figura 4.24: Precisión vs recall de arquitecturas con 500 neuronas por capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

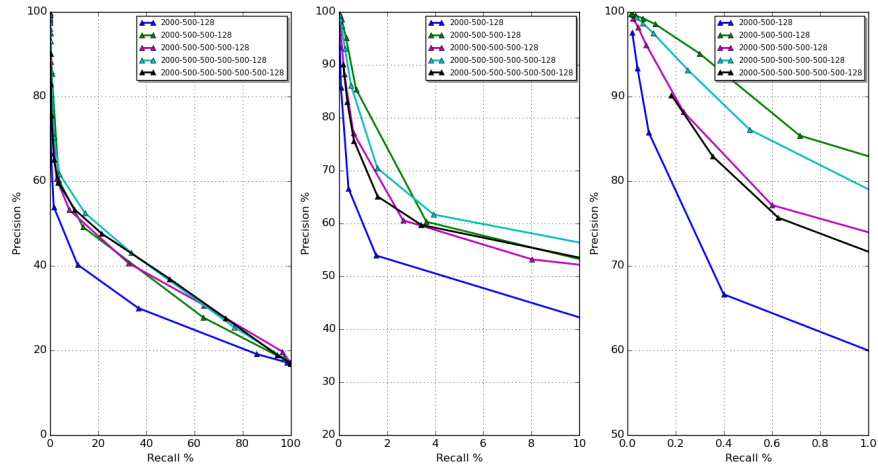


Figura 4.25: Precisión vs recall de arquitecturas con 500 neuronas por capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

De modo similar que en las arquitecturas con 250 neuronas, el panel izquierdo de las figuras 4.24 y 4.25 muestra que las arquitecturas con más capas ocultas poseen un mejor desempeño que aquellas con menos capas ocultas.

En el panel central de las figuras 4.24 y 4.25 se observan diferencias en el desempeño de las arquitecturas con más de una capa oculta: la arquitectura con mayor precisión es 2000-500-500-500-500-128 con recall superior al 2 %, el resto no presenta mayor diferencia. Sin embargo, para recall menor la arquitectura 2000-500-500-128 alcanza una mayor precisión.

En el panel de la derecha de las figuras 4.24 y 4.25 se observa que 2000-500-500-128 alcanza la mayor precisión, seguido de 2000-500-500-500-500-128; luego, se observa un comportamiento bastante similar entre 2000-500-500-500-128 y 2000-500-500-500-500-128, siendo la primera levemente superior. Finalmente, se corrobora el peor comportamiento de 2000-500-128, según el criterio micro y macro su precisión es menor o igual al 60 % para el 1 % de recall.

4.2.2.2.3 1000 neuronas

Se compara la precisión y recall según el criterio micro y macro para las arquitecturas con 1000 neuronas en sus capas ocultas.

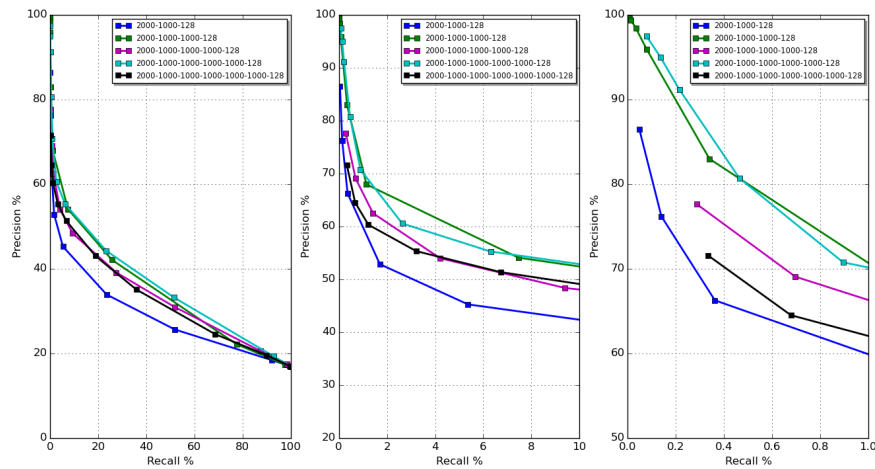


Figura 4.26: Precisión vs recall de arquitecturas con 1000 neuronas por capa oculta según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

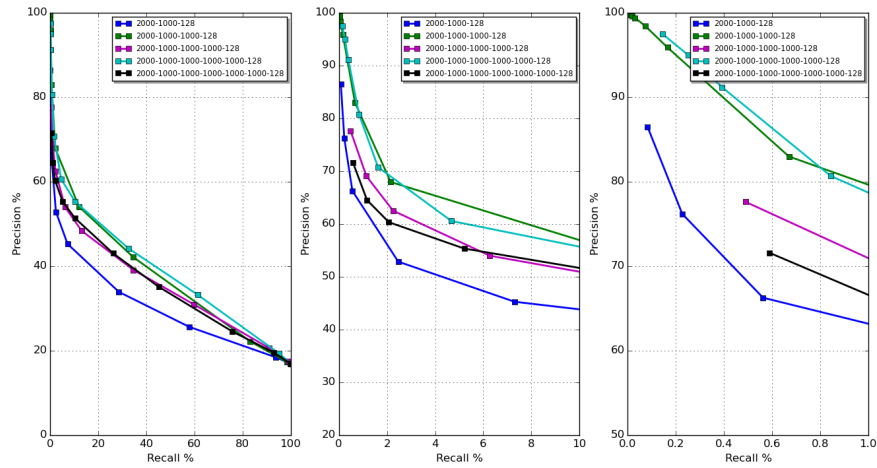


Figura 4.27: Precisión vs recall de arquitecturas con 1000 neuronas por capa oculta según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

El panel izquierdo de las figuras 4.26 y 4.27 muestra que las arquitecturas 2000-1000-1000-128 y 2000-1000-1000-1000-1000-128 alcanzan una precisión muy similar. Lo mismo ocurre entre 2000-1000-1000-1000-128 y 2000-1000-1000-1000-1000-1000-128 pero alcanzan una precisión levemente inferior. La excepción vuelve a ser la arquitectura con menos capas ocultas, cuyo desempeño es claramente inferior al resto.

En el panel central de las figuras 4.26 y 4.27 se mantiene el comportamiento similar entre 2000-1000-1000-1000-128 y 2000-1000-1000-1000-1000-128, en cambio se observa que 2000-1000-1000-128 presenta una precisión superior al resto entre el 2 % y 6 % de recall. Las demás arquitecturas mantienen las características indicadas con anterioridad.

En el panel derecho de las figuras 4.26 y 4.27 sólo se encuentra un cambio entre 2000-1000-1000-128 y 2000-1000-1000-1000-1000-128, siendo esta última quien muestra una precisión levemente superior. Para el resto se corrobora lo anteriormente indicado, sin embargo cabe destacar que ni 2000-1000-1000-1000-128 ni 2000-1000-1000-1000-1000-128 superan el 80 % de precisión, incluso la segunda no llega al 75 %.

Manteniendo constante la cantidad de neuronas por capa oculta se observa que la red

menos profunda se mantuvo con el peor desempeño en todos los experimentos, del mismo modo la más profunda tampoco alcanzó la mayor precisión. En cambio, los modelos con 2 y 4 capas ocultas presentaron mejor desempeño para los experimentos con recall hasta e 1 % y 10 %.

4.2.2.3. Comparación mejores modelos

Se eligen las arquitecturas con mejor desempeño para una misma cantidad de neuronas por capa oculta y se comparan con el desempeño de la arquitectura 2000-128 para determinar cuál presenta el mejor desempeño dentro de los experimentos realizados. Las tablas 4.3 y 4.4 muestran el desempeño para el criterio predictivo con promedio micro y macro, respectivamente.

Arquitectura	Recall 1 %	Recall 10 %	Recall 100 %
2000-250-128	0,4509	2,578	17,16
2000-250-250-128	0,2276	1,375	12,22
2000-250-250-250-128	0,5395	1,410	18,94
2000-250-250-250-250-128	0,2407	1,174	14,09
2000-250-250-250-250-250-128	0,7203	3,768	19,08
2000-500-128	0,1589	2,801	17,19
2000-500-500-128	0,1838	2,198	11,11
2000-500-500-500-128	0,2085	1,917	15,60
2000-500-500-500-500-128	0,4475	2,797	13,79
2000-500-500-500-500-500-128	0,7688	3,142	18,30
2000-1000-128	0,3656	2,801	19,35
2000-1000-1000-128	0,1593	2,040	13,08
2000-1000-1000-1000-128	0,9756	5,651	21,40
2000-1000-1000-1000-1000-128	0,7096	2,845	16,14
2000-1000-1000-1000-1000-1000-128	1,002	4,475	20,62

Cuadro 4.3: F1 score promedio de las arquitecturas para recall hasta el 100 %, 10 % y 1 % en Reuters según el criterio predictivo con promedio micro.

Arquitectura	Recall 1 %	Recall 10 %	Recall 100 %
2000-250-128	0,2342	0,7130	17,77
2000-250-250-128	0,1787	2,184	13,09
2000-250-250-250-128	0,3285	2,111	19,83
2000-250-250-250-250-128	0,4450	2,030	15,62
2000-250-250-250-250-250-128	0,5568	2,368	20,43
2000-500-128	0,2715	0,8188	17,78
2000-500-500-128	0,3572	1,170	12,12
2000-500-500-500-128	0,3868	2,998	16,79
2000-500-500-500-500-128	0,3829	1,777	15,37
2000-500-500-500-500-500-128	0,7681	2,384	19,98
2000-1000-128	0,5775	3,803	20,09
2000-1000-1000-128	0,3215	0,8721	14,32
2000-1000-1000-1000-128	0,9713	4,702	22,78
2000-1000-1000-1000-1000-128	0,8102	2,516	17,87
2000-1000-1000-1000-1000-1000-128	1,168	4,252	22,43

Cuadro 4.4: F1 score promedio de las arquitecturas para recall hasta el 100 %, 10 % y 1 % en Reuters según el criterio predictivo con promedio macro.

Los experimentos a comparar con la arquitectura 2000-128, según recall y criterio predictivo utilizado, son:

1. Micro:

a) Recall 100 %:

- 2000-250-250-250-250-250-128
- 2000-500-500-500-500-500-128
- 2000-1000-1000-1000-128

b) Recall 10 %:

- 2000-250-250-250-250-250-128

- 2000-500-500-500-500-128
- 2000-1000-1000-1000-128

c) Recall 1 %:

- 2000-250-250-250-250-128
- 2000-500-500-500-500-128
- 2000-1000-1000-1000-1000-128

2. Macro:

a) Recall 100 %:

- 2000-250-250-250-250-128
- 2000-500-500-500-500-128
- 2000-1000-1000-1000-128

b) Recall 10 %:

- 2000-250-250-250-250-128
- 2000-500-500-500-128
- 2000-1000-1000-1000-128

c) Recall 1 %:

- 2000-250-250-250-250-128
- 2000-500-500-500-500-128
- 2000-1000-1000-1000-1000-128

Las arquitecturas que presentan un mejor F1 score son aquellas con 5 capas ocultas, con excepción de 2000-1000-1000-1000-128 para recall 100 % y 10 %, para ambos criterios, y 2000-500-500-500-128 para recall 10 % para el criterio predictivo con promedio macro.

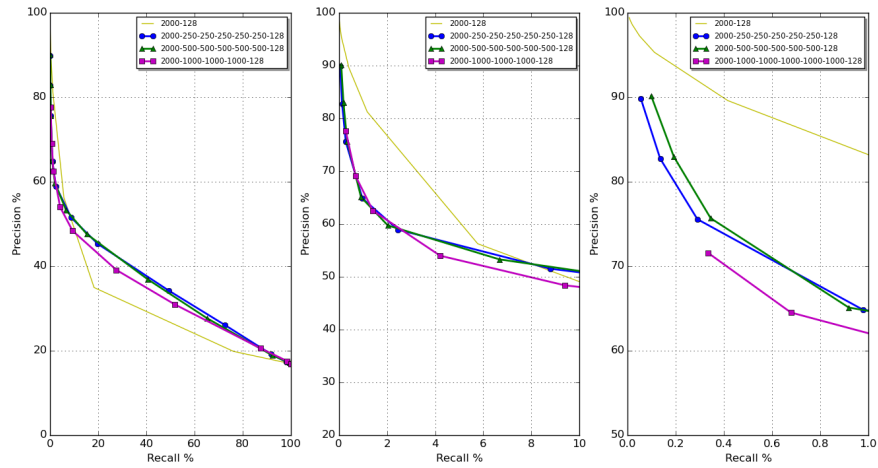


Figura 4.28: Precisión vs recall de arquitecturas con mejor desempeño según el criterio predictivo con promedio micro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

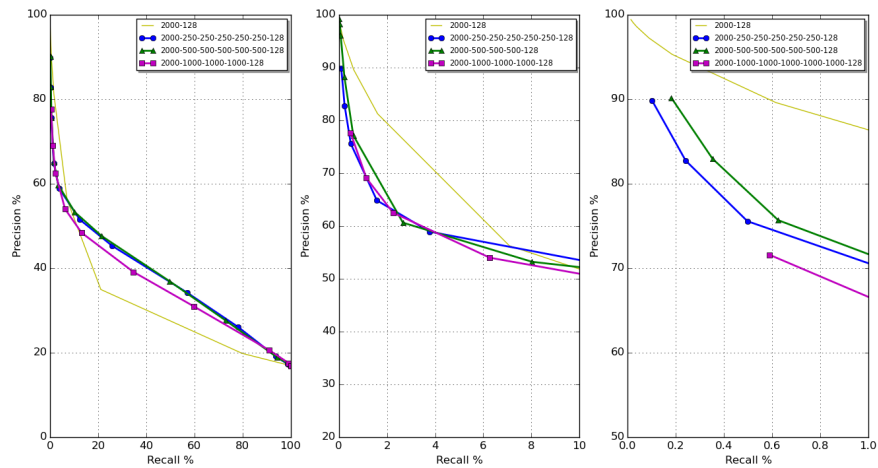


Figura 4.29: Precisión vs recall de arquitecturas con mejor desempeño según el criterio predictivo con promedio macro. Panel izquierdo: Desempeño hasta el 100 % de recall. Panel central: Desempeño hasta el 10 % de recall. Panel derecho: Desempeño hasta el 1 % de recall.

El panel izquierdo de las figuras 4.28 y 4.29 muestra que 2000-250-250-250-250-250-128 y 2000-500-500-500-500-500-128 muestran resultados similares y levemente superiores

que 2000-1000-1000-1000-128, no superan en más de 3 % de precisión. Respecto a 2000-128, se observa que posee la menor precisión para recall superior al 10 %.

El panel central de las figuras 4.28 y 4.29 mantiene el comportamiento similar en las arquitecturas con 250 y 500 neuronas. Cabe destacar que para la figura 4.28 la arquitectura 2000-500-500-500-500-500-128 presenta mayor F1 score, mientras que en la otra se elige 2000-500-500-500-128. Además, la arquitectura 2000-128 tiene una considerable mejora, bajo el 10 % alcanza una mayor precisión respecto a los experimentos elegidos.

En el panel derecho de las figuras 4.28 y 4.29 se utilizan sólo arquitecturas con 5 capas ocultas. Pese a esto, se corroboró que las arquitecturas con 250 neuronas y 500 neuronas mantuvieran un comportamiento similar. Además, el desempeño de la arquitectura 2000-128 se mantuvo superior a los demás experimentos.

Mediante las figuras 4.28 y 4.29 se observó que las arquitecturas con más capas ocultas mantienen un mejor desempeño para la recuperación de muchos documentos. En cambio, bajo el 10 % de recall el modelo sin capas ocultas alcanzó una precisión mayor.

Las tablas 4.5 y 4.6 muestran la precisión de las distintas arquitecturas para los recall utilizados según el criterio predictivo con promedio micro y macro, respectivamente.

Arquitectura	Recall 1 %	Recall 10 %	Recall 100 %
2000-250-128	55,988	42,216	16,911
2000-250-250-128	68,727	49,731	16,911
2000-250-250-250-128	65,329	57,241	16,911
2000-250-250-250-250-128	87,457	59,684	16,911
2000-250-250-250-250-250-128	64,849	51,530	16,911
2000-500-128	66,635	40,303	16,911
2000-500-500-128	85,396	49,192	16,911
2000-500-500-500-128	77,178	53,216	16,911
2000-500-500-500-500-128	70,454	52,501	16,911
2000-500-500-500-500-500-128	65,106	53,303	16,911
2000-1000-128	66,275	45,272	16,911
2000-1000-1000-128	82,396	54,106	16,911
2000-1000-1000-1000-128	69,073	48,386	16,911
2000-1000-1000-1000-1000-128	70,769	55,267	16,911
2000-1000-1000-1000-1000-1000-128	64,522	51,371	16,911

Cuadro 4.5: Precisión para distintos Recall de las arquitecturas entrenadas en Reuters según el criterio predictivo con promedio micro.

Arquitectura	Recall 1 %	Recall 10 %	Recall 100 %
2000-250-128	68,81	55,98	16,91
2000-250-250-128	88,40	49,73	16,91
2000-250-250-250-128	77,66	57,24	16,91
2000-250-250-250-250-128	87,45	59,68	16,91
2000-250-250-250-250-250-128	75,56	58,92	16,91
2000-500-128	66,63	53,95	16,91
2000-500-500-128	85,39	60,32	16,91
2000-500-500-500-128	77,17	53,21	16,91
2000-500-500-500-500-128	86,08	61,70	16,91
2000-500-500-500-500-500-128	75,68	59,73	16,91
2000-1000-128	66,27	45,27	16,91
2000-1000-1000-128	82,99	67,95	16,91
2000-1000-1000-1000-128	77,63	53,99	16,91
2000-1000-1000-1000-1000-128	80,69	60,57	16,91
2000-1000-1000-1000-1000-1000-128	71,58	55,34	16,91

Cuadro 4.6: Precisión para distintos Recall de las arquitecturas entrenadas en Reuters según el criterio predictivo con promedio macro.

Al comparar la precisión según el criterio predictivo con promedio micro (cuadro 4.5) con la precisión según el criterio predictivo con promedio macro (cuadro 4.6) se observa que el promedio macro muestra un desempeño igual o superior al promedio micro.

Capítulo 5

Conclusiones

En base al trabajo realizado y expuesto en esta memoria se presentan las conclusiones en función de los objetivos propuestos.

5.1. Conclusiones

Para este trabajo se implementó un modelo basado en lo propuesto por [28], de modo que sea posible hacer una representación binaria y compacta de documentos de textos que refleje la similitud semántica entre ellos. Esto permite realizar búsqueda por similitud de modo eficiente y escalable, tanto en memoria como en tiempo de búsqueda [11]. Para esto, mediante NumPy se diseñó una *DBN* que reciba un *TF* y lo transforme en una cadena binaria, según el entrenamiento descrito en la sección [2.4.4](#).

Al comparar los resultados de la implementación original con la implementación obtenida en *NumPy*, utilizando el mismo entrenamiento, se observa que este trabajo logra un desempeño similar para *20 Newsgroups* y superior en *Reuters*, esto puede ser efecto de la supresión del ruido utilizado en el baseline. Por ende, se puede concluir que la implementación obtenida es apropiada para resolver el problema propuesto, alcanzando un desempeño similar y superior al baseline.

A continuación, se estudió la sensibilidad de los resultados de la ANN al cambiar la cantidad de neuronas manteniendo constante la cantidad de capas ocultas y cambiando la cantidad de capas ocultas manteniendo constante la cantidad de neuronas. Para ello se compararon modelos de profundidad 1 a 5 y utilizando 250, 500 y 1000 neuronas.

Al comparar los resultados según la cantidad de neuronas por capa oculta se observó que las arquitecturas con 1000 neuronas tienden a mostrar un mejor desempeño, sin embargo al utilizar modelos más profundos los modelos de 250 y 500 neuronas mejoran su desempeño, superando en algunos experimentos el desempeño obtenido por las arquitecturas de 1000 neuronas, un buen ejemplo se observa en los experimentos con 4 capas ocultas, en donde la arquitectura con 250 neuronas alcanza desempeño similar o superior a los demás. Además, para distintos rangos de recall el comportamiento de las arquitecturas no suele variar. Por esto, se concluye que variar la cantidad de neuronas influye levemente en el desempeño de la red, de modo que al cambiar la profundidad se observan mayores diferencias.

Al comparar los resultados según la cantidad de capas ocultas se observa que las arquitecturas más profundas alcanzan el mayor desempeño a medida que aumenta el recall, esto significa que para el objetivo de recuperar una gran volumen de datos suelen ser más precisas las arquitecturas con más capas ocultas. En cambio, en los paneles que muestran el desempeño hasta el 10 % y 1 % se observa que arquitecturas menos profundas pueden alcanzar una precisión similar e incluso superior a redes más profundas. En la mayoría de los experimentos de *20 Newsgroups* como los de *Reuters* muestran que para recall inferior al 10 % las arquitecturas con 5 capas ocultas alcanzan una precisión levemente superior o inferior a la alcanzada en las arquitecturas con 2 o 3 capas ocultas, luego para la medición hasta el 1 % de recall las arquitecturas de 2 capas ocultas pueden alcanzar 10 % más en precisión. Además, la comparación con el modelo sin capas ocultas, 2000-128, muestra que es posible obtener buenos resultados, alcanzar una precisión superior al 60 %, para el 1 % de recall. Variar la cantidad de capas ocultas muestra distinto desempeño, tanto con distinta cantidad de neuronas como en rangos de recall, por ende modificar este parámetro influye en mayor medida en el desempeño de la red.

Estos resultados resultan muy interesantes pues se espera que modelos más profundos posean un mejor desempeño, el problema es que entre más profundo el modelo mayor es

su complejidad computacional y se requiere de hardware más sofisticado para su uso, en este caso se demuestra que modelos menos complejos pueden obtener un buen desempeño. Además, en los escenarios de recuperación de información se suele requerir de una baja cantidad de documentos pero que correspondan con la búsqueda realizada, esto considera aún más los modelos menos profundos pues alcanzan una precisión comparable, inclusive superior, a modelos más profundos.

5.2. Trabajo futuro

Si bien se experimentó cambiando la cantidad de capas ocultas y la cantidad de neuronas por capa oculta estos no son los únicos parámetros que pueden ser modificados al momento de entrenar una *ANN*.

Dentro de los cambios posibles está variar la tasa de aprendizaje (learning rate) con el objetivo de acelerar o disminuir la convergencia del entrenamiento, un learning rate muy alto podría evitar encontrar una solución óptima, en cambio si es muy bajo puede tardar demasiado en encontrar dicha solución. También es posible modificar este parámetro de forma dinámica durante el entrenamiento.

Para este trabajo se entrenó la red por batchs, sin embargo es posible utilizar técnicas online para el entrenamiento, de modo que los pesos se actualicen por cada elemento.

Bibliografía

- [1] Deep belief networks. <http://deeplearning.net/tutorial/DBN.html>, 03-08-2016. Online; revisado 03-08-2016.
- [2] RCV1-v2. http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm, 2004. Online; revisado 03-Julio-2016.
- [3] 20 Newsgroups. <http://qwone.com/~jason/20Newsgroups/>, 2008. Online; revisado 30-Marzo-2016.
- [4] B. Borah and D. K. Bhattacharyya. An improved sampling-based dbscan for large spatial databases. In *Intelligent Sensing and Information Processing, 2004. Proceedings of International Conference on*, pages 92–96, 2004.
- [5] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, September 2001.
- [6] Courtney Corley and Rada Mihalcea. Measuring the semantic similarity of texts. In *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*, pages 13–18. Association for Computational Linguistics, 2005.
- [7] S. R. Fitriyani and H. Murfi. The k-means with mini batch algorithm for topics detection on online news. In *2016 4th International Conference on Information and Communication Technology (ICoICT)*, pages 1–5, May 2016.
- [8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.
- [9] Kristen Grauman. Efficiently searching for similar images. *Commun. ACM*, 53(6):84–94, June 2010.
- [10] Ilkka Havukkala. Ontologies and semantic mining for bio-technology and chemistry data and patents. In *Proceedings of the 2Nd International Workshop on Patent Information Retrieval, PaIR '09*, pages 41–42, New York, NY, USA, 2009. ACM.

- [11] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.
- [12] Prateek Jain, Ambuj Tewari, and Inderjit S. Dhillon. Partial hard thresholding. *IEEE Trans. Information Theory*, 63(5):3029–3038, 2017.
- [13] Yan Ke, Rahul Sukthankar, Larry Huston, Yan Ke, and Rahul Sukthankar. Efficient near-duplicate detection and sub-image retrieval. In *ACM Multimedia*, volume 4, page 5. Citeseer, 2004.
- [14] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.
- [15] Alex Krizhevsky and Geoffrey E. Hinton. Using very deep autoencoders for content-based image retrieval.
- [16] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.
- [17] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081. IEEE Computer Society, 2012.
- [18] Chun-Shien Lu, Chao-Yong Hsu, Shih-Wei Sun, and Pao-Chi Chang. Robust mesh-based hashing for copy detection and tracing of images. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, volume 1, pages 731–734. IEEE, 2004.
- [19] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*, chapter 2, pages 19–47. Cambridge University Press, New York, NY, USA, 2008.
- [20] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, pages 775–780, 2006.
- [21] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.
- [22] David CL Ngo, Andrew BJ Teoh, and Alwyn Goh. Biometric hash: high-confidence face recognition. *IEEE transactions on circuits and systems for video technology*, 16(6):771–775, 2006.
- [23] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization, second edition*. World Scientific, 2006.

- [24] Xinyu Ou, Lingyu Yan, Hefei Ling, Cong Liu, and Maolin Liu. Inductive transfer deep hashing for image retrieval. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 969–972, New York, NY, USA, 2014. ACM.
- [25] Juan Carlos Pérez-Cortes, Rafael Llobet, and Joaquim Arlandis. Fast and accurate handwritten character recognition using approximate nearest neighbours search on large databases. In *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 767–776, London, UK, UK, 2000. Springer-Verlag.
- [26] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*:. Cambridge University Press, Cambridge, 10 2011.
- [27] Ruslan Salakhutdinov and Geoff Hinton. Training a deep autoencoder or a classifier on MNIST digits. <http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html>. Online; revisado 01-Marzo-2017.
- [28] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, July 2009.
- [29] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August 1988.
- [30] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.
- [31] Anshul Sehgal and Uday B Desai. 3d object recognition using bayesian geometric hashing and pose clustering. *Pattern Recognition*, 36(3):765–780, 2003.
- [32] Cyrus Shahabi, Farnoush Banaei-Kashani, Yi-Shin Chen, and Dennis McLeod. Yoda: An accurate and scalable web-based recommendation system. In *International Conference on Cooperative Information Systems*, pages 418–432. Springer, 2001.
- [33] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 750–757. IEEE, 2003.
- [34] Qinfeng Shi, Hanxi Li, and Chunhua Shen. Rapid face recognition using hashing. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2753–2760. IEEE, 2010.
- [35] Benno Stein. Principles of hash-based text retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 527–534. ACM, 2007.
- [36] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

- [37] Tijmen Tieleman and Geoffrey Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1033–1040, New York, NY, USA, 2009. ACM.
- [38] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.
- [39] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on learning to hash. *CoRR*, abs/1606.00185, 2016.
- [40] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - A survey. *CoRR*, abs/1509.05472, 2015.
- [41] Qifan Wang, Dan Zhang, and Luo Si. Semantic hashing using tags and topic modeling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 213–222, New York, NY, USA, 2013. ACM.
- [42] Dan Zhang, Fei Wang, and Luo Si. Composite hashing with multiple information sources. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 225–234, New York, NY, USA, 2011. ACM.
- [43] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 18–25, New York, NY, USA, 2010. ACM.
- [44] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. Preference preserving hashing for efficient recommendation. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 183–192. ACM, 2014.