AdaBnn: Redes Neuronales Binarizadas entrenadas mediante Aprendizaje Estructural Adaptativo

(Estudio experimental)

Wilmer Alberto Gonzalez Sanchez

Escuela de Computación
Facultad de Ciencias
Universidad Central de Venezuela

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela.

Tutores:
Prof. Jesús Lares.
Prof. Haydemar Núñez.

Mayo 2018

Universidad Central de Venezuela
Facultad de Ciencas
Escuela de Computación

Acta del veredicto

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el Bachiller Wilmer Alberto Gonzalez Sanchez C.I.: 24336448, titulado "**AdaBnn: Redes Neuronales Binarizadas entrenadas mediante Aprendizaje Estructural Adaptativo (estudio experimental)**", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, éste fijó el día **24 de mayo**, a las **04:00 pm**, para que su autor lo defendiera en forma pública en el aula **PB-III**, lo cual éste realizó mediante una exposición oral de su contenido, y luego respondió satisfactoriamente a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

Es de declarar que el Profesor Jesús Lares se encuentra fuera del país. Por esta razón, el Profesor Robinson Rivas, director de la Escuela de Computación, firma el presente documento en su lugar.

En fe de lo cual se levanta la presente acta, en Caracas el 28 de mayo de 2018.

pos.

Prof. Jesús Lares (Tutor)

Prof. Haydemar Núñez (Tutor)

Prof. Héctor Navarro
(Jurado Principal)

REPUBLICA DE VENEZUELA
Facultad de Ciencias
Escuela de Computación
Universidad Central - Caracas

Prof. Eugenio
(Jurado

Agradecimientos

Gracias a los profesores Haydemar Nuñez y Jesus Lares, por su asesoría y tutoría durante el desarrollo de esta investigación y durante la carrera.

Gracias a los profesores Fernando Crema y Eugenio Escalise por su disposición y apoyo en la aclaratoria de temas académicos.

Gracias a mi familia y amigos por el apoyo brindado.

Dedicado a mi abuela Belén Blanco y a mis padres Wilmer Gonzalez y Rosa Sanchez.

Resumen

En la actualidad, cada vez más productos de software, como aplicaciones móviles o web, hacen uso de interfaces que procesan datos en forma de audio, imágenes, videos y texto, así mismo, la incorporación de inteligencia artificial en estos productos es también incrementalmente frecuente. En ese sentido, las redes neuronales profundas han mostrado un desempeño sobresaliente en tareas como reconocimiento de voz [8], clasificación de imágenes[18, 15], clasificación de documentos[12], entre otras. Sin embargo, el proceso de implementación de estos algoritmos conlleva ciertas consideraciones técnicas, por ejemplo, una iterativa calibración de parámetros cómo la profundidad de la red y la cantidad de neuronas en cada capa, a manera de ensayo y error, así como, grandes cantidades de memoria, accesos y amplios tiempos de ejecución para su entrenamiento y uso. En el presente trabajo de investigación se diseñaron, implementaron y compararon redes neuronales profundas concebidas mediante aprendizaje estructural adaptativo y cuyos pesos fueron restringidos al conjunto $\{-1, 1\}$, cómo una forma de disminuir las consideraciones tecnicas mencionadas, dichas redes neuronales fueron evaluadas en el problema de clasificación de imágenes tanto en su configuración binaria cómo multiclase, los resultados de dichos experimentos son expuestos junto a un análisis de desempeño.

Índice general

Índice de figuras

Capítulo 1

Introducción

En la actualidad, cada vez más productos de software, como aplicaciones móviles o web, hacen uso de interfaces que procesan datos en forma de audio, imágenes, videos y texto, así mismo, la incorporación de inteligencia artificial en estos productos es también incrementalmente frecuente. En ese sentido, las redes neuronales profundas han mostrado un desempeño sobresaliente en tareas como reconocimiento de voz [8], clasificación de imágenes[18, 15], clasificación de documentos[12], entre otras. Sin embargo, el proceso de implementación de estos algoritmos conlleva ciertas consideraciones técnicas, veamos algunas:

- **Estos algoritmos requieren una iterativa calibración de parámetros cómo la profundidad de la red y la cantidad de neuronas en cada capa, a manera de ensayo y error:** En este sentido, **AdaNet** de *Cortes et al*[1] presenta un análisis teórico que contempla un escenario de aprendizaje supervisado, en el cual, la topología y los parámetros de una red neuronal son aprendidos simultáneamente, a partir de los datos. Dicho aprendizaje también es parametrizado, pero la configuración de estos parámetros se decide en función a la complejidad del modelo deseado para resolver el problema ajustando aspectos cómo la cantidad de neuronas que se agregarán por vez y la cantidad de iteraciones límite mediante las cuales puede entrenarse el algoritmo, que finalmente también definirá la profundidad tope de la red neuronal.

- **Estos algoritmos requieren grandes cantidades de memoria y accesos, así como, amplios tiempos de ejecución para su entrenamiento y uso:** A este problema, *Courbariaux et al*, con las redes neuronales binarizadas[11] han propuesto la definición de ciertas restricciones sobre el espacio en el que están definidos los pesos entre las conexiones de las neuronas y de los valores resultantes de las funciones de activación.

De esta forma, la habilitación de operaciones con un tipo de dato más ligero, binario, podría reducir la cantidad de memoria requerida por la red neuronal.

Con el objetivo de satisfacer ambas consideraciones se plantea la siguiente interrogante: ¿Es posible construir un algoritmo de red neuronal profunda para problemas de clasificación cuyos parámetros sean producto de una implementación del análisis teórico planteado en *AdaNet*[1] y que, adicionalmente, posea una restricción en los valores posibles para los pesos y el resultado de la aplicaciones de funciones de activación(*redes neuronales binarizadas*[11]), con el fin de habilitar el uso de dicho algoritmo en máquinas de bajas prestaciones y mejorar la efectividad en las tareas de clasificación para las que se use dicho modelo? Las siguientes páginas, desarrollarán los elementos suficientes para responder esta pregunta.

1.1 Objetivo general

Implementar un modelo de red neuronal binarizada entrenada mediante aprendizaje estructural adaptativo y evaluar dicha implementación en el problema de aprendizaje supervisado, planteado sobre los conjuntos de datos **CIFAR-10** y **MNIST**.

1.2 Objetivos específicos

- Recopilar información de previas implementaciones que hayan integrado los conceptos de aprendizaje estructural adaptativo (**AdaNet**[1])y binarización en los pesos de redes neuronales profundas(**Redes neuronales binarizadas**[11]).
- Implementar un algoritmo de redes neuronales que instancie los principios teóricos planteados en **AdaNet**[1].
- Diseñar un modelo de redes neuronales que integre aprendizaje estructural adaptativo y binarización en los pesos.
- Implementar un modelo de redes neuronales que integre aprendizaje estructural adaptativo y binarización en los pesos.
- Comparar el desempeño general y específico del modelo creado con algunos algoritmos de redes neuronales previamente usados en el problema de aprendizaje planteado sobre los conjuntos de datos **CIFAR-10** y **MNIST**.
- Proveer el código suficiente y necesario tanto para replicar los experimentos planteados, como para hacer uso del modelo desarrollado, fuera de esta investigación.

1.3 Justificación

El desarrollo de este proyecto propone explorar la alternativa de binarización de pesos con el fin de disminuir las limitaciones técnicas al usar algoritmos basados en redes neuronales profundas al mejorar tiempos de entrenamiento y predicción, integrando el concepto de aprendizaje estructural adaptativo sobre redes neuronales, para disminuir la intervención humana en el uso de las redes neuronales, así como incorporar la habilidad de establecer la complejidad de modelo usado a partir de los datos de entrenamiento.

1.4 Organización del documento

Las siguientes secciones de este documento están organizadas en capítulos, como sigue:

Capítulo 2 **Marco Teórico:** En este capítulo, se describe el fundamento teórico necesario para asegurar el entendimiento del trabajo realizado, en este sentido, trata sobre aprendizaje automático y más específicamente en redes neuronales profundas, así como, la teoría relacionada con los problemas de aprendizaje destinados a la evaluación del modelo desarrollado.

Capítulo 3 **Marco Metodológico:** En este capítulo, se describe la forma en que se desarrolló el modelo AdaBnn y el proceso de evaluación del modelo desarrollado.

Capítulo 4 **Algoritmo propuesto: AdaBnn**: En este capítulo, se describen las características del modelo **AdaBnn** y la necesaria implementación de **AdaNet** utilizada..

Capítulo 5 **Experimentos:** En este capítulo, se incluyen la configuración y los resultados de cada uno de los experimentos realizados con los algoritmos desarrollados, sobre los conjuntos de datos **CIFAR-10** y **MNIST**, así como las observaciones derivadas de dichos experimentos.

Capítulo 6 **Conclusiones:** En este capítulo, se consolidan las conclusiones de la presente investigación, se sumarizan los principales hallazgos y los posibles trabajos futuros.

Capítulo 2

Marco teórico

En esta sección, se incluyen los conceptos que proveen el sustento teórico y técnico de las actividades realizadas en favor de la consecución de los objetivos planteados por esta investigación. Iniciando con el área general de conocimiento que alberga los lineamientos que rigen la optimización de redes neuronales, incluyendo los estudios que explican, por separado, las modificaciones que integran el algoritmo propuesto. Adicionalmente, se incluyen los trabajos previos relacionados a los casos de uso seleccionados, en los cuales se evaluará el algoritmo propuesto, así mismo se incluyen las definiciones de las principales tareas de preprocesamiento frecuentemente realizadas sobre el caso de uso.

2.1 Aprendizaje automático

Las redes neuronales son un subconjunto de técnicas contenidas en el aprendizaje automático, por ello es importante considerar los siguientes conceptos asociados a este último. Convendremos en definir aprendizaje automático cómo:

> *El conjunto de métodos que permiten detectar patrones en datos y luego usarlos para predecir futuros datos o tomar decisiones bajo incertidumbre[22].*

Este conjunto de métodos, suele organizarse en fases, típicamente:

- **Recolección de datos:** En esta fase, se definen los mecanismos de obtención o generación de los datos sobre los cuales se desea efectuar el aprendizaje y las variables que serán representadas del fenómeno modelado. La importancia que recae sobre esta etapa, surge de la correspondencia que se espera encontrar entre el fenómeno existente y el modelo realizado de dicho fenómeno, permitiendo a su vez, llevar el aprendizaje realizado a la toma de acciones en el ámbito en el cual existe el fenómeno.

- **Preprocesamiento de los datos:** En esta fase, se adapta, cualquier decisión tomada al momento de realizar la recolección de datos, a la estructura requerida para alimentar los algoritmos de aprendizaje que se integrarán en las siguientes fases.

- **Modelado predictivo o inferencial de los datos:** En esta fase, se define el tipo de modelado que se realizará en función de las variables y observaciones adquiridas. Utilizar un subconjunto de variables predictoras para estimar el valor de otra subcojunto, de cardinalidad uno generalmente, de variables denominadas variables objetivo para observaciones previamente desconocidas se conoce como modelado predictivo, analizar la relación subyacente en el comportamiento de las variables predictoras entre ellas y luego con respecto a las variables objetivo se conoce como modelado inferencial.[31]

- **Validación del modelo:** En esta fase, se consideran los parámetros del modelo utilizado, en función de mejorar alguna medida de desempeño en la tarea de aprendizaje definida. Esta etapa, requiere un conjunto de datos que permitan apreciar cómo estas variaciones influyen en el tratamiento de datos nuevos para el modelo.

- **Evaluación del modelo:** En esta fase, se evalúa el desempeño del modelo, afinado con el proceso de validación, en un subconjunto de datos, en este caso no se realizan cambios, con respecto a dicho desempeño, dado que ésto, podría consecutir en el sobreajuste del modelo sobre los datos usados y puede no representar un modelo del fenómeno general.

El aprendizaje automático puede tener distintos tipos de retroalimentación en función de los objetivos planteados para el aprendizaje, dado que nos interesa proveer conclusiones comprobables y replicables en distintos problemas de aprendizaje, la presente propuesta se enfocará en las técnicas contenidas en el aprendizaje automático conocidas como aprendizaje supervisado.

Aprendizaje supervisado

En este tipo de aprendizaje, el objetivo es encontrar un patrón en la correspondencia de los valores de entrada X a ciertos valores de salida Y, disponiendo de un conjunto de pares etiquetados $D = \{x_i, y_i\}_{i=1}^{N}$ llamado **conjunto de entrenamiento**, siendo N el número de observaciones ejemplo y cada x_i representa un vector m-dimensional, en el caso general, x_i puede representar objetos más complejos, como imágenes, entre otros. De la misma forma, y_i, también llamada variable objetivo, se trata de formas diferentes dependiendo de sus características, en el caso de que y_i sea una variable nominal de un conjunto finito, esto es,

$y_i \in \{1,...,C\}$ el problema se denomina de **clasificación**, si y_i está definido en un espacio de valores continuos, el problema se denomina **regresión**. En general, el aprendizaje automático de tipo supervisado consiste en la generación de un modelo que use los componentes mencionados para predecir la variable objetivo en observaciones previamente desconocidas. Es importante tomar en cuenta que la capacidad de los algoritmos de proveer un mejor rendimiento en los datos de prueba habiendo realizado ajustes en base a los resultados en el conjunto de entrenamiento, es posible gracias a que se asume que el proceso de generación de los datos (tanto el conjunto de entrenamiento como el de prueba) posee una distribución independiente e idéntica para cada observación posible.

Otros conceptos de utilidad al momento de considerar los algoritmos de aprendizaje supervisado que veremos, son:

- **Generalización**: Es la capacidad que tienen los algoritmos de aprendizaje para predecir correctamente nuevas datos de entrada.
- **Sobreajuste**: Es un estado de los algoritmos de aprendizaje, en el cual sólo se predicen correctamente los datos previamente vistos por el modelo y que disminuye la generalización del mismo.
- **Regularización**: Es un proceso que mejora la generalización de los algoritmos de aprendizaje, usando información adicional que penalice, por ejemplo, la complejidad del modelo usado.

Otro concepto utilizado en esta investigación es la noción de intervalo de confianza

Entre las técnicas contenidas en el aprendizaje automático, esta investigación se centrará en las redes neuronales, especialmente en su uso dentro del aprendizaje supervisado.

2.1.1 Redes neuronales

La abstracción de menor nivel que compone las redes neuronales son las neuronas (también llamadas unidades o nodos), en el aprendizaje automatizado, son abstracciones de procesamiento expresadas, por ejemplo, mediante modelos matemáticos[33, 20], conformados por:

Componentes de cada neurona

Valores de entrada:

Un conjunto de valores numéricos representados en un vector, que representa la interfaz de la neurona con el problema de aprendizaje.

Vector de pesos:

Un conjunto de valores numéricos que determinan la ponderación de cada entrada de cada neurona y cuyos pesos se ajustan, con el fin de mejorar el rendimiento de la red con respecto a la función objetivo.

Sumador:

Un valor h resultante de la sumatoria de los productos entre el vector de valores de entrada y el vector de pesos.

Función de activación:

Esta función representa el criterio de activación de la neurona tomando en cuenta el resultado obtenido por la función sumadora, típicamente está representada por una funcion definida a trozos. La escogencia de esta función se realiza a nivel de toda la red neuronal, o a menor grado, a nivel de capas, algunas funciones de activación conocidas son incluidas en la tabla 2.1, con gráficas definidas en el dominio {-4,4} y rango {-1,1}.

Según el modelo propuesto por *McCulloch* y *Pitts* (Figura 2.1), en la inferencia de una neurona, las entradas x_i se multiplican por los pesos w_i y se suman sus valores, si la suma es mayor al valor θ, la neurona se activa.

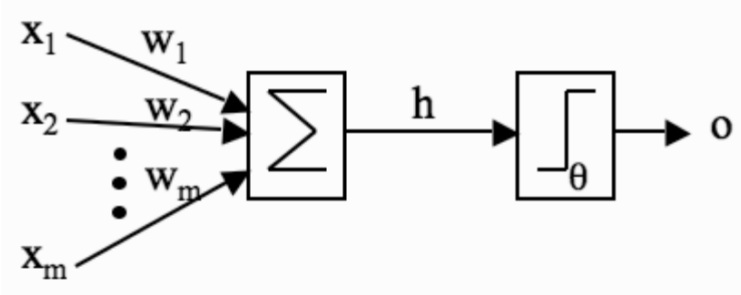


Fig. 2.1 Modelo matemático de una neurona (*perceptrón*)[33]

Finalmente, la función de salida de la neurona puede ser expresada como sigue:

$$f(x_i, w_i) = g(\sum_{i}^{n}(w_i \cdot x_i)) \tag{2.1}$$

El encadenamiento de las neuronas descritas previamente, forma las estructuras llamadas **redes neuronales**, cuyo mecanismo de inferencia y entrenamiento será explicado en detalle en (2.1.1). A las propiedades del encadenamiento de las neuronas la denominaremos **Topología**. Las topologías admiten diferentes formas, sin embargo, pueden nombrarse propi-

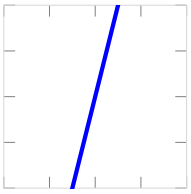
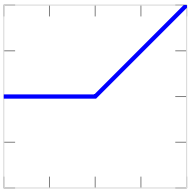
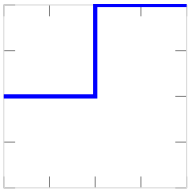
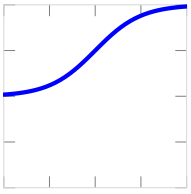
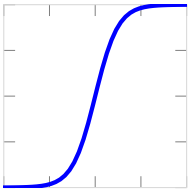
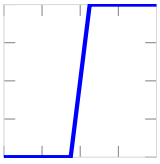
Función de activación	Definición	Gráfica
Lineal	$\Phi(x) = x$	
Unidades lineales rectificadas	$\Phi(x) = max(0,x)$	
Escalonada	$\Phi(x) = \begin{cases} 1 & \text{si } x \geq 0.5, \\ 0 & \text{si } x < 0.5 \end{cases}$	
Sigmoide	$\Phi(x) = \frac{1}{1+e^{-x}}$	
Tangente hiperbólica	$\Phi(x) = tanh(x)$	
Tangente hiperbólica ajustada	$\Phi(x) = max(-1,min(1,x))$	

Table 2.1 Algunas funciones de activación

amente distintos roles de las neuronas en las topologías conocidas, según su función en la red neuronal[9].

Tipos de neuronas

- **Neuronas de entrada**: Reciben los datos de entrada de la red y están ubicadas en una capa determinada **capa de entrada**, en esta capa deben haber tantas neuronas como elementos posea el vector de valores de entrada a la red, estas neuronas no necesitan computar ninguna función de activación, en su lugar, transmiten directamente los valores recibidos que generalmente son valores reales.

- **Neuronas de salida**: Transmiten los valores de salida de la red neuronal y están ubicadas en una capa determinada **capa de salida**, cada neurona transmite un valor de salida, *i.e.* habrán tantas salidas de la red neuronal como neuronas en la capa de salida, en el caso de redes neuronales usadas para problemas de clasificación las neuronas de salida se interpretan como el puntaje de pertenencia a la clase que represente dicha neurona (se requerirían una neurona por clase posible), mientras en el caso de los problemas de regresión se tiene generalmente una neurona en esta capa.

- **Neuronas ocultas**: Reciben como valores de entrada las salidas de otras neuronas en la red, bien sean, otras neuronas ocultas o neuronas de entrada, estas neuronas conforman una o múltiples capas llamadas *capas ocultas*, en esta capa se implementan las funciones de activación que permiten realizar el aprendizaje.

- **Neuronas de sesgo**: Estas neuronas proveen un mecanismo de mejora de la generalización de las redes neuronales, incorporando un valor fijo, que no dependerá de los valores de entrada, en consecuencia estas neuronas no poseen valores de entrada, sólo generan valores de salida hacía otras neuronas, aunque existen topologías que no hacen uso de estas neuronas, cuando éstas son incorporadas, se ubican una neurona de sesgo por cada capa oculta o de salida.

- **Otras neuronas**: Existen otros tipos de neuronas de las que se hacen uso en topologías mas especializadas-como las neuronas de contexto-que serán explicadas en sus respectivas implementaciónes, sin embargo la descripción general de dichos nodos se encuentra fuera del alcance de este estudio.

Funciones objetivo

Las funciones objetivo, también llamadas funciones de error, funciones de costo cuando se define el problema como una minimización de parámetros, proveen un mecanismo de medición del desempeño de la red neuronal en el problema de aprendizaje, esto es, modelar el comportamiento subyacente en los datos, sin aprenderlos de forma individual desde los datos de entrenamiento. En el caso general, podemos hablar de la probabilidad condicional

$p(y,x)$ donde x representa el vector de entrada, y representa el vector de salida y ambos permiten la siguiente descomposición:

$$p(y,x) = p(y|x)p(x) \tag{2.2}$$

Adicionalmente, las funciones de costo se basan en el principio de *máxima verosimilitud* que se puede expresar usando (2.2) como sigue:

$$\mathfrak{L} = \prod_n p(x^n,t^n) = \prod_n p(t^n|x^n)p(x^n) \tag{2.3}$$

Donde, $\{x^n,t^n\}$ representan los datos de entrenamiento y cada observación se asume como independiente pero derivada de la misma distribución. Finalmente, en lugar de maximizar la probabilidad asociada a (2.3), generalmente, se minimiza la función $-ln(\mathfrak{L})$ dado que $-ln()$ es una función monótona decreciente, como sigue:

$$E = -ln(\mathfrak{L}) = -\sum_n ln(p(t^n|x^n)) - \sum_n ln(p(x^n)) \tag{2.4}$$

Sin embargo, dado que x^n es independiente de los parámetros de la red neuronal, nuestro deseo es minimizar:

$$E = -ln(\mathfrak{L}) = -\sum_n ln(p(t^n|x^n)) \tag{2.5}$$

Seguidamente, la selección de la función de costo apropiada esta determinada por los conocimientos iniciales de la distribución condicional $p(t|x)$.

Regularización

Son todas las modificaciones que imponemos sobre un algoritmo de aprendizaje con el objetivo de mejorar su capacidad de generalización pero no necesariamente su desempeño en los datos de entrenamiento. Estos métodos pueden representar conocimiento previo, o establecer una preferencia por modelos con ciertas carácteristicas.

Proceso de entrenamiento de redes neuronales

Al iniciar el entrenamiento de una red neuronal, el *vector de pesos* $\hat{W}$ es inicializado con valores aleatorios –otras técnicas de inicialización de parámetros serán presentadas en la sección 2.1.1.1– luego se transmiten las observaciones x_i a la red, una por una o por lotes, a menos que sea empleada alguna de las estrategias planteadas en sección 2.1.1.1. Tras cada imputación, los pesos se actualizan para minimizar una función de error definida sobre y_i que llamaremos E en el caso general, la función de actualización de pesos está definida como

sigue:

$$\hat{W}^{t+1} = \hat{W}^t + \eta \frac{\partial E(\hat{W})}{\partial \hat{W}} \tag{2.6}$$

Donde, el parámetro η representa la **tasa de aprendizaje**, que regula el tamaño de los cambios que se hacen en los pesos para disminuir el error, un gran valor para η generará grandes cambios rápidamente, pero puede llegar a soluciones subóptimas, mientras que un pequeño valor de η proveerá soluciones cercanas a óptima pero puede tardar mucho más tiempo en converger. Por todo esto, es conveniente inicializar un valor relativamente grande de η para las primeras iteraciones y decrementarlo gradualmente, en la sección 2.1.1.1 veremos algunas formas de implementar esta estrategia. La equación de actualización de pesos (2.6) se deriva del concepto de **descenso de gradiente** que consiste en usar la derivada de la función empírica generada por el vector de pesos para disminuir la función de costo.

Los algoritmos más sencillos iteran sobre las observaciones ajustando los pesos cada vez, hasta conseguir la convergencia de los pesos. Cada iteración sobre todas las observaciones se denomina **época**.

El uso de las redes neuronales requiere, generalmente, un uso intensivo de recursos computacionales, por ello nos enfocaremos en redes neuronales con ciertas restricciones que permitan reducir dichos requerimientos, así como, los tiempos de espera. Sin embargo, es justamente la capacidad de satisfacer estos requerimientos computacionales, la que ha permitido su resurgimiento en tareas cómo clasificación de imágenes, entre otras.

2.1.1.1 Redes neuronales profundas

Son aquellas que poseen una gran cantidad de capas ocultas, estas redes neuronales están motivadas por el *teorema de aproximación universal* [3] el cual establece que una red neuronal hacia adelante con una capa oculta y un número finito de neuronas, puede aproximar funciones continuas estableciendo restricciones sobre las funciones de activación y contando con el número adecuado de neuronas en la capa oculta. Sin embargo, este teorema no especifica que tantas neuronas sean necesarias para aproximar la función deseada y en muchos casos puede volverse ineficiente entrenar este tipo de algoritmos, justificando así, la agregación de capas ocultas que reducen el número de neuronas necesitadas por cada capa, reduciendo a su vez el error de generalización. Otra de las razones que favorece el uso de redes neuronales profundas, es el entendimiento de la composición del problema de aprendizaje en cuestión, esto es, la secuencia de pasos mediante las cuales una computadora puede entender el fenómeno presentado, en el caso de arquitecturas profundas, se asume que la función que queremos aproximar es accesible desde un conjunto de pasos incrementales

que se componen en pasos más sencillos y que generan salidas intermedias útiles para la red neuronal, pero no necesariamente para el entendimiento humano.

Técnicas de optimización

De todas los aspectos que involucran técnicas de optimización, esto es, todas las operaciones que permiten encontrar los argumentos que maximizan o minimizan una función dada, en las redes neuronales profundas, el entrenamiento de dichas redes resulta uno de los problemas más complejos de resolver, veamos a continuación algunas técnicas empleadas en el entrenamiento de redes neuronales profundas.

Descenso estocástico de gradiente

El método de descenso de gradiente requiere un tiempo de entrenamiento mejorable, cuando éste es aplicado sobre un conjunto de datos de entrenamiento lo suficientemente grande. Una solución a este problema es el muestreo de un conjunto de observaciones de los datos de entrenamiento, sobre los cuales se calcula el descenso de gradiente, a esta mejora se le conoce como *descenso estocástico de gradiente*.

Inicialización de parámetros

La resolución de los problemas de optimización encontrados en las redes neuronales profundas, puede depender de la correcta inicialización de los parámetros en el entrenamiento del algoritmo, mas aún, la definición de estos valores puede determinar la velocidad de la convergencia del aprendizaje. A este aspecto se han suministrado distintas estrategias heurísticas, dado que el problema de la optimización de las redes neuronales profundas, sigue siendo un área de investigación activa con mucho que descubrir. Una de las heurísticas más asentada entre las implementaciones documentadas en la literatura, es la inducción de asimetría en distintas neuronas ocultas que posean las mismas entradas y mismas funciones de activación.

Vector de pesos

En cuanto a la inicialización de los pesos de la red, en general, suelen usarse muestreos derivados de distribuciones *Gaussianas* o *Uniformes*. La escala de los pesos también influye en ciertas propiedades de la red neuronal, al pertenecer a una escala mayor por ejemplo, los pesos suelen ser menos redundantes.

Una de las heurísticas más difundidas para seleccionar los valores pertenecientes al vector de pesos para la conexión entre capas totalmente conectadas, es tomar los valores de una

muestra de la expresión:

$$W_{i,j} \approx U(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}) \tag{2.7}$$

Conocida como *inicialización normalizada*[5]. Sin embargo, la inicialización de los pesos suele también considerarse un parámetro del modelo, cuando los recursos computacionales lo permiten. La escala inicial de estos valores también puede establecerse al analizar la desviación estándar de las funciones de activación resultante en un subconjunto de los datos de entrenamiento, por ejemplo, si los pesos en las capas iniciales son muy pequenõs puede disminuirse la señal que se transmita a través de las siguientes capas, por lo que resulta conveniente aumentar la escala. Este ajuste suele llevarse a cabo sobre el conjunto de datos de validación.

Neuronas de sesgo

Tomando en cuenta que la estrategia de definición de los valores de las neuronas sesgo debe estar ligada a la estrategia usada en la inicialización de los pesos, es importante considerar que en algunas situaciones las neuronas sesgo pueden habilitar o inhibir el impacto de ciertas neuronas en la red neuronal, en estos casos puede ser conveniente que las neuronas sesgo habiliten la participación de las neuronas ocultas, con el objetivo de que esta pueda *aprender* a ajustar su peso.

Otra de los mecanismos para definir los valores iniciales de estos y otros parámetros requeridos por las redes neuronales, es aplicar aprendizaje no supervisado sobre el mismo conjunto de datos de entrenamiento y aprender aproximaciones útiles que faciliten la convergencia de la red.

Tasa de aprendizaje adaptativo

La tasa de aprendizaje es uno de los parámetros con mayor influencia en el desempeño de una red neuronal, en este sentido, puede ser necesario dividir el problema de definir una tasa de aprendizaje general, por una tasa de aprendizaje particular a cada etapa de la red neuronal. El algoritmo *delta-bar-delta*[27] fué uno de los primeras heurísticas para adaptar las tasas de aprendizaje individuales de las neuronas durante el entrenamiento y consiste en aumentar la tasa de aprendizaje cuando la derivada parcial de la función de costo, en cuanto al parámetro en cuestión, mantiene el mismo signo y disminuir la tasa de aprendizaje cuando dicho valor cambia de signo, sin embargo, este tipo de optimización sólo puede realizarse cuando el entrenamiento se realiza sobre un conjunto completo de datos, esto es, todos las observaciones pertenecientes al conjunto de entrenamiento.

Uno de los métodos de optimización de tasa de aprendizaje adaptativo es el algoritmo *Adam*[16] que esencialmente, usa el primero y segundo momento de la función de costo, siendo esta definida y diferenciable sobre un conjunto de parámetros θ.

Estrategias generales de optimización

Existen algunas marcos generales que permiten generar implementaciones que representan formas de optimización, en ese sentido consideremos:

Normalización por lotes

Uno de los problemas que se encuentran al implementar redes neuronales profundas es llamado *Cambio covariable interno*[24] y se refiere al cambio de dominios que ocurre entre las capas de la red al momento de entrenar el modelo, esto es, cuando se ajustan los pesos hacia atrás se acarrean actualizaciones en diferentes dominios por capa. A este problema la *normalización por lotes*[14] propone normalizar cada lote con el que se entrene la red neuronal, en el proceso de entrenamiento, esta normalización se realiza con los estimadores muestrales:

$$\sigma_B^2 = \frac{1}{m}\sum_i^m (x_i - \mu_B)^2 \tag{2.8}$$

Al probar la normalización la inferencia en la red neuronal se realiza con el estimador insesgado:

$$Var[x] = \frac{m}{m-1} E_B[\sigma_B^2] \tag{2.9}$$

Descenso por coordenada

En algunos casos, los problemas de optimización planteados sobre el conjunto de variables existentes pueden dividirse en subconjuntos de las variables e incluso en una variable por vez, a este proceso se le conoce como *descenso por coordenada*, aunque frecuentemente se interpreta como la optimización a nivel de un grupo de variables por vez(*descenso por bloque*). A pesar de esto, el *descenso por coordenada* no resulta apropiado cuando existe una fuerte interacción entre las variables con respecto a las variables objetivo.

Aumento de datos

Se refiere a los métodos que permiten construir una introducción gradual, iterativa y oportuna de observaciones o variables nuevas, con el fin de extender la usabilidad de los conjuntos de datos a disposición.[32]

2.1.1.2 Redes neuronales convolucionales

La convolución es una operacion entre dos funciones cuyos valores de entrada pertenecen a $\Re$, esta operación materiliza una evaluación de una función inicial f y una función de pesos w sobre los cuales está definida una noción de orden (o tiempo) específicamente t como parámetro de entrada, así, tenemos:

$$s(t) = \int x(a)w(t-a)da \tag{2.10}$$

En el caso de tener las medidas de tiempo en un espacio discreto:

$$s(t) = \sum_{\infty}^{a=-\infty} x(a)w(t-a) \tag{2.11}$$

Donde a representa el momento de la medida. En general, esta función se denota como:

$$s(t) = (x * w)(t) \tag{2.12}$$

En el contexto de redes neuronales, la función x se asocia con los valores de entrada y la función w se conoce como *núcleo*.

La convolución discreta puede ser vista como una multiplicación entre matrices (véase la Figura 2.2), sin embargo, existen ciertas restricciones sobre la matriz que corresponde con una convolución, por ejemplo, la mayoría de sus entradas tienen valores iguales a cero (matriz esparcida) dado que el *núcleo* es idealmente menor que los datos de entrada. En general, las redes neuronales que no dependen de propiedas particulares en la forma de la matriz generada por los datos de entrada pueden incorporar convoluciones sin cambios adicionales.

Las redes neuronales convolucionales, aquellos que incorporan capas de convoluciones, son compuestas por tres etapas, la transformación de la convolución, la evaluación de las funciones de activación y finalmente, una **función de agrupación**, al momento de especificar los parámetros de una convolución es necesario mencionar el tamaño del filtro y la magnitud del desplazamiento a partir del cual se tomará el siguiente filtro, véase la Figura 2.3.

Funciones de agrupación

Las funciones de agrupación reemplazan una salida particular de la red por indicadores estadísticos de las salidas alrededor, por ejemplo, **maxpooling** [23] consiste en tomar el mayor valor existente en cada región rectangular de los datos de entrada. A pesar de las

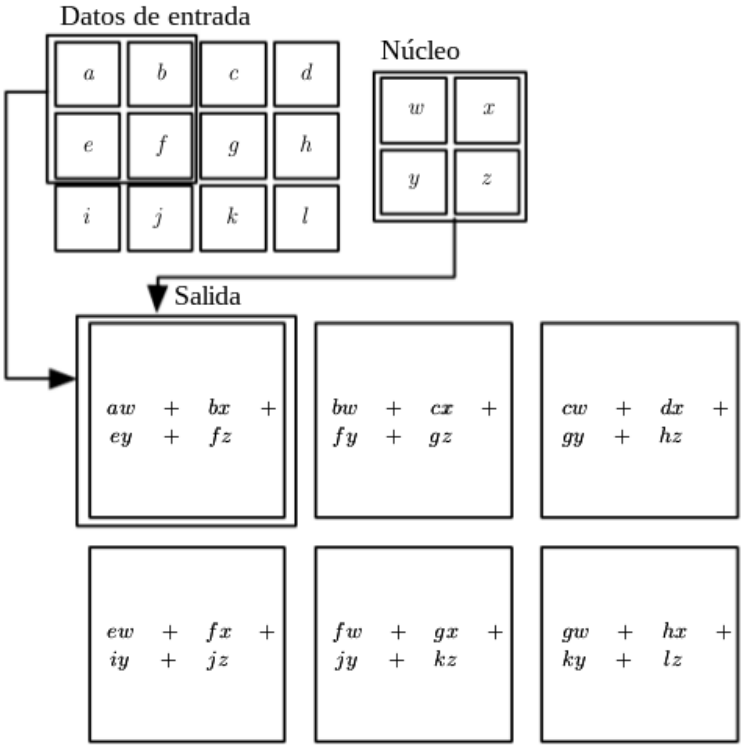


Fig. 2.2 Ejemplo de convolución en $\Re^2$

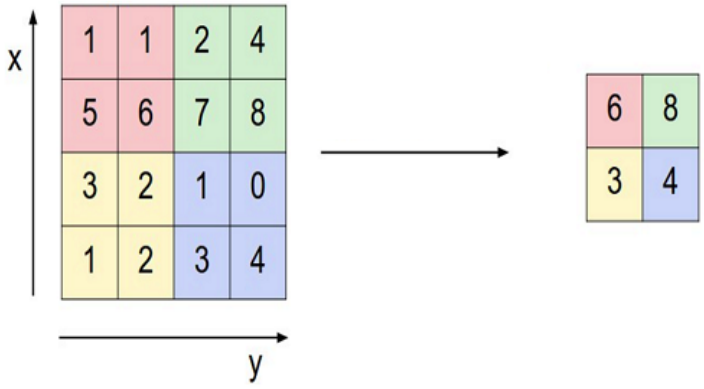


Fig. 2.3 Convolución con *Maxpool* sobre filtro de 2x2 y paso de 2 entradas

posibles diferentes formas de agrupación, todas apuntan a generar cierto margen de invarianza con respecto al traslado de los datos.

Una de las consideraciones necesarias al usar operaciones de convoluciones con funciones de agrupación, es que puede causar subajuste si el tipo de problema no coincide con el enfoque

de localidad del cual se aprovechan estas mejoras. Adicionalmente, estos modelos sólo deben contrastarse entre sí, en cuanto a su desempeño, dada la forma en la que estos algoritmos tratan la información estructural inherente en el conjunto de datos.

El avance en el desarrollo de componentes de hardware más potentes ha habilitado el uso de implementaciones de redes neuronales más complejas, principalmente por la complejidad de los patrones que se encuentran en los cuantiosos datos que se le suministran a estos algoritmos.

La siguiente implementación se presenta en función de proveer un entendimiento general de las prácticas modernas en cuanto a la aplicación de redes neuronales profundas a problemas de aprendizaje. Esta área de aplicación se ha seleccionado dado que implica un enfoque en el modelo utilizado y pocas actividades estrechamente relacionadas con el dominio del problema (y por ende poco generalizables en otros contextos) a diferencia de, por ejemplo, algoritmos dedicados a reconocimiento de voz[10].

2.1.1.3 Redes neuronales binarizadas

Son redes neuronales cuyo espacio de valores posibles tanto para los vectores de pesos, como para los valores que pueda tomar la función de activación pertenecen a un espacio $\mathbb{B}$, donde:

(i) $\mathbb{B} \subset \mathbb{N}$ **(ii)** $|\mathbb{B}| = 2$ **(iii)** $\forall b_1, b_2 \in \mathbb{B}, b_1 \neq b_2$

Especialmente nos concentraremos en aquellas redes neuronales binarizadas en el espacio $\mathbb{B} = \{-1, 1\}$, habilitando así, el uso de operaciones a nivel de bits.

Las redes neuronales pueden considerarse como un conjunto de valores de entrada multiplicados y acumulados mediante diferentes metodologías, sin embargo, al restringir el espacio de los valores a valores en $\mathbb{B}$, las operaciones se simpifican en adiciones (y sustracciones).

Componentes

Funciones de binarización

Para transformar los valores en $\Re$ manejados en el entrenamiento de las redes neuronales se consideran las siguientes funciones de binarización:

Determinística:

$$x^b = \begin{cases} 1 & \text{if } x \geq 0, \\ -1 & \text{de otra forma} \end{cases} \tag{2.13}$$

Estocástica:

$$x^b = \begin{cases} 1 & \text{con probabilidad } p = \sigma(x), \\ -1 & \text{con probabilidad } 1 - p \end{cases} \tag{2.14}$$

Donde x representa el valor en $\Re$ de la variable en cuestión, x^b el valor binarizado y σ está definida como sigue:

$$\sigma(x) = max(0, min(1, \frac{x+1}{2})) \tag{2.15}$$

La introducción de un componente estocástico en esta función permite compensar la pérdida de información generada por la discretización de los valores.

Función de costo

La implementación propuesta de esta arquitectura considera la función de costo **Hinge**

$$max(0, 1 - t * y) \tag{2.16}$$

Donde, t representa el valor objetivo e y representa la salida de la red neuronal.

Proceso de entrenamiento

Las redes neuronales binarizadas poseen pesos restringidos a $\{-1, 1\}$, sin embargo, los gradientes de los mismos se mantienen en variables que toman valores en $\Re$ dado que dicho nivel de precisión es necesario para que el método de descenso de gradiente funcione en lo absoluto.

Algoritmo

La secuencia de pasos para entrenar una red binarizada sucede de la siguiente forma:

Considerando:

- C: La función de costo de *minbatch*
- λ: El factor de disminución de aprendizaje.
- L: La cantidad de capas.
- $\circ$: La multiplicación elemento a elemento.
- $binarizar()$: La función de binarización(deterministica o estocástica).
- $acotar()$: La función de acotación.
- $normLotes()$: La función de normalización por lotes sobre las salidas de las funciones de activación.
- $atrasNormLotes()$: La función de propagación hacia atras por medio de la normalización.
- $actualizar()$: La política de actualización de pesos.
- X^{t+1}: Representando la actualización del parámetro X en el tiempo $t+1$.

Requiriendo:

- (a_0, a^*): Un lote de valores de entradas y valores de salida.
- W: El vector de pesos previo.
- θ: Los parámetros previos de $normLotes()$.
- γ: Coeficiente de inicialización de pesos.
- η: Previo parámetro de velocidad de aprendizaje.

Una vez planteado el uso de redes neuronales binarizadas, que soluciona la eficiencia en cuanto al uso recursos computacionales, resta definir un mecanismo que permita diseñar la topología de la red neuronal binarizada, para ello veremos un mecanismo de generación de la topología de forma incremental y automática.

Algoritmo 1 Entrenamiento de una red neuronal binarizada.

{1.Cálculo del gradiente de los parámetros:}
{1.1.Propagación hacia adelante:}

for k=1 to L **do**
  $W_k^b \leftarrow \text{binarizar}(W_k)$
  $s^b \leftarrow a_{k-1}^b W_k^b$
  $a_k \leftarrow \text{normLotes}(s_k, \theta_k)$
  if k<L **then**
    $a_k^b \leftarrow \text{binarizar}(a_k)$

{1.2.Propagación hacia atrás:}
{Puede observarse que los gradientes calculados usan valores en $\Re$}
Calcular $g_{a_L} = \frac{\partial C}{\partial a_L}$ conociendo a_L y a^*

for k=L to 1 **do**
  if k<L **then**
    $g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}$
  $(g_{s_k}, g\theta_k) \leftarrow \text{atrasNormLotes}(g_{a_k}, s_k, \theta_k)$
  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b$
  $gW_k^b \leftarrow g_{s_k}^\top a_{k-1}^b$

{2.Acumulación de los gradientes:}

for k=1 to L **do**
  $\theta_k^{t+1} \leftarrow \text{actualizar}(\theta_k, \eta, g\theta_k)$
  $W_k^{t+1} \leftarrow \text{acortar}(\text{actualizar}(W_k, \gamma_k \eta, gW_k^b), -1, 1)$
  $\eta^{t+1} \leftarrow \lambda \eta$

2.1.1.4 Aprendizaje estructural adaptativo de redes neuronales

Este algoritmo, propuesto por *Cortes et al*[1] es capaz de aprender tanto la estructura, como los pesos de una red neuronal a partir de los datos de entrenamiento. Adicionalmente, dicho estudio establece que los algoritmos implementados resultan en la solución de funciones convexas, por lo que garantizan la existencia de una única solución global. Otros algoritmos de entrenamiento de redes neuronales, se basan en la especificación de una topología que acota la complejidad de la hipótesis propuesta para solucionar el problema de aprendizaje, sin embargo, **AdaNet** aprende el nivel de complejidad del problema basándose en los datos, inicialmente, este algoritmo provee un modelo lineal simple, al cual se le van incorporando neuronas y capas, según sea necesario.

Componentes

Arquitectura de la red neuronal

Como se muestra en la Figura 2.4, la arquitectura propuesta considera conexiones entre neuronas de cualquier par de capas, no únicamente entre las capas adyacentes, como típicamente se propone en los modelos de redes neuronales hacia adelante.

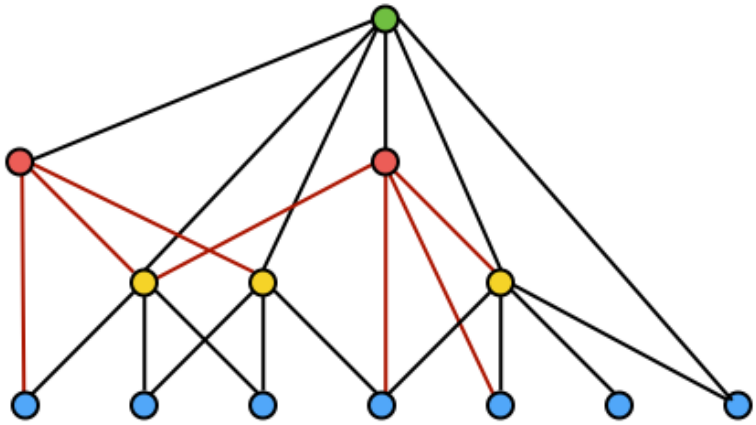


Fig. 2.4 Arquitectura general propuesta en AdaNet[1]

Los elementos contemplados en dicha arquitectura se definen como sigue. Sea L el número de capas intermedias de la red neuronal y n_k el número máximo de neuronas en la capa $k \in [L]$. Cada unidad $j \in [n_k]$ en la capa k representa una función denotada por $h_{k,j}$ (antes de su composición con la función de activación). Sea X el espacio de valores de entrada y $\forall x \in X$ y $\Psi(x) \in \Re^{n_o}$ representando el vector característico. Luego, la familia de funciones contempladas en la primera capa $h_{1,j}, j \in [n_1]$ es la siguiente:

$$H_1 = \{x \mapsto \mathbf{u} \cdot \Psi(x) : \mathbf{u} \in \Re^{n_0}, ||\mathbf{u}||_p \leq \Lambda_{1,0}\} \tag{2.17}$$

Donde, $p \geq 1$ define la norma-l_p y $\Lambda_{1,0} \geq 0$ es un parámetro de los pesos entre las capas 0 y 1. Asi mismo, la expresión general para la familia de funciones en $h_{k,j}, j \in [n_k]$, para $k > 1$ se define como sigue:

$$H_k = \{x \mapsto \sum_{s=1}^{k-1} \mathbf{u}_s \cdot (\varphi_s \circ \mathbf{h}_s)(x) : \mathbf{u}_s \in \Re^{n_s}, ||\mathbf{u}_s||_p \leq \Lambda_{k,s}, h_{k,s} \in H_s\} \tag{2.18}$$

Donde, para cada función de la neurona $h_{k,s}$ en (2.18), $\mathbf{u}_s$ representa el vector de pesos de dicha capa con la capa anterior $s < k$. $\Lambda_{k,s}$ representa los parámetros (no negativos) y $\varphi_s \circ \mathbf{h}_s$

denota la composición componente a componente, adicionalmente, φ_s representa la función de activación utilizada. Finalmente, el parámetro p denotará el nivel de esparción de la red neuronal y la complejidad del conjunto de hipotésis H_k

Proceso de entrenamiento

Sea $x \mapsto \Phi(-x)$ una función convexa no-creciente, como la función exponencial $\Phi(x) = e^x$ o la función logística $\Phi(x) = log(1+e^x)$. Luego, *AdaNet* hace uso de la siguiente función objetivo:

$$F(w) = \frac{1}{m}\sum_{i=1}^{m}\Phi(1 - y_i\sum_{j=1}^{N} w_j h_j) + \sum_{j=1}^{N}\Gamma_j|w_j| \tag{2.19}$$

Donde, $w \in \Re^N$ y $\Gamma_j = \lambda r_j + \beta$, con $\lambda \geq 0$ y $\beta \geq 0$ como parámetros.

Sea también $B \geq 1$ un parámetro fijo que determine la cantidad de neuronas por capa en una súbred neuronal candidata a extender el actual modelo, el algoritmo sucede en T iteraciones. Sea L_{t-1} la profundidad de la red neuronal construida antes de la iteración t, luego, en la iteración t el algoritmo puede proseguir a partir de las siguiente opciones:

1. Extender la red neuronal existente con una súbred de la misma profundidad $h \in H^{*B}_{L_{t-1}}$, con B neuronas por capa. Cada neurona en la capa k de esta súbred puede tener conexiones con neuronas existentes en las capa $k-1$, asi como, conexiones con las neuronas de la capa k de la propia súbred.

2. Extender la red neuronal existente con una súbred de mayor profundidad$(L_{t-1}+1)$,$h' \in H^{*B}_{L_{t-1}}$, las conexiones de esta súbred poseen las mismas restricciones que h.

La opción escogida, es aquella que lleve a la mayor reducción del existente valor de la función objetivo, que a su vez depende de la complejidad de la súbred incorporada y el error empírico resultante. El algoritmo termina despues de T iteraciones, o si la extensión de la red neuronal existente no mejora la función objetivo(2.19).

Nótese que tanto h como h' son generados por algún *aprendizDebil* que puede ser, por ejemplo, una red neuronal de un conjunto de redes neuronales aleatorias, que optimicen 2.20, o bien, su versión regularizada directamente (2.21)

$$F_t(\mathbf{w},\mathbf{u}) = \frac{1}{m}\sum_{i=1}^{m}\Phi(1 - y_i f_{t-1}(x_i) - y_i\mathbf{w}\cdot\mathbf{u}(x_i)) + \Gamma_{\mathbf{u}}||\mathbf{w}||_1, \tag{2.20}$$

Donde, $\Gamma_{\mathbf{u}} = \lambda r_u + \beta$ y $r_u = \mathfrak{R}_m(H_{L_{t-1}})$ si $\mathbf{u} = h$ y $r_u = \mathfrak{R}_m(H_{L_{t-1}+1})$ de lo contrario. (version regularizada)

$$\hat{F}_t(\mathbf{w}, \mathbf{h}) = \frac{1}{m} \sum_{i=1}^{m} \Phi(1 - y_i f_{t-1}(x_i) - y_i \mathbf{w} \cdot \mathbf{h}(x_i)) + \mathsf{R}(\mathbf{w}, \mathbf{h}), \tag{2.21}$$

En la Figura mostrada en (2.5) los bloques en azul representan las capas de entrada, los

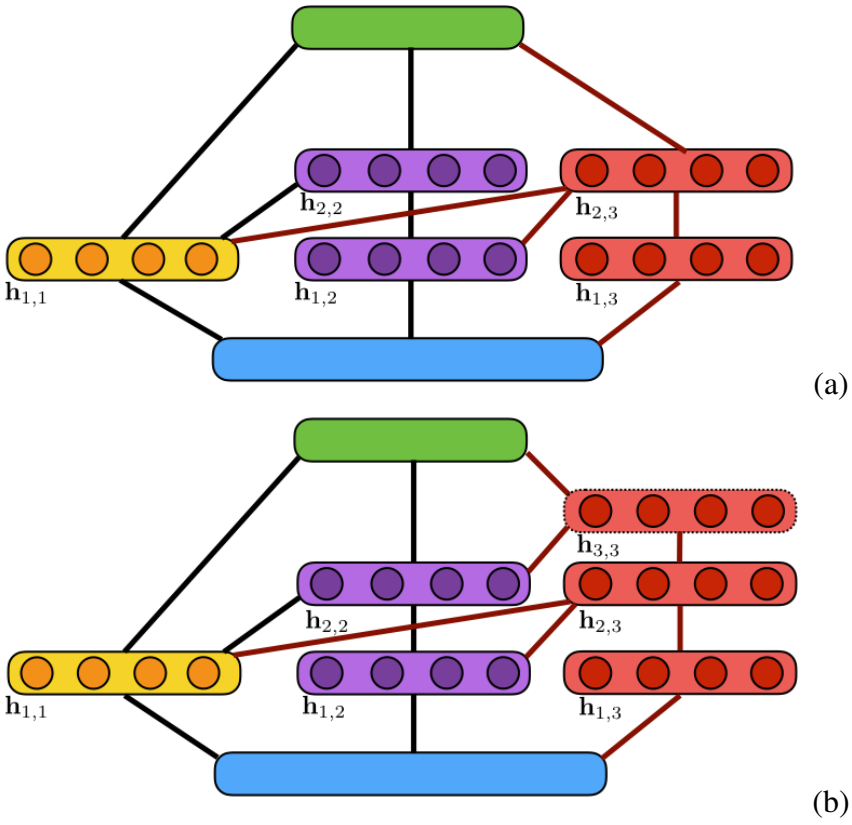


Fig. 2.5 Ejemplo de construcción incremental de una red neuronal con AdaNet[1]

bloques en verde representan la capa de salida, mientras que los bloques en amarillo representan las unidades agregadas en la primera iteración y las unidades en los bloques morados son agregados en la segunda iteración, finalmente, los bloques rojos contienen dos posibles extensiones a la red neuronal: (a) una extensión de dos capas; (b) una extensión de tres capas.

Evaluación del desempeño de redes neuronales

Con el objetivo de evaluar el desempeño de la red neuronal implementada y facilitar su comparación con otros algoritmos existentes, basados en redes neuronales, y enfocándonos

Algoritmo 2 Pseudocódigo para AdaNet($S = ((x_i, y_i)_{i=1}^m)$)

$f_0 \leftarrow 0$
for t=1 to T **do**
    $\mathbf{h}, \mathbf{h}' \leftarrow$ aprendizDebil(S, f_{t-1})
    $w \leftarrow$ minimizar$(F_t(\mathbf{w}, \mathbf{h}))$
    $w' \leftarrow$ minimizar$(F_t(\mathbf{w}, \mathbf{h}'))$
    if $F_t(\mathbf{w}, \mathbf{h}) \leq F_t(\mathbf{w}', \mathbf{h}')$ **then**
        $\mathbf{h}_t \leftarrow \mathbf{h}$
    else $\mathbf{h}_t \leftarrow \mathbf{h}'$
    if $F_t(\mathbf{w}_{t-1} + \mathbf{w}^*) < F(\mathbf{w}_{t-1})$ **then**
        $f_{t-1} \leftarrow f_t + \mathbf{w}^* \cdot \mathbf{h}_t$
    else **return** f_{t-1}
return f_T

particularmente en su capacidad de generalización (2.1), consideraremos la métrica de evaluación denominada **exactitud** y definida como la proporción de registros para los cuales el algoritmo genera la salida correcta. El mencionado indicador, será calculado mediante la metodología de validación cruzada.

Validación cruzada de K conjuntos

Este método consiste en dividir el conjunto de aprendizaje en K subconjuntos de, aproximadamente, el mismo tamaño. El modelo es entrenado con los datos pertenecientes a K-1 subconjuntos y se evalúa el desempeño con el subconjunto que fue excluido inicialmente. En una siguiente iteración, el subconjunto excluido pasa a formar parte del conjunto de entrenamiento y se excluye un nuevo subconjunto,de los K subconjuntos, cada vez. Este proceso es realizado K veces. Al finalizar, se retorna un promedio del modelo aplicado de acuerdo a la métrica seleccionada.

2.2 Procesamiento digital de imágenes

El algoritmo propuesto, **AdaBnn**, será probado en el problema de clasificación de imágenes según los objetos detectados en ellas, el cúal está contenido en el área de investigación de procesamiento digital de imágenes, que definiremos cómo el conjunto de procesos dedicados al tratamiento de imáges capturadas mediante el uso de dispositivos digitales [26]. Particularmente, dentro del procesamiento digital de imágenes, se contemplan diversos mecanismos que permiten expandir la información que puede contener una imagen representada en su forma más simple, que típicamente consta de un arreglo bi-dímensional, cuyas

entradas poseen una función de composición de colores que varía de acuerdo a los términos en que se realice dicha composición, por ejemplo, el modelo de color *RGB* requiere un arreglo de tres posiciones, para referir la intesidad cada canal que contiene: Rojo, verde, azúl.

2.2.1 Descriptores de características

En función, del problema planteado, puede ser apropiado, recurrir a diferentes representaciones de los datos, en este caso imágenes, convengamos en definir **Descriptor de características** como una representación de la imagen de entrada que resalta ciertos rasgos que mejoran su aporte en la consecución de los objetivos definidos, por ejemplo, un descriptor puede consistir en representar una imagen de dimensiones *alto* x *ancho* x *canales* en un vector de *N* posiciones que muestren el brillo de cada píxel. Veamos algunos de los descriptores referidos en la presente investigación.

2.2.1.1 Histograma de color

Es una representación de la distribución de los valores que toma la función de composición de colores en las entradas del arreglo bídimensional que caracteriza la imagen.

2.2.1.2 Histograma de gradiente de orientaciones

Es una representación de la distribución de las orientaciones de los gradientes con respecto a *x* e *y* de la imagen. Este descriptor ha sido extensamente usado para tareas relacionadas con la detección de objetos, ya que provee información de bordes y esquinas de las imágenes, caracterizando así, la forma en general.[4]

2.2.2 Visión por computador

En la aplicación de algoritmos basados en redes neuronales profundas, una de las áreas más activas es la visión por computador, siendo pertenecientes a ésta, las tareas de aprendizaje que se toman cómo criterio para probar nuevos algoritmos. Específicamente, utilizaremos cómo referentes para la aplicación del modelo propuesto **AdaBnn** las áreas de reconocimiento de objetos y reconocimiento de caracteres; siendo este último un caso particular del anterior en el cúal se conoce de antemano la naturaleza del objeto a detectar, a saber, caracteres.

2.2.2.1 Estado del arte

En esta sección veremos algunas de las soluciones más efectivas actualmente usadas para la solución de las tareas de reconocimiento de objetos y reconocimiento de caracteres

planteadas, para ello, restrigiremos el espacio de busqueda de dichos problemas a los conjuntos de datos seleccionados para la prueba del modelo propuesto en el presente estudio **MNIST**, **CIFAR-10**.

CIFAR-10[17]

Este conjunto de datos consiste en 60.000 imágenes a color de 32x32 píxeles etiquetadas con 10 diferentes clases, con 6.000 imágenes por clase, resultando así en 50.000 imágenes para el conjunto de entrenamiento y 10.000 imágenes para el conjunto de prueba, inicialmente este conjunto de datos permite tareas de clasificación tanto para considerar el escenario de clasificación como binario puede plantearse el problema como: *venado-camion*, *venado-caballo*, *carro-camion*, *perro-gato*, *perro-caballo*. Aunque una reciente investigación postula alcanzar **97.69%**[35], la misma se encuentra aún bajo revisión de otros científicos para asegurar su validez académica, por tanto, la mejor(mayor) precisión alcanzada a conocimiento del investigador ha sido obtenida con una variación de la función de agrupación (2.1.1.2) llamada *Max-pooling* llamada *fractional Max-pooling*[7], que consiste, en líneas generales, de permitir que α tome valores no enteros, dicha versión considera también la aleatoriedad de tomar diferentes regiones de agrupación, la topología con la cúal abarcaron el conjunto de datos **CIFAR-10** consistió en **10** épocas de entrenamiento con una arquitectura de 300 neuronas con C2 cómo convoluciones, luego, 300 neuronas más con C2 cómo convoluciones, Max-pooling2 repetido 5 veces, convolución C2, convolución C1 y la capa de salida. Finalmente, este mecanismo consiguió una precisión de **96.53%**.

MNIST[19]

Ésta, es una base de datos de dígitos escritos a mano, este conjunto de datos posee un total de 60.000 observaciones etiquetadas y 10.000 observaciones no etiquetadas, cada imagen consiste en imágenes de 28×28 representando dígitos del 0 al 9., fué seleccionado dado el mismo funciona como referencia para la comparación del desempeño de modelos aplicados a la clasificación de imágenes. La mejor(menor) tasa de error fue alcanzada al hacer uso de un mecanismo de regularización llamado *DropConnect*[34] que es una generalización de la técnica Dropout [25] y consiste en igualar a cero un subconjunto, seleccionado aleatoriamente, de pesos de las conecciones entre las neuronas de una red neuronal. en dicho estudio, se consiguió una tasa de error de **0.21%**, veamos algunos detalles de la implementación de *DropConnect* que alcanzó dichos resultados.

Para los experimentos con **MNIST**, *Li Wan, et al* utilizaron una red neuronal densa (totalmente conectada) de **2** capas, la primera capa tomaba cómo entrada los píxeles de cada imagen y la salida de la segunda capa alimentaba una capa de activación *softmax* de **10** clases.

Cómo funciones de activación de las capas ocultas se usaron: tangente hiperbólica, sigmoide y unidades lineales rectificadas, para el descenso estocastico de gradiente usaron una tasa de aprendizaje de **0.1** y probaron con entrenamientos de 600-400-20 épocas respectivamente, en este conjunto de datos no utilizaron ningún tipo de *data augmentation*2.1.1.1.

2.2.2.2 Herramientas de software

Junto al auge del aprendizaje profundo, se han popularizado diversas herramientas de software, principalmente, involucrando los lenguajes de programación *Python, Lua, C++*, de éstos, *Python* se ha seleccionado dada la cantidad de marcos de desarrollo existente en dicho lenguaje.

Python

Principalmente *Tensorflow, Theano, Keras*, siendo este último el que utilizaremos para la implementación de la solución.

Capítulo 3

Marco de metodológico

En el presente capítulo se explica el proceso de desarollo de los modelos de aprendizaje, implementación de los algoritmos que hagan uso de dichos modelos y la evaluación de los algoritmos implementados, con el fin de proveer la información necesaria para reproducir los productos de la presente investigación y auditar el desarollo realizado.

En líneas generales, se siguieron las recomendaciones de *Goodfellow et al*[13] sobre el proceso de diseño práctico, cómo sigue:

- Determinar los objetivos, métricas de error y el valor deseado para dichos métricas, con respecto al problema que se trata de resolver.
- Establecer un flujo completo del proceso de aprendizaje junto a la estimación de las métricas de desempeño.
- Auditar el sistema para determinar posibles puntos de congestión que afecten el desempeño del modelo.
- Realizar cambios incrementales, repetitivamente, en aspectos cómo ajuste de hiperparámetros, de la implementación, entre otros.

3.1 Metodología de desarrollo

El presente estudio está enfocado en una etapa específica del flujo de trabajo de aprendizaje automático, la creación y evaluación del algoritmo de generación de modelos de aprendizaje(**AdaBnn**), por lo que se redujeron lo esfuerzos en los procesos relacionados con otras actividades de dicho flujo. Así mismo, los conjuntos de datos seleccionados para la evaluación del modelo, poseen extensa variedad de soluciones y flujos de trabajo mediante

los cuales fueron abarcados, así que se tomaron cómo guía las actividades realizadas en las dos influencias principales de **AdaBnn**,*Courbariaux et al*[11](**AdaNet**) y *Cortes et al* [1]**Redes neuronales binarizadas**. A continuación, se describen los procesos involucrados en el desarrollo de **AdaBnn**:

1. Se diseño un algoritmo en código python que considera los fundamentos teóricos relativos al modelo **AdaNet**.

2. Se diseño un algoritmo a partir de la implementación del modelo **AdaNet**, que tuviera los ajustes necesarios para integrar los principios relacionados a las **Redes neuronales binarizadas**.

3. Se iteró en la modificación de dichas implementaciónes en la medida en la que ambas lograban emular en cierta medida los estudios respectivos.

3.1.1 Consideraciones de implementación

Los principios de diseño que fundamentaron la implementación, fueron en principio la usabilidad de la solución propuesta, para ello, se definió cómo criterio la integración con *frameworks* existentes, entre ellos, Keras, dado que éste admite la integración de modelos predefinidos de redes neuronales profundas bajo la figura de aplicaciones, de hecho ya provee un plataforma para incorporar modelos cómo el descrito en el presente estudio, facilitando así la evolución y adopción del presente desarrollo al ser parte de dicho proyecto. Para propiciar esta integración, la implementación se realizó tomando cómo lineamientos las aplicaciones actualmente dispinibles en Keras, cómo el modelo *Inception*[28], entre otros. En este sentido, los algoritmos desarrollados se han empaquetado en forma de funciones, tanto para su uso en el plataforma de experimentación usada, cómo para su adición a las aplicaciones del paquete Keras.

3.1.1.1 Funciones de activación

Cómo funciones de activación se consideraron inicialmente aquellas mencionads en los estudios respectivos a los antecesores del modelo propuesto, especificamente la función **ReLu**, adicionalmente, la función de activación de la capa de salida depende de la cardinalidad de las clases a predecir por parte del modelo de aprendizaje, en el caso binario se trabajó con la función **sinusoidal**, en el caso de la clasificación multiclase se trabajó con la función **softmax**.

3.1.1.2 Funciones objetivo

A pesar de que el modelo **AdaNet** especifíca una función objetivo (o de costo) que habilita bondades teorícas, que también sustentan la metodología de extensión incremental que caracteriza el estudio, se demostró mediante la experimentación que usando otras funciones de costo prevalece la mejora de desempeño que representa el aprendizaje estructural adaptativo. Específicamente, las funciones usadas fueron la **entropía cruzada binaria** y la **entropía cruzada categórica DEFINIR** dependiendo de la cardinalidad de las clases a predecir por parte del modelo de aprendizaje, binario o multiclase, respectivamente.

3.1.1.3 Técnicas de optimización

Para minimizar las mencionadas funciones objetivo, se decidió trabajar con las función F propuesta en **AdaNet**, y el método **AdamDESARROLLAR**. Adicionalmente, se agregó la propiedad de detener el algoritmo en alguna iteración en que una nueva época de entrenamiento de la red no refleje mejoras en la función objetivo.

3.2 Infraestructura del entorno de experimentación

Para los experimentos se uso la plataforma Google colaboratory que permite usar aceleración de cómputo mediante GPU, se incluyen algunas de las especificaciones de dicho entorno:

- **Hardware**:

  CPU: Intel(R) Xeon(R) CPU @ 2.20GHz.

  GPU: NVIDIA Tesla K80 - 24 GB GDDR5.

  RAM: 13 GB de memoria RAM.

- **Software**:

  Sistema operativo: Ubuntu 17.10.

  Lenguaje de programación: python, versión 3.6.3

  Principales paquetes: Keras (sobre Tensorflow) y algunas utilidades del paquete sci-kit learn.

3.3 Metodología de evaluación

En esta sección se describen las condiciones bajo las cuales se evaluaron los algoritmos desarrollados, esto es, la implementación de **AdaNet** y la implementación de **AdaBnn**.

Para la evaluación de los modelos implementados, se consideraron problemas relativos al área de visión por computador, dado que estos problemas suelen usarse de referencia para la evaluación y comparación de nuevas implementaciones de redes neuronales profundas, como las presentes. Así mismo, se consideró cómo métrica de desempeño de referencia la precisión general, esto es, la proporción de observaciones correctamente clasificadas por el modelo, adicionalmente se consideran la cantidad de neuronas por capa y el tiempo de ejecucion. Para el entrenamiento y validación del modelo se reservo un conjunto de datos y otra conjunto se uso para la medición del desempeño del algoritmo desarrollado, mediante la técnica de validación cruzada en k conjuntos, en este sentido, se promediaron los valores de precisión resultantes y se consideró la desviación estándar en cada caso. Adicionalmente, se comprobó la significancia estadística del estimador de la precisión en la muestra considerada para el conjunto de prueba, con el objetivo de definir un intervalo de confianza del **95%**.

Capítulo 4

Algoritmo propuesto: AdaBnn

En este capítulo, se detalla el algoritmo **AdaBnn**, que es el modelo propuesto en el cúal se integran los conceptos presentados en **AdaNet** y **Redes neuronales binarizadas** con la que se plantea solucionar las limitantes de los modelos al ser usados por separado, incluyendo las funciones de activación de las que hace uso, su procedimiento de entrenamiento, entre otros aspectos.

El algoritmo de creación de modelos basados en redes neuronales **AdaBnn** está inspirado en dos propuestas independientes que ofrecen diferentes mejoras técnicas y que, a conocimiento del investigador, no habian sido integradas antes de la elaboración de esta investigación, la síntesis de estos componentes caracteriza entonces el modelo resultante **AdaBnn**.

En primer lugar, el modelo propuesto hace uso de *aprendizaje estructural adaptativo* para su conformación, esto es, parte de una topología de red neuronal sencilla, típicamente de una capa con M neuronas, y crece adaptativamente según dos hasta que:

- Se alcance un número fijado de iteraciones para el algoritmo.

- Se alcance un desempeño no mejorable con respecto a la función objetivo.

Adicionalmente, el modelo incorpora restricciónes sobre los parámetros de la red que han probado tener ventajas técnicas[11], cómo mejoras en los tiempos de entrenamientos, específicamente, la binarización de los pesos de las conexiones de las neuronas en tiempo de entrenamiento, a pesar de que la implementación provista por los autores no parece hacer uso de una representación que aproveche en su totalidad las bondades de usar un conjunto de pesos aislado al espacio $\{-1, 1\}$.

4.1 Detalles de la arquitectura

La arquitectura del modelo **AdaBnn**, por definición, es adaptable a los datos inputados. Sin embargo, existen algunos principios que todas las topologías resultantes tendrán en común, dichos similitudes se presentan a continuación.

Funciones de optimización

Las funcines de optimización que contemplan los estudios sobre los cuales se soportan las propuestas **AdaNet** y **Redes neuronales binarizadas**, punto de partida de la presente arquitectura, son:

- **Adam**[16]
- **Descenso estócastico de gradiente** (sección 2.1.1.1)

Funciones de activación

La función de activación por defecto del modelo **AdaBnn** Es una binarización de la función ReLu, en la cúal se binarizan los valores luego de su evaluación en ReLu.

Funciones de costo

Cómo función de costo se utilizó la entropía binaria cruzada, para involucrar el componente de la complejidad Rademacher dicha función se agrego cómo una función de regularización dado que se determino ésta, cómo la forma mas eficiente de ubicar dicha métrica dentro de los conceptos manejados por **Keras** y al mismo tiempo para representar de la mejor forma la metodología propuesta en **Adanet**[1]

4.2 Implementación propuesta del algoritmo AdaNet

Este modelo fue implementado, al replicar la descripción realizada en **Adanet**[1], sin embargo, otro de los principios considerados es el uso de la biblioteca **Keras** para facilitar su adopción en problemas fuera del presente estudio y su mejora por parte de la comunidad al liberar el código relacionado en repositorios públicos adabnn_keras, una de las consideraciones necesarias es la correspondencia de los bloques descritos en AdaNet, que en la implementación realizada se asemeja más a la abstracción de capas, las cuales se definen por partes dado que se admiten nuevas neuronas agregadas en iteraciones diferentes. Una

de las características que se decidió no agregar aunque fue mencionada en el artículo en cuestión, es la finalización del algoritmo al no percibir variaciones de al menos λ, ya que requeriría un parámetro adicional para AdaNet e igualmente la cota superior en cuanto a la cantidad de iteraciones está definida por el parámetro **T**. Es importante resaltar que no existe, a conocimiento del investigador una implementación previa de **AdaNet**[1] que haya logrado replicar los resultados provistos por dicho artículo, aún considerando la referencia de https://github.com/davidabek1/adanet, que provee un desempeño mucho menor para el conjunto de datos CIFAR-10.

Binarización de la red

La binarización de los pesos en la red, puede lograrse mediante dos estrategias [2], citadas en la sección (iii), para la presente implementación de AdaBnn, también se agregaron *kernels* de inicialización de pesos y así no tener la necesidad de binarizarlos al principio, para dicho proceso se usaron los parametros de *Glorot, Bengio*[6].

Capítulo 5

Experimentos

El presente capítulo detalla las prácticas de experimentación realizadas para poner a prueba los algoritmos implementados (AdaNet y AdaBnn). Específicamente, sobre los conjuntos de datos **CIFAR-10** y **MNIST**.

5.1 Clasificación binaria

Uno de las aplicaciones estándar de los algoritmos de redes neuronales profundas es la clasificación binaria, a su vez, uno de los casos de uso más comunes es la clasificación de imágenes, veamos entonces el comportamieno y desempeño de los algoritmos implementados en el caso de uso de clasificación de imagenes con el conjunto **CIFAR-10**.

5.1.1 CIFAR-10

Para las pruebas asociadas a la calidad de los modelos creados bajo aprendizaje estructural adaptativo se escogió el conjunto de datos **CIFAR-10** [17]. Este conjunto de datos fue recopilado por académicos tanto del MIT cómo NYU durante seis meses, proceso descrito en [30], en el cúal, se buscaron imágenes en páginas cómo Google, Flickr, y Altavista, usando cómo términos de búsqueda sustantivos concretos contenidos en la base de datos [21]. De allí fueron almacenados apróximadamente 3000 resultados, por cada término de búsqueda. Luego, se removieron duplicados e imágenes con grandes proporciones de píxeles blancos en la imágen, cómo un esfuerzo por mantener sólo imágenes naturales. El término de búsqueda es fijado cómo la etiqueta, así conformó un total de 80.000.000 imágenes a color redimensionadas a 32 x 32, a lo largo de 79.000 términos de búsqueda. Finalmente, el conjunto de datos usado en esta investigación es un subconjunto del proceso mencionado anteriormente, consistiendo en un total de 60.000 imágenes pertenecientes a

10 clases(6.000 imágenes por clase), siendo además divididas en dos subconjuntos: Un subconjunto de entrenamiento compuesto de 50.000 imágenes y un subconjunto de prueba de 10.000 imágenes, ambos divididos aletoriamente, adicionalmente, las clases son mútuamente excluyentes.

Para evaluar los algoritmos desarrollados en la clasificación binaria, se filtró el conjunto de datos para incluir sólo las imagenes pertenecientes a un par de etiquetas por vez, como sigue[1]:

- deer-truck.
- deer-horse.
- automobile-truck.
- cat-dog.
- dog-horse

Después de dicho filtro, Las imágenes fueron preprocesadas para generar las características propuestas en **AdaNet**[1], esto es, se generaron los histogramas de color con 8 segmentos de agrupación de los valores, para cada uno de los 3 canales definidos con el espacio de colores RGB, adicionalmente, se generaron los histogramas de orientación de los gradientes tomando en cuenta 9 direcciones, 8x8 píxeles por celda y 3x3 celdas por bloque, resultando en un vector característico de **836** valores númericos.

Se utilizó la técnica de data escondida para la evaluación del modelo usando **70%** para el conjunto de aprendizaje y un **30%** para el conjunto de prueba ya que la configuración de evaluación por validación cruzada implica tiempos de entrenamiento muy altos al recrear el modelo en cada iteración de la técnica de validación cruzada[2]. Sin embargo, se estimó el intervalo de confianza para cada evaluación del modelo, de modo de proveer un estimado del comportamiento en datos previamente no vistos por el modelo, manteniendo así el estimado que provee la desviación estándar de la precisión en la validación cruzada[29].

De cada problema de clasificación binaria planteado, se incluyen en esta sección las tablas con todos los parámetros fijados y resultados obtenidos pero sólo las gráficas de precisión y función de costo del algoritmo con mejores resultados entre todas las iteraciones, esto es, la mejor configuración probada para el problema en cuestión. El resto de las gráficas se han incluído en la sección de **Anexos B**.

[1] Dicha selección es la misma propuesta por [1]

[2] github/keras/issues, googlegroups/keras/users

AdaNet

Los experimentos con la implementación propuesta de AdaNet sobre el conjunto de datos CIFAR-10 en el planteamiento de clasificación binaria se realizaron como sigue:

- Se realizaron 3 iteraciones sobre el primer par de etiquetas (deer-truck) con diferentes parámetros, para explorar el comportamiento de AdaNet sobre dicho problema y definir una configuración inicial que profundizara en el comportamiento de AdaNet con diferentes configuraciones.

- Se realizaron 2 iteraciones con el resto de los pares de etiquetas del problema con el objetivo de contrastar diferentes niveles de complejidad de los modelos resultantes con AdaNet y estudiar su desempeño para cada problema.

Las primeras pruebas exploratorias de este algoritmo, permitieron definir un conjunto de parámetros que proveen el mejor rendimiento, a partir de éstos, se realizaron variaciones para comprender mejor el comportamiento del modelo, sin embargo, sólo se incluyen los parámetros que influyeron en el mejor desempeño del algoritmo en las mencionadas pruebas exploratorias. Entre dichos parámetros se encuentra:

- **Función de costo:** La función de costo que permitió generar los mejores resultados en las pruebas con **AdaNet** fue la entropia cruzada binaria(*binary_crossentropy*), en contraste con *RMSprop*.

- **Función de optimización:** La función de optimización que permitió generar los mejores resultados en las pruebas con **AdaNet** fue Adam[16].

- **Tamaño de lote:** El tamaño de lote que permitió generar los mejores resultados fue 100, este tamaño también viene dado por las pruebas realizadas en **Redes neuronales binarizadas**[11].

- **B**: Este parámetro, que define la cantidad de neuronas que se agregaran por capas en cada iteración, fue probado en 2 configuraciones principalmente, 100 y 150.

Los demás parámetros se han configurado de diferentes formas y se han incluído sus variaciones en la forma de iteraciones, dado que los mismos permiten realizar ciertas conjeturas sobre el comportamiento de la implementación de **AdaNet** realizada.

Los resultados obtenidos muestran que, con poca o ninguna referencia de cómo definir la topología de la red neuronal necesaria para el problema de clasificación en cuestión, el algoritmo **AdaNet** puede proveer resultados favorables, aunque mejorables con el ajuste de

parámetros necesario. Es observable, desde los resultados obtenido para el problema **deer-truck** (tabla 5.2) y los parámetros fijados para el mismo (tabla 5.1), que la complejidad de la red resultante no implica un mejor desempeño sobre el problema en cuestión. Adicionalmente, durante la generación de la red neuronal finalmente usada para el problema de clasificación, se muestra un consistente crecimiento en la precisión general sobre los datos de aprendizaje, así cómo un decreciente valor resultante de la función de costo (véase la Fig. 5.1).

En el problema **deer-horse** resalta que, quizá por la complejidad superior de este problema, la red resultante que tiene mejores resultados (tabla 5.3 y tabla 5.4) es de mayor profundidad que para el problema **deer-truck** aún con un número menor de épocas de entrenamiento.

Otra observación meritória, apreciable en los resultados del problema **automobile-truck**, es que **AdaNet** puede proveer redes neuronales de desempeño equivalente (tabla 5.5 y tabla 5.6) pero con diferentes complejidades en este caso en términos de la cantidad de parámetros, esto puede instar a que el ajuste de parámetros inicial para un problema cualquiera comience con variaciones que consideren redes neuronales poco complejas, esto es, valores pequenños para el parametro **B** y **T**.

En general, los resultados obtenidos de las experiencias realizadas con **AdaNet** dan indicios de ciertas propiedades cómo la consistencia en el aprendizaje (a lo largo de las iteraciones de **AdaNet** por el parámetro **T**), así mismo, existe la posibilidad de que el modelo resultante sea en muchos casos propenso al sobreajuste con respecto a los datos de entrenamiento. Futuras investigaciones pudieran agotar esta conjetura al exponer **AdaNet** a muchos otros escenarios de aprendizaje, tanto cómo a una mayor variación de parámetros probados.

A pesar de que la propuesta inicial de **AdaNet** considera una función de costo diferente y que las precisiones logradas en el artículo referido difieren un poco de las presentes, el comportamiento resultante, parece confimar los argumentos de creación de la metodología de aprendizaje estructural adaptativo. Sin embargo, el caso de uso planteado (**CIFAR-10**) ha demostrado ser efectivamente abarcado por redes neuronales convolucionales, que resultan incompatibles con ciertos aspectos de la actual implementación de **AdaNet**, específicamente la incompatibilidad de las dimensiones entre las capas de convoluciones ya que el modelo general de **AdaNet** admite la conexión entre neuronas no consecutivas y las convoluciones implican una variación de la dimención de los datos de entrada en cada una de sus instancias.

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	100	15	binary_crossentropy	Adam	100	2
2	100	5	binary_crossentropy	Adam	100	20
3	150	2	binary_crossentropy	Adam	100	25

Table 5.1 Parámetros por iteración de pruebas
de AdaNet sobre CIFAR(**deer-truck**)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	727	7	6879200	0.891 ± 0.006
2	476	2	942200	0.890 ± 0.007
3	178	2	556650	0.893 ± 0.006

Table 5.2 Rendimiento por iteración de pruebas
de AdaNet sobre CIFAR(**deer-truck**)

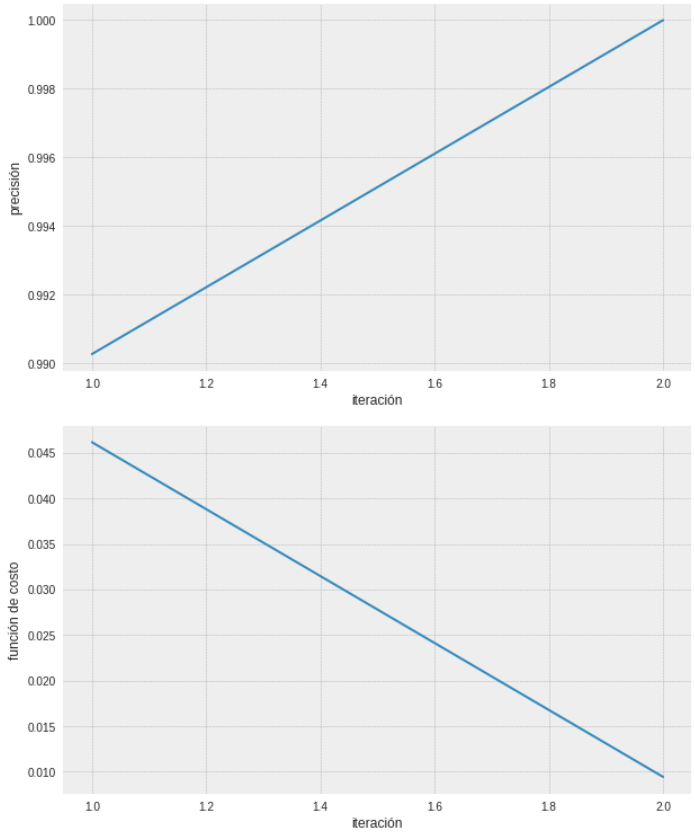


Fig. 5.1 Desempeño de AdaNet
sobre CIFAR-10 (**deer-truck**)

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	2	binary_crossentropy	Adam	100	25
2	150	5	binary_crossentropy	Adam	100	20

Table 5.3 Parámetros por iteración de pruebas
de AdaNet sobre CIFAR(**deer-horse**)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	198	2	556650	0.747 ± 0.009
2	638	3	1765800	0.750 ± 0.009

Table 5.4 Rendimiento por iteración de pruebas
de AdaNet sobre CIFAR(**deer-horse**)

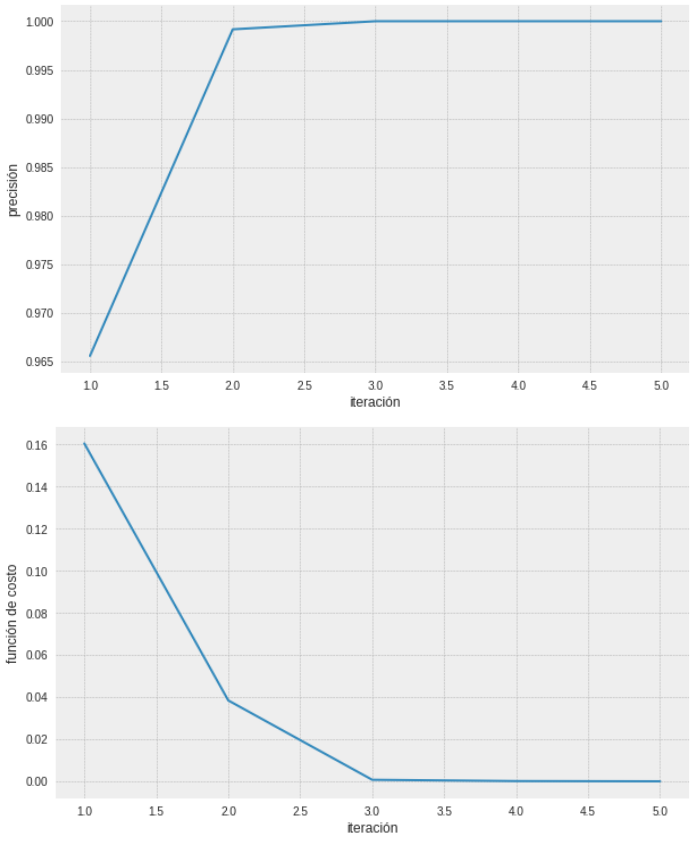


Fig. 5.2 Desempeño de AdaNet
sobre CIFAR-10 (**deer-horse**)

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	2	binary_crossentropy	Adam	100	25
2	150	5	binary_crossentropy	Adam	100	20

Table 5.5 Parámetros por iteración de pruebas de AdaNet sobre CIFAR(**auto-truck**)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	39	1	556650	0.690 ± 0.010
2	73	2	1743300	0.690 ± 0.010

Table 5.6 Rendimiento por iteración de pruebas de AdaNet sobre CIFAR(**auto-truck**)

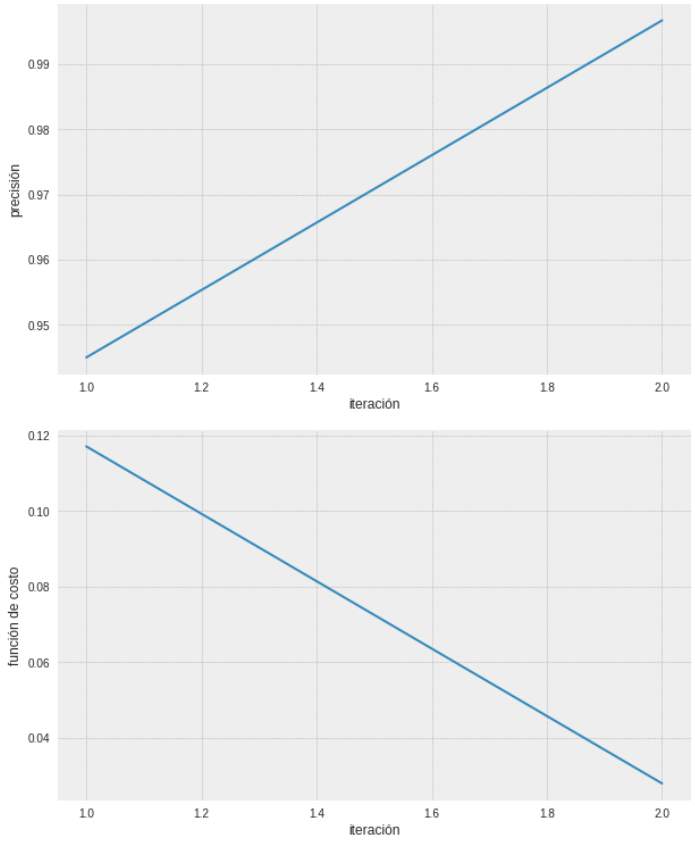


Fig. 5.3 Desempeño de AdaNet sobre CIFAR-10 (**automobile-truck**)

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	2	binary_crossentropy	Adam	100	25
2	150	5	binary_crossentropy	Adam	100	20

Table 5.7 Parámetros por iteración de pruebas
de AdaNet sobre CIFAR(**cat-dog**)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	40	1	556650	0.597 ± 0.010
2	119	2	1765800	0.593 ± 0.010

Table 5.8 Rendimiento por iteración de pruebas
de AdaNet sobre CIFAR(**cat-dog**)

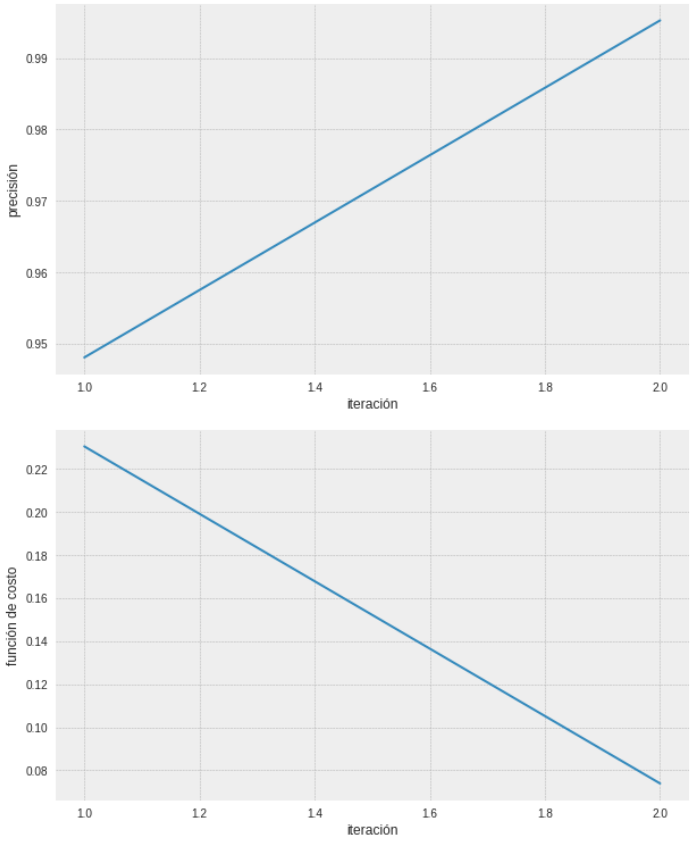


Fig. 5.4 Desempeño de AdaNet
sobre CIFAR-10 (**cat-dog**)

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	5	binary_crossentropy	Adam	100	20
2	150	2	binary_crossentropy	Adam	100	25

Table 5.9 Parámetros por iteración de pruebas
de AdaNet sobre CIFAR(**dog-horse**)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	132	2	1720800	0.710 ± 0.010
2	53	1	556650	0.714 ± 0.010

Table 5.10 Rendimiento por iteración de pruebas
de AdaNet sobre CIFAR(**dog-horse**)

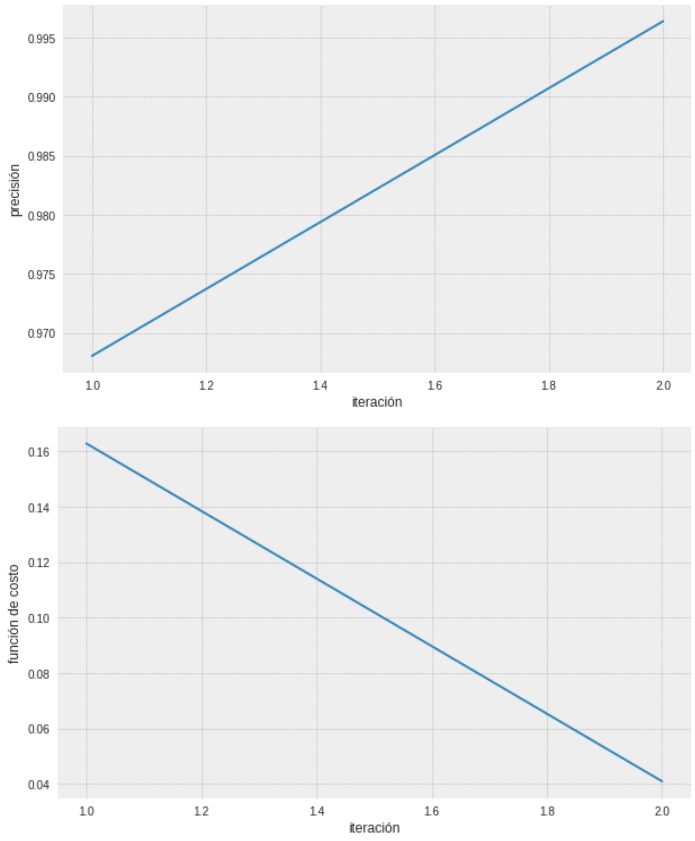


Fig. 5.5 Desempeño de AdaNet
sobre CIFAR-10 (**dog-horse**)

5.1.2 AdaBnn

El algoritmo **AdaBnn** fue evaluado sobre el conjunto de datos **CIFAR-10** con diferentes parámetros, para determinar su comportamiento en distintos escenarios. Sin embargo, con menos iteraciones que el algoritmo **AdaNet** ya que pruebas exploratorias iniciales demostraron la inconsistencia inicial del modelo para este conjunto de datos y la poca mejora de los resultados obtenidos.

Entre las diferencias de las presentes pruebas con las citadas **Redes Binarizadas**[11], resaltan los siguientes aspectos:

- Se presenta una binarización sobre redes neuronales profundas y densas, mientras que en la bibliografía se utilizan redes convolucionales.
- Se realizan las tareas de preprocesamiento especificadas en **AdaNet** mientras que en la bibliografía no se utiliza ningún tipo de preprocesamiento.
- Se modificaron los métodos de optimización dado que en pruebas exploratorias se observaron inconsistentes resultados, dichas modificaciones se especifican en cada caso.

La configuración de las primeras iteraciones de pruebas con este algoritmo (véase la tabla 5.11), generaba un desempeño incosistente en cuanto a la precisión sobre los datos de aprendizaje, mostrando el mejor desempeño en las primeras iteraciones del algoritmo (véase la Fig.5.6), obteniendo los resultados ilustrados en la tabla 5.12.

Al reducir el número de iteraciones a la mitad se logró mejorar la precisión del modelo y al mismo tiempo usar menos parámetros, finalmente, el mejor rendimiento fue alcanzado usando solamente 2 iteraciones del modelo **AdaBnn**. Hasta ahora no se observa una mejora importante en cuanto a los objetivos planteados al fijarnos en los tiempos de entrenamiento y precisión del modelo AdaBnn. De nuevo reduciendo la cantidad de iteraciones del modelo, esta vez a sólo dos iteraciones del algoritmo y aumentando el número de épocas a 25, el modelo tuvo un mejor desempeño con muchos menos parámetros que en la primera iteración, el comportamiento de la precisión y costo final se incluye en la Fig. 5.7. Los ajustes realizados para el primer par(*deer-truck*) no generan el mismo comportamiento en el segundo (*deer-horse*), véase tabla 5.13 y las Fig.B.1, sin embargo, al probar de nuevo con **T = 10**, el algoritmo decrece en su rendimiento. En general, se puede observar que los modelos generados por AdaBnn se están viendo afectados por el mecanismo de regularización de la binarización de los pesos en forma negativa, al menos para los datos preprocesados que se le están dando al algoritmo. Los siguientes pares para AdaBnn fueron entonces probados con T=2 dada los resultados hasta ahora.

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	10	binary_crossentropy	RMSprop	100	20
2	150	5	binary_crossentropy	RMSprop	100	20
3	150	2	binary_crossentropy	RMSprop	100	25

Table 5.11 Parámetros por iteración de pruebas de AdaBnn sobre CIFAR(deer-truck)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	383	4	5903550	0.589 ± 0.010
2	143	2	1743300	0.619 ± 0.010
3	51	1	376650	0.639 ± 0.010

Table 5.12 Rendimiento por iteración de pruebas de AdaBnn sobre CIFAR(deer-truck)

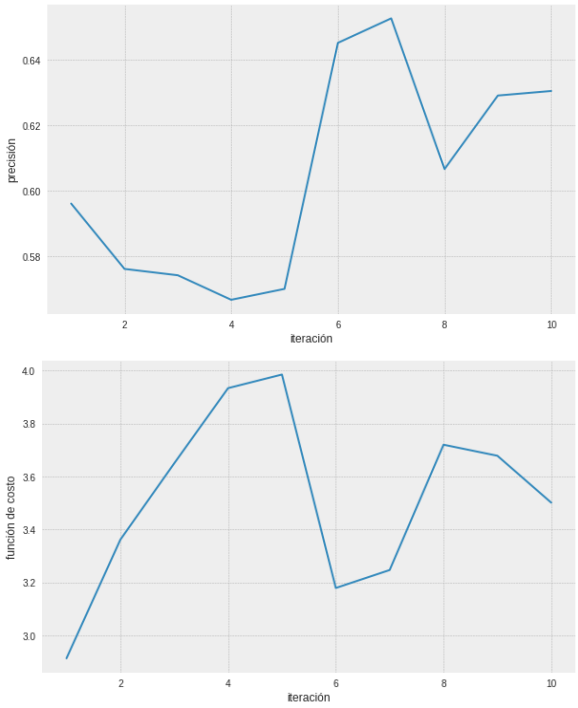


Fig. 5.6 Desempeño de AdaBnn
sobre CIFAR-10 (**deer-truck**) Iteración 1.

Las figuras correspondientes a la precisión sobre el resto de las pruebas con los siguientes pares se adjuntan en la sección de Anexos B.

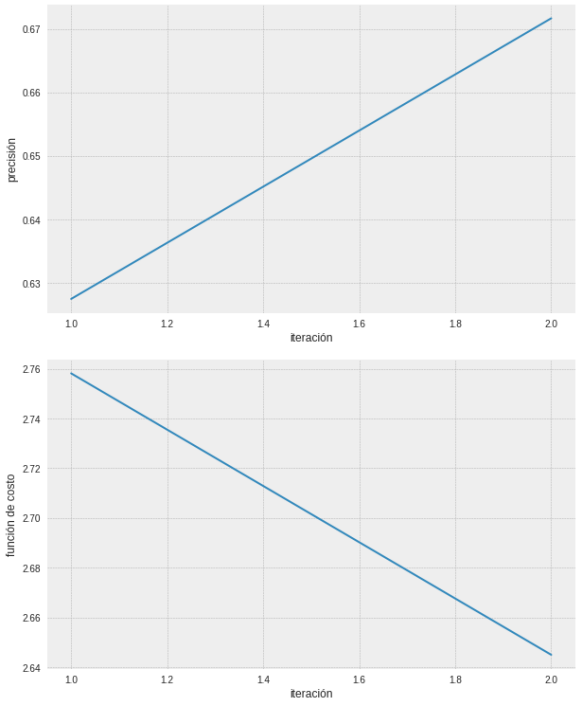


Fig. 5.7 Desempeño de AdaBnn
sobre CIFAR-10 (**deer-truck**) Final.

par	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
deer-horse	51	1	556650	0.525 ± 0.011
automobile-truck	68	1	489150	0.460 ± 0.011
cat-dog	68	1	444150	0.495 ± 0.011
dog-horse	91	1	376650	0.512 ± 0.011

Table 5.13 Rendimiento en las pruebas de AdaBnn sobre los siguientes pares de CIFAR

Resultados generales

A continuación se adjuntan los resultados de precisión general consolidados entre los algoritmos desarrollados **AdaNet** y **AdaBnn** sobre el conjunto de datos **CIFAR-10** en la configuración de clasificación binaria. Es necesario mencionar, que aunque no se alcanzaron los mismos resultados que el artículo original de adanet para dicha implementación, a conocimiento del investigador, la presente es la implementación pública más cercana en términos de la precisión a pesar de que el modelo fuera probado originalmente mediante validación y cruzada y la presente implementación haya sido probada mediante data escondida,

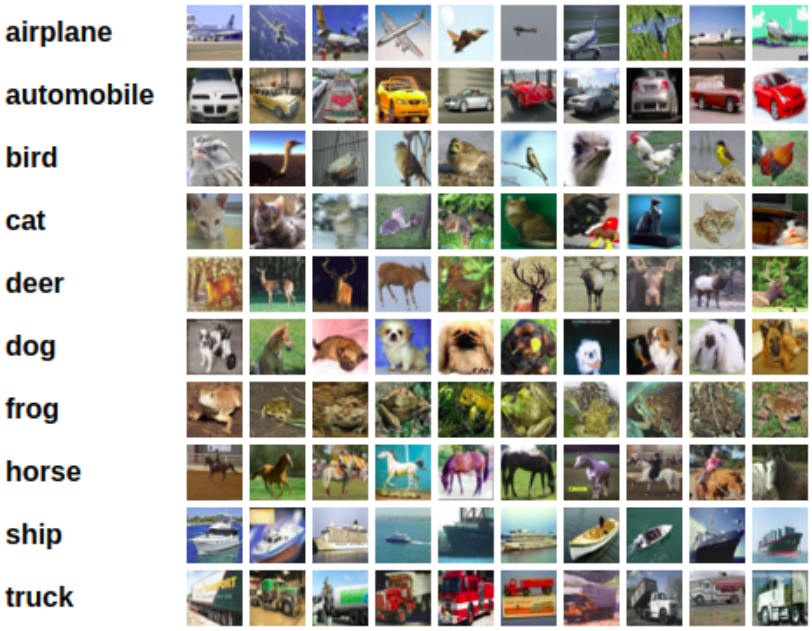


Fig. 5.8 Muestra aleatoria de CIFAR-10[17]

Clases seleccionadas	AdaNet[3]	AdaNet[4]	AdaBnn
deer-truck	**0.937 ± 0.008**	**0.893 ± 0.006**	**0.639 ± 0.010**
deer-horse	**0.843 ± 0.007**	**0.750 ± 0.009**	**0.525 ± 0.011**
automobile-truck	**0.846 ± 0.006**	**0.690 ± 0.010**	**0.460 ± 0.011**
cat-dog	**0.692 ± 0.012**	**0.597 ± 0.010**	**0.495 ± 0.011**
dog-horse	**0.835 ± 0.008**	**0.714 ± 0.010**	**0.512 ± 0.011**

Table 5.14 Precisión general de los algoritmos desarrollados sobre distintos pares del conjunto de datos CIFAR-10

5.2 Clasificación multiclase

Para evaluar los modelos en la configuración de clasificación multiclase se usaron tanto el conjunto de datos **CIFAR-10** cómo el conjunto de datos **MNIST**.

5.2.1 CIFAR-10

Para la evaluación de algoritmos en la tarea de clasificación multiclase de este conjunto de datos, se tomaron 50000 observaciones para entrenar(5000 de cada clase) y 10000 observaciones para evaluar el modelo (1000 de cada clase). A pesar de que las redes

neuronales binarizadas presentadas en [11], fueron probadas sin ningún preprocesamiento o *data augmentation*, en el presente estudio se incluyen para estas redes, las mismas operaciones de preprocesamiento propuestas en [1] y así poder establecer un punto de comparación entre ambos.

AdaNet

Los resultados de la clasificación del algoritmo **AdaNet** sobre el conjunto de datos **CIFAR-10** muestran un interesante comportamiento en cuanto a la cantidad de épocas en el entrenamientode las redes neuronales, al presentar un mejor desempenño con tan sólo 25 épocas y disminuyendo en 0.005 al usar el doble de las épocas. También puede observarse cómo la cantidad de parámetros no implica una mayor precisión del modelo, aún con 20 épocas de entrenamiento.

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	5	categorical_crossentropy	Adam	100	20
2	150	2	categorical_crossentropy	Adam	100	25
3	150	2	categorical_crossentropy	Adam	100	50

Table 5.15 Parámetros por iteración de pruebas de AdaNet sobre CIFAR(multiclase)

t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
462	9	1751400	0.396 ± 0.005
185	5	560700	0.435 ± 0.005
301	5	560700	0.430 ± 0.005

Table 5.16 Rendimiento en las pruebas de AdaNet sobre los siguientes pares de CIFAR

AdaBnn

Los resultados de la clasificación del algoritmo **AdaBnn** sobre el conjunto de datos **CIFAR-10** muestran un interesante comportamiento en cuanto a la cantidad de iteraciones del algoritmo (parámetro T) ya que fijando el resto de los parámetros se observa que entre 10 y 2 iteraciones del modelo, éste último es el que provee mejores resultados, ésto puede ocurrir porque el modelo de mayor cantidad de iteraciones se esté sobreajustando a los datos de entrenamiento. Sin embargo, es necesario mencionar que en el planteamiento multi clase, adabnn tiene un desempeño bastante bajo incluso para la mejor configuración de los parámetros probados.

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	10	categorical_crossentropy	RMSprop	100	20
2	150	2	categorical_crossentropy	RMSprop	100	20

Table 5.17 Parámetros por iteración de pruebas de AdaBnn sobre CIFAR(multiclase)

t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
2081	24	5873400	0.106 ± 0.003
211	6	560700	0.148 ± 0.003

Table 5.18 Rendimiento en las pruebas de AdaNet sobre los siguientes pares de CIFAR

5.2.2 MNIST

Este conjunto de datos se utilizó tal y como es provisto para el problema de clasificación multiclase, esto es, cada observación consta de un dígito y la etiqueta que identifica a dicho dígito. Para evaluar el desempeño de **AdaNet** en este conjunto de datos se modificaron los parámetros T, función de optimización y el número de épocas (tabla 5.19) , con el fin de estudiar el comportamiento del algoritmo con variaciones diferentes a las realizadas en el problema de clasificación binaria. Cómo resultado, se observó un buen desempeño al usar cómo función de optimización RMSprop en contraste con las otras pruebas realizadas para la clasificación binaria, sin embargo, usando la función Adam también se lograron resultados similiares incluso con un menor número de iteraciones (tabla 5.20).

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	5	categorical_crossentropy	RMSprop	100	20
2	150	5	categorical_crossentropy	Adam	100	20
3	150	2	categorical_crossentropy	Adam	100	25

Table 5.19 Parámetros por iteración de pruebas de AdaNet sobre MNIST(multiclase)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	449	11	1704600	0.972 ± 0.001
2	617	11	1704600	0.961 ± 0.002
3	230	6	537300	0.970 ± 0.001

Table 5.20 Rendimiento en las pruebas de AdaNet sobre MNIST

En este escenario a pesar de poseer ciertas inconsistencias, el algoritmo **AdaBnn** tuvo un mejor desempeño desde la primera iteración, fijando el parámetro de número de épocas esta vez, se logra notar que el algoritmo resulta sensible a la función de optimización ya que con el resto de los parámetros iguales, si la función de optimización es Adam o RMSprop, se generan los mejores y peores resultados de desempeño del algoritmo, respectivamente.

iteración	B	T	f. costo	f. optimización	tamaño de lote	# épocas
1	150	10	categorical_crossentropy	Adam	100	20
2	150	2	categorical_crossentropy	Adam	100	20
3	150	5	categorical_crossentropy	Adam	100	20
4	150	2	categorical_crossentropy	RMSprop	100	20
5	150	5	categorical_crossentropy	RMSprop	100	20
6	150	10	categorical_crossentropy	RMSprop	100	20

Table 5.21 Parámetros por iteración de pruebas de AdaBnn sobre MNIST(multiclase)

iteración	t. entrenamiento(seg)	t. inferencia(seg)	# parámetros	precisión (95% conf)
1	2923	28	5000100	0.937 ± 0.002
2	130	5	537300	0.847 ± 0.003
3	537	10	1704600	0.868 ± 0.003
4	130	5	537300	0.732 ± 0.004
5	509	10	1704600	0.880 ± 0.003
6	1981	10	1704600	0.107 ± 0.003

Table 5.22 Rendimiento en las pruebas de AdaBnn sobre MNIST

En las experiencias realizadas con el conjunto de datos MNIST a pesar que representan un menor desempenño con respecto al artículo de redes neuronales binarizadas, futuras pruebas pueden encontrar una configuración de parámetros que capture la complejidad existente ene el conjunto de datos y genere mejores resultados. Es necesario mencionar que los resultados de dicho artículo fueron producto de 1000 épocas de entrenamiento mientras que las redes neuronales planteadas en esta investigación se mantienen un orden muy por debajo de este número, siendo máximo 20 en cada caso, adicionalmente los resultados de **Bnn** fueron obtenidos, al igual que en la presente investigación, mediante data escondida.

Bnn [5]	AdaNet[6]	AdaBnn
0.990	$\mathbf{0.972 \pm 0.001}$	$\mathbf{0.937 \pm 0.002}$

Table 5.23 Precisión general sobre el conjunto de datos **MNIST**

Capítulo 6

Conclusiones

En el presente trabajo de investigación se diseñaron, implementaron y compararon redes neuronales profundas concebidas mediante aprendizaje estructural adaptativo y cuyos pesos fueron restringidos al conjunto $\{-1, 1\}$, dichas redes neuronales fueron evaluadas en el problema de clasificación de imágenes tanto en su configuración binaria cómo multiclase. Para realizar estas actividades fue necesaria la comprensión del artículo original de **AdaNet**, y replica de las configuraciones del entorno de experimentación de dicho artículo, así cómo el estado del arte de los problemas de clasificación planteados para evaluar los modelos resultantes. Al momento de implementar los algoritmos desarrollados se investigaron las principales herramientas y tecnologías usadas en el área, teniendo como prioridad la facilidad de uso e integración con otras tecnologías o herramientas de otras etapas del ciclo de resolución de problemas de clasificación y aprendizaje automático en general. Dada la novedad de los algoritmos desarrollados, los esfuerzos de experimentación fueron dirigidos a probar la mayor cantidad de variantes en la configuración de parámetros de los algoritmos y así entender su funcionamiento en los problemas planteados, esta consideración requirió de varias iteraciones en algunos casos. La métrica principal considerada para la evaluación de los modelos fue la precisión dado que de esta forma se pueden realizar comparaciones directas con los algoritmos alternativos en el problema de aprendizaje seleccionado.

Durante el entrenamiento y evaluación de los modelos desarrollados, se pudo observar patrones cómo la inconsistencia de precisión por parte del modelo **AdaBnn** en los problemas de clasificación binaria para el conjunto **CIFAR-10**, característica principalmente atribuída a la binarización de los pesos de la red, dado que el algoritmo **AdaNet** aplicado sobre el mismo conjunto de datos presenta consistencia en cuanto a la precisión y funciones de costo obtenidas aún con pocas iteraciones, también se observaron fenómenos cómo la predominación de redes neuronales de pocas capas en términos de la precisión obtenida, al usar el algoritmo **AdaBnn**.

En la clasificación multiclase, se pudo observar un mejor rendimiento para ambos algoritmos, con las mismas configuraciones de parámetros en que en los problemas de clasificación binaria, pero también con redes neuronales un poco más complejas.

Finalmente, se puede concluir que los objetivos planteados en el capítulo 1 fueron alcanzados tras haber realizado la integración del aprendizaje estructural adaptativo en redes neuronales binarizadas.

6.1 Contribuciones

Las principales contribuciones de esta investigación son las siguientes:

- Se realizó una revisión profunda de la bibligrafiía sobre la métodología de aprendizaje estructural adaptativo en redes neuronales.
- Se diseñó un algoritmo que integrará las propiedades de las redes neuronales binarizadas y la metodología de entrenamiento de aprendizaje estructural adaptativo.
- Se desarrolló e hizo disponible el código necesario para usar el algoritmo resultante y replicar los resultados alcanzados en la presente investigación.
- Se comparó el desempeño de los algoritmos resultantes con sus antecesores **AdaNet** y Redes neuronales binarizadas.

6.2 Limitaciones

La principal limitación de esta investigación fue no contar con los recursos de hardware necesarios para reducir los tiempos de entrenamiento e inferencia de los modelos con el fin de aumentar la cantidad de iteraciones realizadas con los algoritmos, permitiendo así obtener un conocimiento exhaustivo del funcionamiento de los mismos con diferentes parámetros.

6.3 Trabajos futuros

Tras la realización de esta investigación se proponen los siguientes trabajos futuros:

- Documentar exhaustivamente el rendimiento de los algoritmos desarrollados con una mayor variedad de parámetros probados.

- Analizar el uso de los algoritmos desarrollados sobre conjuntos de datos de aprendizaje resultantes de diferentes metodologías de muestreo.
- Establecer un mecanismo de compatibilidad entre las convoluciones y el aprendizaje estructural adaptativo para el caso específico de visión por computador dado que, las primeras han provisto muy buenos resultados en dichos problemas.
- Extender la evaluación de los algoritmos desarrollados a otras areas cómo por ejemplo, la clasificación de sentimientos en texto o la clasificación de audio.

Referencias

[1] Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., and Yang, S. (2016). Adanet: Adaptive structural learning of artificial neural networks. *CoRR*, abs/1607.01097.

[2] Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131.

[3] Cybenko, G. (1992). Approximation by superpositions of a sigmoidal function. *MCSS*, 5(4):455.

[4] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

[5] Glorot, X. and Bengio, Y. (2010a). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256.

[6] Glorot, X. and Bengio, Y. (2010b). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

[7] Graham, B. (2014). Fractional max-pooling. *CoRR*, abs/1412.6071.

[8] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1764–1772.

[9] Heaton, J. (2013). *Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms*. CreateSpace Independent Publishing Platform.

[10] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., and Sainath, T. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29:82–97.

[11] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4107–4115.

[12] Hughes, M., Li, I., Kotoulas, S., and Suzumura, T. (2017). Medical text classification using convolutional neural networks. *CoRR*, abs/1704.06841.

[13] Ian Goodfellow, Yoshua Bengio, A. C. (2016). *Deep learning*.

[14] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[15] Jafari, M. H., Nasr-Esfahani, E., Karimi, N., Soroushmehr, S. M. R., Samavi, S., and Najarian, K. (2016). Extraction of skin lesions from non-dermoscopic images using deep learning. *CoRR*, abs/1609.02374.

[16] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[17] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

[18] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114.

[19] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[20] Marsland, S. (2014). *Machine Learning. An Algorithmic Perspective.* CRC.

[21] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[22] Murphy, K. P. (2013). *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.].

[23] Nagi, J., Ducatelle, F., Caro, G. A. D., Ciresan, D. C., Meier, U., Giusti, A., Nagi, F., Schmidhuber, J., and Gambardella, L. M. (2011). Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications, ICSIPA 2011, Kuala Lumpur, Malaysia, November 16-18, 2011*, pages 342–347.

[24] Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244.

[25] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[26] Sundararajan, D. (2017). *Digital Image Processing: A Signal Processing and Algorithmic Approach*. Springer.

[27] Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 171–176.

[28] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

[29] Tan, P.-N. et al. (2006). *Introduction to data mining*. Pearson Education India.

[30] Torralba, A., Fergus, R., and Freeman, W. T. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970.

[31] Trevor Hastie, Robert Tibshirani, J. F. (2008). *The Elements of Statistical Learning. Data Mining, Inference and Prediction*. Springer.

[32] Van Dyk, D. A. and Meng, X.-L. (2001). The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50.

[33] W. McCulloch, W. P. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(115-133).

[34] Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066.

[35] Yamada, Y., Iwamura, M., and Kise, K. (2018). Shakedrop regularization.

Appendix A

Notación

Este apartado provee una referencia descriptiva de la notación usada a lo largo del documento.

A.1 Números y vectores

- x: Valor Entero o real.
- W: Vector

A.2 Índices

- x_i: Elemento i del vector x comenzando en 1.
- W^t: Vector con valores definidos en la unidad de tiempo t.

A.3 Conjuntos

- $\mathbb{A}$: Conjunto.
- $\mathbb{A} \subset \mathbb{B}$: A es un subconjunto de B.
- $\{0,1\}$: El conjunto que contiene 0 y 1
- $[N]$: Conjunto de los elementos desde 1 hasta N ó $\{1,\ldots,N\}$

A.4 Funciones

- $\mathbb{A} \mapsto \mathbb{B}$: Una función que va del dominio $\mathbb{A}$ al rango $\mathbb{B}$.
- $f \circ g$: Composición de las funciones f y g.
- $||x||$: Norma-2 de x
- $||x||_p$: Norma-p de x

Appendix B

Resultados adicionales de los experimentos

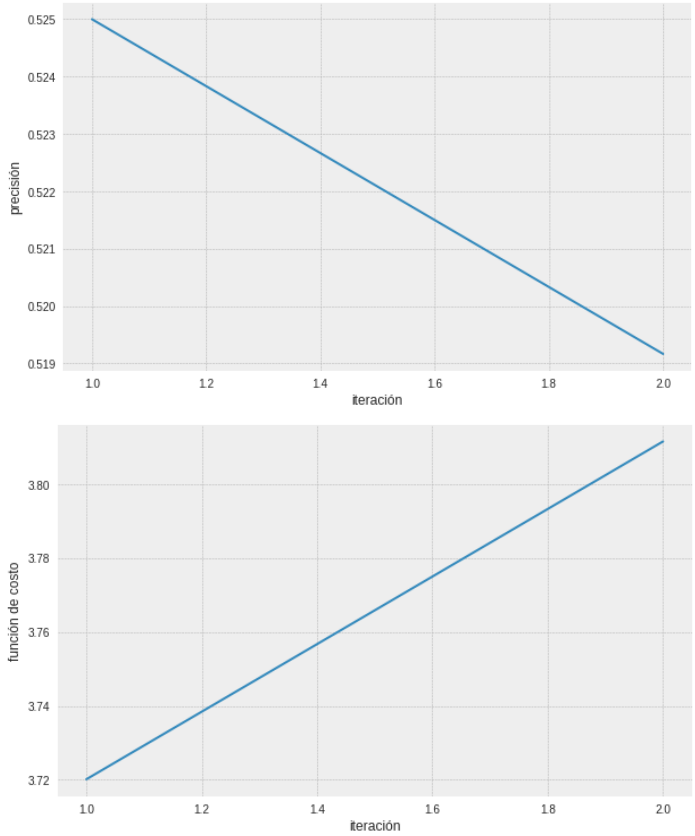


Fig. B.1 Desempeño de AdaBnn sobre CIFAR-10 (**deer-horse**) Final.

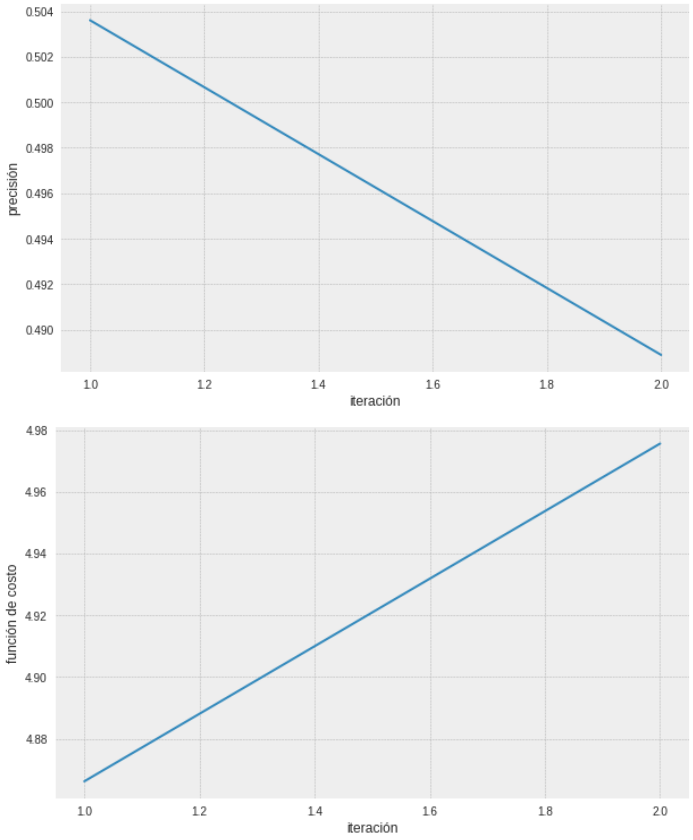


Fig. B.2 Desempeño de AdaBnn
sobre CIFAR-10 (**automobile-truck**) Final.

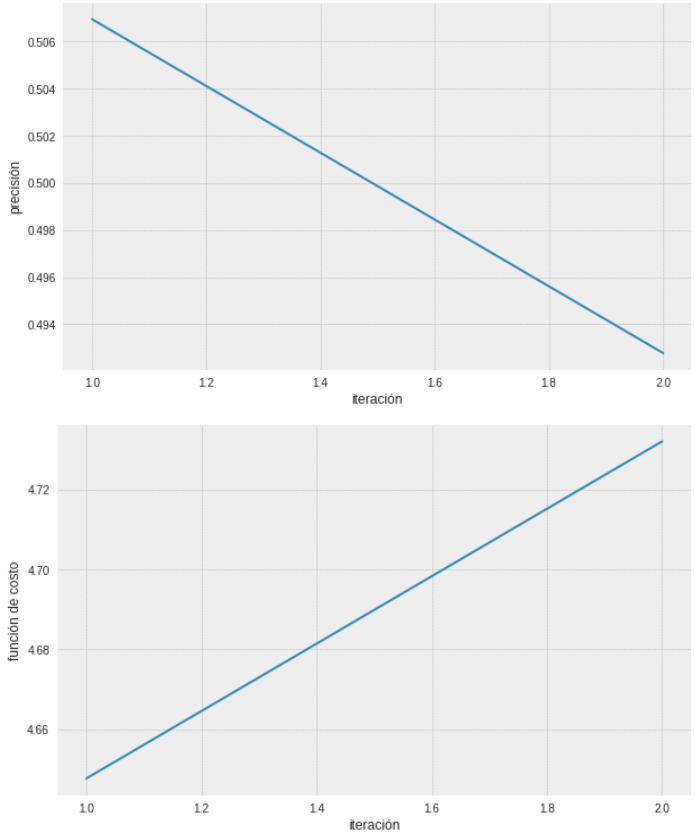


Fig. B.3 Desempeño de AdaBnn
sobre CIFAR-10 (**cat-dog**) Final.

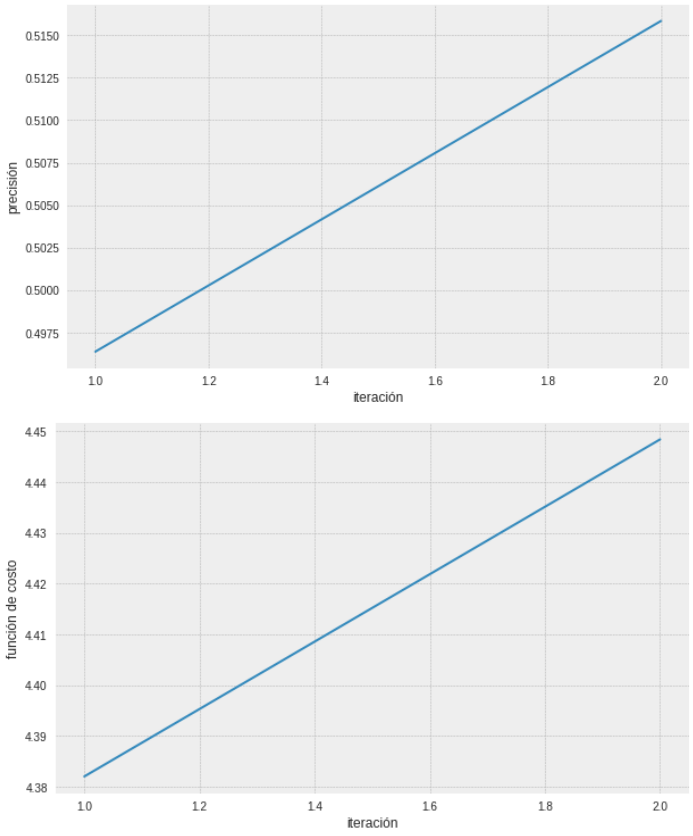


Fig. B.4 Desempeño de AdaBnn
sobre CIFAR-10 (**dog-horse**) Final.