

INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



Generación de sucesiones utilizando redes neuronales recurrentes

TESIS

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS

PRESENTA

GEORGES BÉLANGER ALBARRÁN

ASESOR: M.Sc. JOSÉ MANUEL RODRIGUEZ SOTELO

“AI” wants only the genuine article: machines with minds, in the full and literal sense. This is not science fiction, but real science, based on a theoretical conception as deep as it is daring: namely, we are, at root, computers ourselves

- John Haugeland, Artificial Intelligence,

Índice general

1. Introducción	1
2. Conceptos de aprendizaje de máquina	3
2.1. Introducción	3
2.2. Algoritmos de aprendizaje	3
2.2.1. La tarea, T	3
2.2.2. La medida de desempeño, P	4
2.2.3. La experiencia, E	4
2.3. Capacidad, sobreajuste y subajuste	5
2.3.1. Búsqueda de distribuciones útiles	6
2.3.2. Regularización	7
2.4. Hiperparámetros y conjuntos de validación	7
2.5. Algoritmo de gradiente estocástico	8
2.6. Cómo construir un algoritmo de aprendizaje	9
2.7. Limitaciones en aprendizaje de máquina tradicional	9
2.7.1. La maldición de la dimensión	10
3. Conceptos de aprendizaje profundo	11
3.1. Introducción	11
3.2. Redes neuronales prealimentadas	12
3.2.1. Aprendizaje con descenso de gradiente	14
3.2.1.1. Funciones de pérdida	14
3.2.1.2. Unidades de salida	14
3.2.1.2.1. Redes de densidad mixta	15
3.2.2. Unidades ocultas	15
3.2.3. Diseño de la arquitectura de la red	15
3.2.4. Propagación hacia atrás	16
3.2.5. Propagación hacia atrás con gráficas computacionales	16

ÍNDICE GENERAL

3.3.	Regularización en aprendizaje profundo	17
3.3.1.	Detención temprana	17
3.3.2.	Dropout	18
3.4.	Optimización en aprendizaje profundo	18
3.4.1.	Momentum	19
3.4.2.	Algoritmos con tasas de aprendizaje adaptativas	19
3.5.	Redes neuronales recurrentes	19
3.5.1.	Desdoblado gráficas computacionales	20
3.5.2.	Redes neuronales recurrentes	21
3.5.3.	Dependencias de largo plazo	23
3.5.4.	Recorte de gradiente	23
3.6.	Memoria larga de corto plazo	24
4.	Experimento con texto	27
4.1.	Introducción	27
4.2.	Descripción de las bases de datos	28
4.2.1.	Rayuela	28
4.2.2.	La Biblia	28
4.2.3.	Concurso Hutter	28
4.3.	Resultados	28
4.3.1.	Modelo Rayuela	29
4.3.2.	Modelo Biblia	30
4.3.3.	Modelo Hutter	33
5.	Experimento con caligrafía en línea	36
5.1.	Introducción	36
5.2.	Descripción de la base de datos	37
5.3.	Descripción del modelo	37
5.3.1.	Red de densidad mixta	38
5.3.2.	Mecanismos de atención	40
5.4.	Resultados	41
6.	Conclusiones	45

CAPÍTULO 1

INTRODUCCIÓN

El intento por poder entender, analizar y, más importante, reproducir la inteligencia, no es una búsqueda que se pueda atribuir únicamente a los últimos años. Durante siglos, los humanos han soñado con crear máquinas que piensen. Con la llegada de las computadoras programables, ese sueño ha tomado grandes pasos. Hoy en día, la inteligencia artificial es una disciplina con un gran rango de aplicaciones y temas de investigación activos. Por ejemplo, se utiliza software inteligente para manejar coches automáticamente, predecir el precio de acciones en el futuro o hacer diagnósticos en medicina. [6] [9] [10]

En sus comienzos, la inteligencia artificial pudo resolver problemas intelectualmente complicados para los humanos, pero relativamente fáciles para las computadoras: problemas que se pueden describir con una lista de reglas formales o matemáticas. El verdadero reto de la inteligencia artificial resultó ser resolver tareas que son fáciles para las personas de realizar, pero difíciles para las personas de describir formalmente, problemas que resolvemos intuitivamente, que sentimos automáticos, como comprender palabras habladas o reconocer caras de personas en imágenes. [6]

Una solución que se ha encontrado para resolver estos problemas es permitir a las computadoras aprender de la experiencia y entender al mundo en términos de una jerarquía de conceptos, con cada concepto siendo definido en términos de su relación con conceptos más sencillos. Si dibujamos una gráfica mostrando cómo estos conceptos se construyen uno encima del otro, el gráfico es profundo con muchas capas. Por esta razón, este enfoque de inteligencia artificial se conoce con el nombre de aprendizaje profundo. [4] [6]

En este trabajo, se abordará el aprendizaje profundo desde un punto de vista teórico y práctico. La parte teórica se basará en el libro "Deep Learning" de Ian Goodfellow, Yoshua Bengio y Aaron Courville. En específico, el capítulo 2 explicará los conceptos básicos de aprendizaje de máquina y el capítulo 3 describirá el aprendizaje profundo: redes neuronales,

redes neuronales recurrentes y, finalmente, el modelo que se utilizará en la parte práctica, redes neuronales recurrentes con "Long Short-Term Memory".

Los capítulos 4 y 5 se basarán en los experimentos realizados por Alex Graves, investigador de Google Deepmind, en su artículo "Generating Sequences With Recurrent Neural Networks", publicado en 2013. En el capítulo 4, se generarán sucesiones de texto y en el capítulo 5 se generarán sucesiones de caligrafía en línea. Por último, en el capítulo 6 se harán conclusiones sobre los experimentos y describirán algunas nuevas posibilidades de investigación. [8]

Una de las primeras preguntas que pueden surgir sobre este trabajo es la siguiente: ¿por qué generar sucesiones? Graves explica que la razón más importante de hacer esto es para entender mejor a los datos que se están analizando. El ser capaz de modelar todo sobre los datos forza a tu sistema a encontrar la estructura prominente de los datos y al modelo a entender los datos que está viendo. Otro beneficio importante es que, al ver lo que se está generando, será posible saber qué tanto ha realmente entendido el modelo. [7] [8]

La inteligencia artificial promete traer una nueva revolución industrial en los próximos 10 o 20 años. Esta traerá más "poder" al mundo en cierto sentido. Pero, con más poder, también vendrá más responsabilidad de hacer lo correcto. Dado el estado actual en que vive el mundo: crecimiento del nacionalismo, de la intolerancia y una gran tensión entre muchos países, será muy importante que los científicos y la sociedad en general nos concentremos en ayudarnos los unos a los otros. El discurso de Charles Chaplin en "El gran dictador" está hoy más presente que nunca: "Luchemos para liberar al mundo. Para derribar barreras nacionales. Para eliminar la ambición, el odio y la intolerancia. Luchemos por un mundo de razón, un mundo en el que la ciencia y el progreso nos conduzcan a todos a la felicidad". [6]

CAPÍTULO 2

CONCEPTOS DE APRENDIZAJE DE MÁQUINA

2.1. Introducción

El aprendizaje profundo es un tipo específico de aprendizaje de máquina. Por lo tanto, para abordar el aprendizaje profundo con más naturalidad, se explicarán los conceptos básicos de aprendizaje de máquina en este capítulo. Se comenzará este capítulo explicando lo que es un algoritmo de aprendizaje. Después se explicarán algunas propiedades importantes de ellos: su capacidad, sus hiperparámetros y el algoritmo de optimización más utilizado, gradiente estocástico. Se finalizará el capítulo hablando sobre las limitaciones que se han tenido al utilizar algoritmos de aprendizaje de máquina tradicional.

2.2. Algoritmos de aprendizaje

Un algoritmo de aprendizaje de máquina es un algoritmo que es capaz de aprender de datos. Pero, ¿qué quiere decir exactamente aprender? Tom Mitchell, en su libro "Machine Learning", da la siguiente definición: un programa de computadora, se dice que aprende de la experiencia E con respecto a alguna clase de tareas T y una medida de desempeño P , si su desempeño en las tareas T , al ser medido por P , mejora con la experiencia E . A continuación, se presentarán algunos ejemplos de tareas T , experiencias E y medidas de desempeño P . [14]

2.2.1. La tarea, T

El aprendizaje de máquina permite abordar tareas que son muy difíciles de resolver con programas fijos escritos y diseñados por humanos. En esta definición de la palabra "tarea",

el proceso de aprender no es en sí mismo la tarea. Aprender es nuestro medio de conseguir la habilidad para realizar la tarea. Por ejemplo, si se quiere que un robot sea capaz de manejar, entonces manejar será la tarea. Se podría programar al robot para que aprenda a manejar, o se podría intentar escribir directamente un programa que especifique como manejar manualmente. [6] [14]

Las tareas en aprendizaje de máquina se describen en términos de cómo el sistema de aprendizaje de máquina debería procesar un ejemplo. Un ejemplo es una colección de características que han sido cuantitativamente medidas de un objeto o evento que queremos que el sistema procese. Generalmente representamos un ejemplo como un vector $x \in R^n$ donde cada entrada x_i del vector es otra característica. Por ejemplo, las características de una imagen son generalmente los valores de los píxeles de la imagen. En el ejemplo de programar a un robot para que aprenda a manejar, las características serían los valores de los píxeles de todas las imágenes del video. [6]

Muchas tareas se pueden resolver con aprendizaje de máquina. Algunas de las más comunes son las siguientes: clasificación, regresión, transcripción, traducción, detección de anomalías, síntesis y muestreo. [6]

2.2.2. La medida de desempeño, P

Para poder evaluar las habilidades de un algoritmo de aprendizaje de máquina, se debe diseñar una medida cuantitativa de su desempeño. Para tareas tales como clasificación y transcripción, generalmente se mide la precisión del modelo. La precisión es simplemente la proporción de ejemplos para los cuales el modelo produce el resultado correcto. También se puede, de manera equivalente, medir la tasa de error. En el ejemplo del robot manejando, la medida de desempeño podría ser el porcentaje de decisiones que llevan a un accidente.

Usualmente, se busca observar qué tan bien el algoritmo se desempeña con datos que no ha visto antes, puesto que esto determina qué tan bien funcionará cuando sea usado en el mundo real. Por lo tanto, se evalúan estas medidas de desempeño utilizando un conjunto de prueba con datos que están separados de los datos que se utilizaron para entrenar al sistema de aprendizaje de máquina. [6]

2.2.3. La experiencia, E

Los algoritmos de aprendizaje de máquina pueden ser categorizados como supervisados o no supervisados de acuerdo al tipo de experiencia que se le permite tener durante el proceso de entrenamiento.

En los algoritmos de aprendizaje supervisado o de predicción, se experimenta con un conjunto de datos conteniendo características, pero también, a cada ejemplo se le asigna una

etiqueta. Es decir, el objetivo es aprender a mapear los datos de entrada x hacia las etiquetas o datos de salida y partiendo de un conjunto de entrenamiento $D = \{(x_i, y_i)\}_{i=1}^N$. En este ejemplo, D es el conjunto de entrenamiento y N es el número de ejemplos de entrenamiento. [15]

En los algoritmos de aprendizaje no supervisado, o de descripción, se experimenta solamente el conjunto de datos $D = \{(x_i)\}_{i=1}^N$, y el objetivo es encontrar patrones interesantes en los datos. Por lo que regresando al ejemplo del robot que maneja, la experiencia son las imágenes que el robot recibe y, si es aprendizaje supervisado, las etiquetas podrían ser la decisión que debería tomar el robot en ese momento: frenar, acelerar, girar a la derecha, etc. [15]

2.3. Capacidad, sobreajuste y subajuste

El reto principal del aprendizaje de máquina es que debe tener buen desempeño en datos de entrada nuevos que nunca se hayan visto, no solamente en los datos con los que se entrenó al algoritmo. La habilidad de desempeñarse bien en datos no antes vistos se llama generalización.

Generalmente, cuando se entrena un modelo de aprendizaje de máquina, se tiene acceso a un conjunto de entrenamiento y se calcula una medida de error en este conjunto, la cual se conoce con el nombre de error de entrenamiento. En una primera etapa, este error es el que se quiere reducir. Sin embargo, esto es simplemente un problema de optimización. Lo que separa al aprendizaje de máquina de la optimización es que se busca que el error de generalización, también conocido como error de prueba, sea bajo también. [6]

El error de generalización se define como el valor esperado del error en un nuevo dato de entrada. Generalmente se estima el error de generalización de un modelo de aprendizaje de máquina al medir su desempeño con respecto a un conjunto de prueba con ejemplos que fueron recolectados de manera separada al conjunto de entrenamiento.

En la práctica, los factores que determinan qué tan bien se desempeñará un algoritmo de aprendizaje de máquina dependerá de su habilidad para hacer lo siguiente:

- 1.- Reducir el error de entrenamiento
- 2.- Hacer que la brecha entre el error de entrenamiento y el error de prueba sea pequeña

Estos dos factores corresponden a los dos retos centrales en aprendizaje de máquina: subajuste y sobreajuste. El subajuste ocurre cuando el modelo no es capaz de obtener un error suficientemente pequeño en el conjunto de entrenamiento. El sobreajuste ocurre cuando la brecha entre el error de entrenamiento y el error de prueba es demasiado grande.

Se puede controlar si un modelo es más probable que sobreajuste o subajuste al modificar su capacidad. Informalmente, la capacidad de un modelo es su habilidad para ajustar una variedad de funciones. Modelos con baja capacidad tendrán dificultades para ajustar al conjunto de entrenamiento. Modelos con una alta capacidad pueden sobrestimar al memorizar propiedades del conjunto de entrenamiento que no le funciona bien en el conjunto de prueba. [6]

Una forma en que se puede controlar la capacidad de un algoritmo de aprendizaje es al escoger el espacio de hipótesis, es decir, el conjunto de funciones entre las que el algoritmo puede escoger como soluciones. Por ejemplo, el algoritmo de regresión lineal tiene al conjunto de todas las funciones lineales de sus datos de entrada como su espacio de hipótesis. Se puede generalizar a la regresión lineal para incluir polinomios en su espacio de hipótesis, en vez de solo funciones lineales. Al hacer esto, se incrementa la capacidad del modelo.

Los algoritmos de aprendizaje de máquina generalmente tendrán mejor desempeño cuando su capacidad sea apropiada para la verdadera complejidad de la tarea que se tiene que realizar y la cantidad de datos de entrenamiento que se le da. Modelos con capacidad insuficiente son incapaces de resolver tareas complejas. Modelos con alta capacidad son capaces de resolver tareas complejas, pero, cuando su capacidad es mayor que la necesitada para resolver la tarea en cuestión, pueden sobrestimar los datos.

2.3.1. Búsqueda de distribuciones útiles

Una parte muy importante del aprendizaje de máquina es idear diferentes modelos y diferentes algoritmos para ajustarlos. Sin embargo, se puede demostrar que no hay un modelo que sea universalmente mejor que los demás. Esta idea la formalizó David Wolpert en 1996 y se conoce como el teorema de “no free lunch”. Este teorema expone que, promediando sobre todas las posibles distribuciones de datos, cada algoritmo de clasificación tiene la misma tasa de error. [15] [25]

Afortunadamente, estos resultados son válidos solo cuando promediamos sobre todas las posibles distribuciones de datos. Si se hacen supuestos sobre los tipos de distribuciones de probabilidad que se encuentran en el mundo real, se pueden diseñar algoritmos de aprendizaje que tengan buen desempeño en estas distribuciones.

Esto significa que el objetivo del aprendizaje de máquina no es buscar un algoritmo universal de aprendizaje. El objetivo es encontrar qué tipos de distribuciones son útiles para el mundo real y qué tipo de algoritmos se desempeñan bien en datos obtenidos de distribuciones que nos interesan. [6]

2.3.2. Regularización

El teorema de “no free lunch” implica que se debe diseñar los algoritmos para que se desempeñen bien en tareas específicas. Esto se hace construyendo un conjunto de preferencias dentro del algoritmo. Cuando estas preferencias están alineadas con los problemas de aprendizaje que se le pide al algoritmo resolver, se obtiene un mejor desempeño. [6]

Por ejemplo, se puede modificar el criterio de entrenamiento de una regresión lineal para incluir decaimiento de pesos. Para realizar una regresión lineal con decaimiento de pesos, se minimiza una suma que contiene al error cuadrático medio del entrenamiento y un criterio $J(w)$ que expresa una preferencia para que los pesos tengan una norma menor. En específico:

$$J(w) = ECM_{\text{entrenamiento}} + \lambda w^T w \quad (2.3.1)$$

donde λ es un valor escogido con anterioridad que controla nuestra preferencia por pesos más pequeños. Cuando $\lambda = 0$, no se impone preferencia, y cuando se hace crecer a λ se fuerza a los pesos a hacerse más pequeños. Al minimizar $J(w)$, se obtiene una elección de pesos que compensa entre ajustar los datos de entrenamiento y mantenerlos pequeños.

Hay otras formas de expresar preferencias por diferentes soluciones. Estos enfoques se conocen con el nombre de regularización. Regularización es cualquier modificación que se hace a un algoritmo de aprendizaje para reducir su error de generalización pero no su error de entrenamiento. La regularización es uno de los temas centrales del aprendizaje de máquina, comparado solo en importancia con la optimización. [6]

2.4. Hiperparámetros y conjuntos de validación

La mayoría de los algoritmos de aprendizaje tienen diferentes parámetros que se pueden usar para controlar el comportamiento del algoritmo. Estos parámetros se conocen como hiperparámetros. Los valores de los hiperparámetros no se adaptan por el algoritmo en sí.

Algunas veces, un parámetro se escoge como hiperparámetro debido a que es difícil que el algoritmo lo optimice. Con más frecuencia, un parámetro se escoge como hiperparámetro debido a que no es apropiado aprender este con el conjunto de entrenamiento. Esto aplica a todos los hiperparámetros que controlan la capacidad del modelo. Si se aprendieran estos con el conjunto de entrenamiento, los hiperparámetros siempre escogerían el modelo con la mayor capacidad, resultando en un sobreajuste. [6]

Para resolver este problema, se utiliza un conjunto de validación con ejemplos que el algoritmo de entrenamiento no observa. El conjunto de validación servirá para ajustar los hiperparámetros. Por lo tanto, este conjunto debe estar separado del conjunto de prueba

para no subestimar el error de generalización. Es decir que el conjunto de validación debe provenir de los datos de entrenamiento.

En específico, se divide al conjunto de entrenamiento en dos subconjuntos disjuntos. Uno de estos subconjuntos se usa para aprender los parámetros y se conoce como los datos de entrenamiento. El otro es el conjunto de validación, usado para estimar el error de generalización durante o después del entrenamiento y así permitir que los hiperparámetros se ajusten de manera adecuada. Cuando la optimización de los hiperparámetros termine, se puede estimar el error de generalización utilizando el conjunto de prueba. [6] [9]

2.5. Algoritmo de gradiente estocástico

En aprendizaje de máquina, uno de los aspectos más importantes del algoritmo es el método de optimización que se utiliza. El algoritmo de optimización que más se emplea en aprendizaje profundo es el algoritmo de gradiente estocástico y este es una extensión del algoritmo de descenso de gradiente.

Este algoritmo funciona bien cuando se utilizan bases de datos muy grandes. Se sabe que un problema grande del aprendizaje de máquina es que se necesita una gran cantidad de datos para obtener buenas generalizaciones. Sin embargo, hacer crecer las bases de datos también hace crecer el costo computacional. [6]

Sin embargo, con gradiente estocástico, se busca que el costo computacional no crezca al tomar solo una parte de los ejemplos para calcular los gradientes. Es decir que si nuestra función de pérdida se puede descomponer como una suma sobre los ejemplos de entrenamiento, entonces:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m L(x^{(i)}, y^{(i)}, \theta) \quad (2.5.1)$$

Lo que hace descenso de gradiente es calcular:

$$\nabla_{\theta} J(\theta) = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(x^{(i)}, y^{(i)}, \theta) \quad (2.5.2)$$

En vez de calcular la suma sobre todos los ejemplos, se puede seleccionar una “mini-batch” de ejemplos $\mathbb{B} = \{x^{(1)}, \dots, x^{(m')}\}$ de entre los ejemplos de entrenamiento. El tamaño de la “mini-batch” de ejemplos m' se escoge de tal forma que sea un número relativamente bajo, entre 1 y unos cuantos cientos. La estimación del gradiente que se obtiene es:

$$g = \frac{1}{m'} \sum_1^{m'} \nabla_{\theta} L(x^{(i)}, y^{(i)}, \theta) \quad (2.5.3)$$

usando ejemplos de la “mini-batch” \mathbb{B} . Después de calcular el gradiente estocástico, el algoritmo sigue el descenso estimado:

$$\theta \leftarrow \theta - \epsilon g \quad (2.5.4)$$

donde ϵ es la tasa de aprendizaje. [6]

2.6. Cómo construir un algoritmo de aprendizaje

Casi todos los algoritmos de aprendizaje profundo tienen la siguiente composición: se combina un conjunto de datos, una función de costo, un procedimiento de optimización y un modelo. Como se pueden remplazar estos componentes casi de manera independiente, es posible contruir una gran cantidad de modelos al hacer cambios en alguna parte de la composición.

La función de costo generalmente incluye al menos un término que hace que el proceso de aprendizaje realice una estimación estadística. La función de costo más común es log-verosimilitud negativa, por lo que al minimizar la función de costo se hace una estimación de máxima verosimilitud. La función de costo también puede contener términos adicionales, tales como términos de regularización. Si nuestro algoritmo es no lineal, entonces las funciones de costo no podrán ser optimizadas en una forma cerrada. Esto obliga a utilizar un procedimiento de optimización numérica iterativa, como descenso de gradiente. [6]

2.7. Limitaciones en aprendizaje de máquina tradicional

Los algoritmos de aprendizaje de máquina tradicional tienen buen desempeño en una gran variedad de problemas importantes. Sin embargo, no han tenido éxito en resolver los problemas principales de inteligencia artificial como interpretar lenguaje hablado o reconocer objetos en imágenes. El desarrollo del aprendizaje profundo fue motivado en parte por el fracaso de los algoritmos tradicionales para generalizar bien en dichas tareas.

A continuación, se expondrá un problema que aparece para aprender de los datos cuando estos tienen dimensiones muy grandes. Trabajar en espacios de dimensión alta genera gran-

des costos computacionales. El aprendizaje profundo fue creado para resolver este y otros obstáculos. [6]

2.7.1. La maldición de la dimensión

Parecería razonable que con un conjunto de datos grande siempre sería posible aproximar bien una función. Para hacer esto solo bastaría tomar un gran vecindario de observaciones cercano a cualquier x y promediarlos. Sin embargo, este enfoque deja de funcionar por un fenómeno conocido como la maldición de la dimensión. [1] [9] [15]

Este concepto se expondrá con un ejemplo de (Hastie et al. 2001, p22). Supongamos que se utiliza un método de vecinos más cercanos para datos de entrada uniformemente distribuidos en un hipercubo unitario p -dimensional. Supongamos que se emplea una vecindad hipercúbica alrededor de un punto objetivo para capturar una fracción r de las observaciones. Como esto corresponde a una fracción r del volumen unitario, el tamaño esperado del borde será $e_p(r) = r^{\frac{1}{p}}$. En 10 dimensiones $e_{10}(0.01) = 0.63$ y $e_{10}(0.1) = 0.80$. Por lo que para capturar 1 % o 10 % de los datos para formar un promedio local, se necesita cubrir 63 % o 80 % del rango para cada variable de entrada. Claramente, estos vecinarios ya no son locales. [9]

En otras palabras, la maldición de la dimensión es un desafío estadístico. Cuando aumentan las dimensiones de los datos de entrada, el número de posibles configuraciones en las variables aumenta exponencialmente. Por lo tanto, el desafío estadístico aparece debido a que el número de posibles configuraciones es mayor que el número de ejemplos de entrenamiento. [6]

CAPÍTULO 3

CONCEPTOS DE APRENDIZAJE PROFUNDO

3.1. Introducción

La mayoría de los modelos de aprendizaje de máquina tienen una arquitectura simple de dos capas. Sin embargo, al analizar el cerebro humano, se puede ver que este tiene muchos niveles de procesamiento. Se cree que cada capa aprende características o representaciones con niveles crecientes de abstracción. Por ejemplo, el modelo estándar del cortex humano sugiere que el cerebro primero extrae bordes, después pequeñas áreas, después superficies, después objetos, etc. Esta observación ha inspirado a un área del aprendizaje de máquina conocida como aprendizaje profundo que intenta replicar este tipo de arquitectura en una computadora. [15]

El cerebro humano funciona de una manera diferente a una computadora normal. El cerebro es altamente complejo, no-lineal y funciona en paralelo. También tiene la capacidad de organizar sus componentes estructurales, conocidos como neuronas, para realizar tareas tales como reconocimiento de patrones a una velocidad muchas veces más rápida que la computadora más rápida que existe hoy en día. [11]

¿Cómo es capaz el cerebro humano de realizar dichas funciones? Desde el nacimiento, el cerebro ya tiene suficiente estructura y la habilidad para construir sus propias reglas de comportamiento a través de lo que conocemos como “experiencia”. El continuo desarrollo del sistema nervioso es posible gracias a la plasticidad del cerebro. La plasticidad permite al sistema nervioso de adaptarse al medio ambiente que lo rodea. Así como la plasticidad es esencial para el cerebro humano, también lo es con redes neuronales que están hechas de neuronas artificiales. [11]

Las redes neuronales artificiales se parecen al cerebro humano principalmente en dos aspectos:

- 1.- La red adquiere conocimiento de su ambiente a través de un proceso de aprendizaje.
- 2.- La fuerza de conexión entre las neuronas, conocido como los pesos sinápticos, se usa para almacenar el conocimiento adquirido.

El procedimiento que se usa para el proceso de aprendizaje se conoce como algoritmo de aprendizaje y su función es modificar los pesos sinápticos para mejorar el desempeño en la tarea que se quiere realizar. [11]

Actualmente, el aprendizaje profundo sigue en desarrollo. Sin embargo, este ya resuelve una gran cantidad de tareas y es usado en la industria de manera importante. La mayoría de las tareas que consisten en mapear un vector de entrada a un vector de salida, y que son fáciles para una persona de realizar pueden ser resueltas con aprendizaje profundo si se tienen bases de datos y modelos suficientemente grandes. Otras tareas que no se pueden describir como mapear un vector a otro o que son suficientemente difíciles para que una persona tuviera que reflexionar para resolverlas, se mantienen fuera del alcance del aprendizaje profundo por ahora. [6]

En el siguiente capítulo se describirá el modelo básico de redes neuronales. Después, se explicarán conceptos importantes de regularización y de optimización. Finalmente, se explicarán los conceptos de redes neuronales recurrentes que se utilizarán para los experimentos de los capítulos 4 y 5.

3.2. Redes neuronales prealimentadas

Las redes neuronales prealimentadas, también conocidas como perceptrones multicapa son el modelo básico del aprendizaje profundo. El objetivo de estas es aproximar una función f^* . Una red neuronal prealimentada define un mapeo $y = f(x; \theta)$ y aprende el valor de los parametros θ que resulten en la mejor función de aproximación.

Estos modelos se conocen como prealimentados porque la información fluye a través de la función al ser evaluada en x , después, a través de los cálculos intermedios utilizados para definir a f , y, finalmente, para obtener los datos de salida y . No hay conexiones cíclicas en las que un dato de salida del modelo sirva de nuevo como dato de entrada. Cuando esto sucede, el modelo se conoce como red neuronal recurrente. [6]

Las redes neuronales prealimentadas son redes porque se construyen componiendo varias funciones. Por ejemplo, se puede tener tres funciones $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ conectadas en una cadena para formar $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$.

La longitud de la red da la profundidad del modelo. Es esta terminología la que le da el

nombre de aprendizaje profundo. La última capa se conoce como capa de salida. Los ejemplos de entrenamiento especifican directamente lo que la capa de salida debe hacer en cada punto x , esto es, debe producir un valor que sea cercano a y . El comportamiento de las otras capas no está especificado por los datos de entrenamiento, por lo que estas capas se conocen como capas ocultas. El algoritmo de aprendizaje usado es el que tiene que decidir cómo usar estas capas para implementar una aproximación de f^* .

Finalmente, estas redes se conocen como neuronales porque están inspiradas en el cerebro. Cada capa es un vector y la dimensión de las capas ocultas determina el ancho del modelo. Cada elemento del vector puede pensarse que es una neurona, entonces, en vez de pensar a cada capa como una función de vector a vector, la podemos pensar como varias unidades que actúan en paralelo. Cada unidad se parece a una neurona en el sentido de que recibe datos de entrada de las otras unidades y calcula su valor de activación. La idea de usar muchas capas también proviene de la neurociencia. Sin embargo, la investigación actual en redes neuronales está guiada por muchos principios de matemáticas y de ingeniería y su objetivo no es modelar perfectamente al cerebro sino lograr generalización estadística. [6]

Para hacer la idea de una red neuronal más concreta, se dará un pequeño ejemplo con una capa oculta y dos unidades ocultas. Se quiere escoger la forma de nuestro modelo $f(x; \theta) = f^{(2)}(f^{(1)}(x))$. Las unidades ocultas h de esta red neuronal son calculadas por una función $f^{(1)}(x; W, c)$. Los valores de estas unidades ocultas después son utilizadas como datos de entrada para la segunda capa.

La segunda capa es la capa de salida del modelo y en este caso será una regresión lineal aplicada a h , es decir, $f^{(2)}(h; w, b) = w^T h + b$. Ahora bien, si $f^{(1)}$ fuera lineal, entonces f sería lineal y eso no es lo que se busca. Por lo tanto, $f^{(1)}$, la función que describe los rasgos, tendrá que ser no lineal. La mayoría de las redes neuronales utiliza una transformación afín seguido de una función no-lineal llamada función de activación. [6]

Si se hace esto, entonces $h = g(W^T x + c)$ donde W son los pesos de la transformación lineal y c son los sesgos. La función de activación g generalmente se aplica elemento a elemento, de tal forma que $h_i = g(x^T W_{:,i} + c_i)$. En la actualidad, la función de activación que se recomienda por default es la unidad linear rectificada o ReLU por sus siglas en inglés (Rectified Linear Unit), definida por la función de activación $g(z) = \max\{0, z\}$. [16] [6]

Por lo tanto, el modelo definido es el siguiente:

$$f(x; W, c, w, b) = w^T \max\{0, W^T x + c\} + b \quad (3.2.1)$$

3.2.1. Aprendizaje con descenso de gradiente

Diseñar y entrenar una red neuronal no es muy diferente que entrenar cualquier otro modelo de aprendizaje de máquina con descenso de gradiente. La diferencia más grande entre los modelos lineales y las redes neuronales es que la no linealidad de las redes neuronales hace que muchas veces la función de pérdida se vuelva no convexa. Esto quiere decir que las redes neuronales generalmente se entrenan utilizando métodos de optimización iterativos basados en el descenso del gradiente. [6]

3.2.1.1. Funciones de pérdida

Un aspecto importante del diseño de una red neuronal es la elección de la función de pérdida. Afortunadamente, las funciones de costo para redes neuronales son muy parecidas a las de otros modelos paramétricos, como los modelos lineales.

En la mayoría de los casos, el modelo paramétrico define una distribución $p(y|x; \theta)$ y se usa el principio de máxima verosimilitud. Esto quiere decir que la función de costo se calcula como la entropía cruzada entre los datos de entrenamiento y las predicciones del modelo. La función total de costo usada para entrenar una red neuronal generalmente combinará una de las funciones de pérdida vistas aquí con un término de regularización. [6]

3.2.1.2. Unidades de salida

La elección de la función de costo está altamente relacionada con la elección de la unidad de salida. La mayoría de las veces, simplemente se usa la entropía cruzada entre la distribución de los datos y la distribución del modelo. Por lo tanto, la elección de las unidades de salida determina la forma que tendrá la función de la entropía cruzada. [6]

Una de las unidades de salida más utilizada es la función softmax. Esta usa cuando se quiere representar una distribución de probabilidades sobre una variable discreta con n clases diferentes. Por lo tanto, se necesita producir un vector \hat{y} , con $\hat{y}_i = P(y = i|x)$. Se requiere no solo que cada elemento de \hat{y} esté entre 0 y 1, pero también que el vector sume a 1 para que represente una distribución de probabilidad válida.

Para conseguir eso, primero una capa lineal predice las log probabilidades no normalizadas.

$$z = W^T h + b \tag{3.2.2}$$

Después, la función softmax puede exponenciar y normalizar z para obtener el deseado \hat{y} . Por lo tanto, la función softmax está dada por:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \quad (3.2.3)$$

3.2.1.2.1 Redes de densidad mixta

Las redes de densidad mixta son redes neuronales que utilizan otro tipo de unidad de salida: la mezcla de gaussianas. Estos modelos son muy efectivos para modelar movimientos de objetos físicos como se hará en el capítulo 5 al describir movimientos de caligrafía. En la ecuación (3.2.4), se define una mezcla de gaussianas con n componentes como una distribución de probabilidades condicionales. [6]

$$p(y|x) = \sum_{i=1}^n p(c=i|x) \mathcal{N}(y; \mu^{(i)}(x), \Sigma^{(i)}(x)) \quad (3.2.4)$$

La red neuronal debe entregar tres datos de salida: un vector que defina a $p(c=i|x)$, una matriz que entregue $\mu^{(i)}(x)$ para toda i , y un tensor que entregue $\Sigma^{(i)}(x)$. Por un lado, el vector $p(c=i|x)$, donde c es una variable latente, generalmente se obtiene de aplicar una función softmax a un vector de n componentes. Por otro lado, las medias $\mu^{(i)}(x)$ indican el centro de la i -ésima componente gaussiana, por lo que si y es un vector de d dimensiones, la red nos debe de entregar una matriz de $n \times d$ que contenga a los n vectores de dimensión d . Finalmente, $\Sigma^{(i)}(x)$ calculará la matriz de covarianza para cada componente i . [6]

3.2.2. Unidades ocultas

Las unidades ocultas es un tema exclusivo de las redes neuronales, puesto que estas no aparecen en los algoritmos convencionales de aprendizaje de máquina. El diseño de las unidades ocultas es un tema de investigación muy activo y no existen todavía principios teóricos definitivos que guíen el tema. [6]

Existen varias unidades ocultas, incluyendo la ReLU, que no son diferenciables en todos sus puntos. Se podría pensar que esto invalida a las funciones para ser utilizadas con algoritmos de descenso de gradiente. Pero en la práctica, descenso de gradiente funciona suficientemente bien. Esto es en parte porque los algoritmos con redes neuronales generalmente no llegan a un mínimo local de la función objetivo, sino que simplemente reducen su valor significativamente. [6]

3.2.3. Diseño de la arquitectura de la red

Otra consideración importante para el diseño de la red es su arquitectura, es decir, su estructura en términos generales: cuántas unidades tendrá y cómo se conectan estas unidades

entre ellas. La mayoría de las redes neuronales están organizadas en capas y estas capas se conectan en cadena, donde cada capa es una función de la capa que le precede. Con esta estructura, la primera capa está dada por: [6]

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b^{(1)}), \quad (3.2.5)$$

la segunda capa por:

$$h^{(2)} = g^{(2)}(W^{(2)T}h^{(1)} + b^{(2)}), \quad (3.2.6)$$

y así sucesivamente donde x son los datos de entrada, W las matrices de pesos, b el vector de sesgo y g las funciones de activación. Con este tipo de arquitectura, las principales consideraciones que se tienen que hacer son la profundidad y el ancho del modelo. La arquitectura ideal para cierta tarea debe ser encontrada mediante la experimentación y monitoreando el error de validación. [6]

3.2.4. Propagación hacia atrás

En las redes neuronales prealimentadas, cuando le damos al modelo datos de entrada x para producir un dato de salida \hat{y} , la información fluye hacia adelante a través de la red. Los datos de entrada x proveen la información necesaria que luego se propaga en las unidades ocultas de cada capa para finalmente producir \hat{y} . Esto es llamado propagación hacia adelante. Durante el entrenamiento, la propagación hacia adelante continúa hasta producir un costo escalar $J(\theta)$. El algoritmo de propagación hacia atrás permite a la información de fluir hacia atrás a través de la red para calcular el gradiente. [6] [18]

A menudo se piensa que el algoritmo de propagación hacia atrás es todo el algoritmo de aprendizaje para redes neuronales. Esto es falso, el algoritmo de propagación hacia atrás solo se refiere al método para calcular el gradiente, mientras que otro algoritmo, descenso de gradiente estocástico, es usado para aprender utilizando este gradiente. [6]

3.2.5. Propagación hacia atrás con gráficas computacionales

Para describir con más formalidad el algoritmo de propagación hacia atrás, se utilizará un lenguaje de gráficas computacionales. Cada nodo de la gráfica indicará una variable. También se definirá una operación como una función de una o más variables cuyo valor de salida es otra variable. Si la variable y se calcula aplicando una operación a la variable x , entonces se trazará una arista dirigida de x a y .

Ahora bien, al realizar el algoritmo de propagación hacia atrás, se utilizará la gráfica computacional para decidir el orden de las operaciones y aplicar la regla de la cadena para

calcular el gradiente. La regla de la cadena generalizada dice lo siguiente: sea $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, g es una función de \mathbb{R}^m a \mathbb{R}^n , f es una función de \mathbb{R}^n a \mathbb{R} . Si $y = g(x)$, $z = f(y)$, entonces:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \quad (3.2.7)$$

En notación vectorial, esto es equivalente a:

$$\nabla_x z = \left(\frac{\partial y}{\partial x} \right)^T \nabla_y z, \quad (3.2.8)$$

donde $\frac{\partial y}{\partial x}$ es la matriz jacobiana de g con dimensiones de $n \times m$. De aquí se puede ver que el gradiente de una variable x se puede obtener al multiplicar la matriz jacobiana $\frac{\partial y}{\partial x}$ por el gradiente $\nabla_y z$. El algoritmo de propagación hacia atrás consiste en realizar este producto para cada operación en la gráfica. [6]

3.3. Regularización en aprendizaje profundo

En la subsección 2.3.2 se dio una breve introducción a lo que es regularización. Regularización se refiere a cualquier modificación que se hace a un algoritmo de aprendizaje para reducir su error de generalización pero no su error de entrenamiento. En esta sección, se analizarán algunos de los métodos de regularización que funcionan mejor en aprendizaje profundo. [6]

Vincent Vanhoucke, investigador de Google, utiliza una metáfora para dar una mejor idea de cómo funciona regularización en su curso de aprendizaje profundo. Vanhoucke explica que hay un problema general al hacer optimización numérica que él conoce como el problema de los pantalones “skinny”. Este tipo de pantalones se ven muy bien y se ajustan perfectamente, pero es muy difícil de entrar en ellos, por lo que la mayoría de las personas acaban usando pantalones que les quedan un poco grandes. Lo mismo pasa con las redes profundas: una red que tenga el tamaño justo para los datos es muy difícil de optimizar, por lo que en la práctica se utilizan redes que son muy grandes para los datos y después se intenta que no sobreajusten. A continuación, se describirá el método de detención temprana y el método de “dropout”. [24]

3.3.1. Detención temprana

La técnica de detención temprana se basa en una idea muy simple: utilizar los parámetros que minimizan el error de validación. En general, al utilizar modelos grandes que puedan sobrestimar a los datos, sucede que el error de entrenamiento disminuye a través del tiempo,

pero el error de validación comienza a aumentar. Por lo tanto, se obtiene un mejor error de validación y probablemente de prueba al utilizar los parámetros que entregaron el menor error de validación. [6]

Por lo tanto, lo que se hace en la práctica es guardar los parámetros del modelo cada vez que el error de validación mejore. Al finalizar el algoritmo, se entregarán los parámetros guardados en vez de los últimos parámetros. El algoritmo terminará si los parámetros no han mejorado al error de validación en un número preespecificado de iteraciones. Este es el tipo más común de regularización en aprendizaje profundo por su efectividad y simplicidad. [6]

3.3.2. Dropout

Dropout es otro método de regularización que emergió recientemente (Srivastana et al., 2014) y funciona muy bien debido a su bajo costo computacional. Basicamente, dropout funciona de la siguiente manera. Si se tienen dos capas que se conectan entre sí, los valores que van de una capa a otra se conocen como activaciones. Lo que hace dropout es que cada una de estas activaciones se hace 0 con una probabilidad p , donde generalmente $p = 0.5$ para las capas ocultas. En otras palabras, lo que está haciendo dropout es “destruir” la mitad de los datos. [6] [20] [24]

Esta idea puede sonar un poco rara, pero Vincent Vanhoucke explica la intuición detrás del éxito del método en su curso de aprendizaje profundo: la red no puede confiar en ninguna activación porque podría ser convertida en cero en cualquier momento, por lo que la red aprende una representación redundante para estar segura que al menos alguna parte de la información permanezca. En la práctica, esto hace a la red más robusta y previene sobreajustar a los datos. [6] [24]

3.4. Optimización en aprendizaje profundo

La optimización es uno de los aspectos más importantes a considerar al hablar de aprendizaje profundo. La optimización de las redes neuronales, es decir, encontrar los parámetros θ que minimizan la función de pérdida $J(\theta)$, es un problema bastante complicado. Es por esto que se han creado técnicas especiales para resolver este desafío. En la sección 2.5, se explicó el algoritmo básico de aprendizaje de máquina, gradiente estocástico, por lo que esta sección se dedicará a describir algunos de los algoritmos más avanzados que se usarán en los capítulos 4 y 5.

El aprendizaje de máquina tiene diferencias considerables con la optimización pura. En el aprendizaje de máquina se aprende indirectamente, es decir, normalmente se busca mejorar una medida de desempeño P , pero, por la dificultad de medir a P , se utiliza una función de

pérdida diferente $J(\theta)$ con la esperanza de mejorar P . En cambio, en la optimización pura, minimizar a $J(\theta)$ es un objetivo en sí mismo. [6]

3.4.1. Momentum

El método del gradiente estocástico que se estudió en la sección 2.5 es uno de los algoritmos claves de optimización en redes neuronales. Sin embargo, este método puede ser un poco lento. El método de momentum está diseñado para acelerar el aprendizaje al acumular un promedio de gradientes con decaimiento exponencial. Este método introduce una variable v que tiene el rol de la velocidad debido a que es la dirección y velocidad a la que los parámetros se mueven por el espacio de parámetros. Un hiperparámetro α determina qué tan rápido las contribuciones de gradientes previos decaen exponencialmente. La regla de actualización es la siguiente: [6]

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left(\frac{1}{m} \sum_1^m L(x^{(i)}, y^{(i)}, \theta) \right) \quad (3.4.1)$$

$$\theta \leftarrow \theta + v \quad (3.4.2)$$

3.4.2. Algoritmos con tasas de aprendizaje adaptativas

Elegir la tasa de aprendizaje es muy importante al optimizar redes neuronales debido a que esta elección tiene un impacto significativo en el desempeño del modelo. El algoritmo de momentum que se vio en la subsección anterior puede mitigar estos problemas hasta cierto punto, pero lo hace a expensas de tener que introducir un nuevo hiperparámetro. Para resolver este problema se han creado algoritmos que utilizan una tasa de aprendizaje diferente para cada parámetro y automáticamente adaptan estas tasas a lo largo del entrenamiento. Algunos los algoritmos con tasas de aprendizaje adaptativas más conocidos son los siguientes: AdaGrad, RMSProp y Adam. [6]

3.5. Redes neuronales recurrentes

Las redes neuronales recurrentes son una familia de redes neuronales utilizadas para procesar datos secuenciales. Estas redes permiten procesar sucesiones mucho más grandes de las que podría procesar una red neuronal convencional, además de poder procesar sucesiones con longitud variable. [6] [18]

Una de las claves que llevó a desarrollar las redes neuronales recurrentes es la idea de compartir parámetros a través de diferentes partes del modelo. Compartir parámetros hace

posible extender y aplicar el modelo a sucesiones de diferentes longitudes y generalizar a través de ellas. Compartir parámetros es especialmente importante cuando un pedazo de información puede ocurrir en diferentes posiciones dentro de la sucesión. Por ejemplo, si se consideran los dos enunciados “En 2011, entré al ITAM” y “Entré al ITAM en 2011”. Si se le pide a un modelo de aprendizaje de máquina leer los dos enunciados y extraer el año en el que el narrador entró al ITAM, debería reconocer 2011 como un pedazo clave de información sin importar si apareció en la quinta o en la tercera posición del enunciado. Una red neuronal convencional tendría parámetros separados para cada variable de entrada, por lo que tendría que aprender todas las reglas del language de manera separada para cada posición en el enunciado. En cambio, una red neuronal recurrente comparte los pesos a través de las diferentes ventanas de tiempo. [6]

3.5.1. Desdoblando gráficas computacionales

Como se vio en la sección 3.2.5, las gráficas computacionales permiten formalizar la estructura de una red neuronal. En esta sección, se explicará cómo desdoblar una red neuronal recurrente hacia una gráfica computacional que tenga estructura repetitiva. Al desdoblar la gráfica, se obtendrá una gráfica que comparte parámetros a través de una red profunda. [6]

Por ejemplo, si se considera la forma clásica de un sistema dinámico:

$$s^{(t)} = f(s^{(t-1)}; \theta), \quad (3.5.1)$$

donde $s^{(t)}$ es el estado del sistema. La ecuación (3.3.1) se considera recurrente debido a que la definición de s en el tiempo t alude a la misma definición pero en el tiempo $t - 1$. Para un número finito de pasos τ , la gráfica puede ser desdoblada aplicando la definición $\tau - 1$ veces. Por ejemplo, si se desdobla la ecuación que se definió $\tau = 3$ pasos de tiempo, se obtiene:

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta) \quad (3.5.2)$$

En otras palabras, si se desdobla la ecuación al aplicar la definición, se obtiene una expresión que no involucra recurrencia. Esta expresión se puede representar por una gráfica tradicional sin ciclos como se muestra en la figura 3.1. [6]

Otro ejemplo sería considerar un sistema dinámico afectado por una señal externa $x^{(t)}$,

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \quad (3.5.3)$$

Muchas redes neuronales recurrentes utilizan la ecuación 3.5.3 para definir su estado oculto o “hidden state” en inglés; por eso, se utilizará la letra h para describir la sucesión. Una red neuronal recurrente normalmente tiene también una capa de salida que lee información del estado h para hacer predicciones. [6]

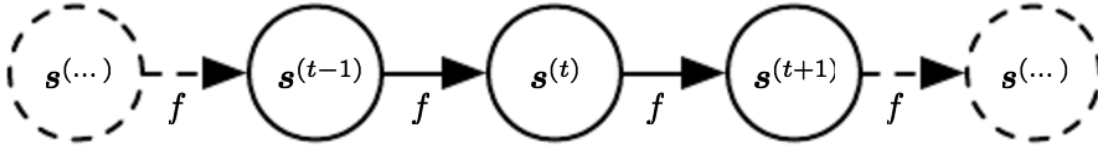


Figura 3.1: Gráfica desdoblada de la ecuación 3.5.1. Cada nodo representa el estado en un tiempo t y la función f mapea el estado t al estado $t+1$. Los mismos parámetros θ se utilizan para todos los pasos de tiempo. Figura tomada de (Goodfellow et al, 2016).

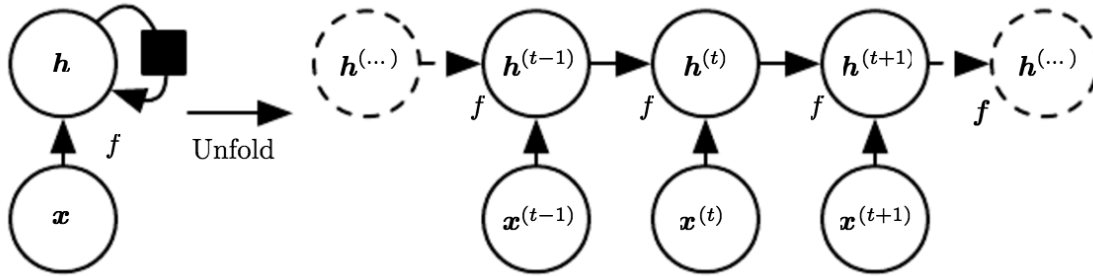


Figura 3.2: Red neuronal recurrente sin capa de salida. Esta red incorpora información de los datos de entrada x al estado h a través del tiempo. El cuadrado negro en las gráficas indica el retraso en un paso de tiempo. Figura tomada de (Goodfellow et al, 2016).

3.5.2. Redes neuronales recurrentes

Con los conceptos hasta ahora desarrollados, ya se pueden construir una variedad de redes neuronales. Se ejemplificará esto con el modelo básico que usará en los capítulos 4 y 5 para desarrollar los experimentos. Este modelo, mostrado en la figura 3.3, es una red neuronal que produce un dato de salida en cada paso de tiempo y tiene conexiones recurrentes entre las capas ocultas. [6] [8]

A continuación, se describirán las ecuaciones de propagación hacia adelante de este modelo. Se supondrá que los datos de salida son discretos, como se hará en el experimento 4, y que la función de activación es la función ReLU. Las siguientes son las ecuaciones para los pasos de tiempo $t = 1$ a $t = \tau$.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \quad (3.5.4)$$

$$h^{(t)} = ReLU(a^{(t)}) \quad (3.5.5)$$

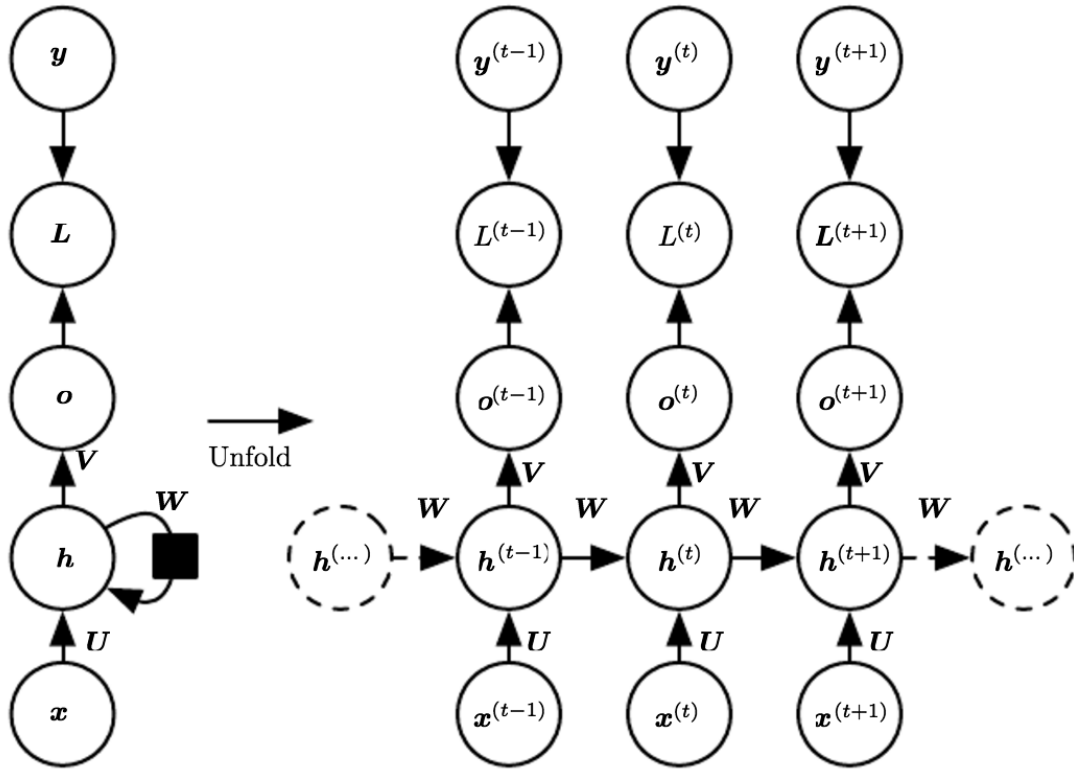


Figura 3.3: Una red neuronal recurrente enrollada y desenrollada. Con la sucesión de datos de entrada x , se calculan los datos de salida o y con estos y las etiquetas y se puede computar la función de pérdida correspondiente L . También se puede ver que U es la matriz de pesos de los datos de entrada a las capas ocultas, W es la matriz de pesos que conecta la recurrencia de las unidades ocultas y V es la matriz de pesos de las unidades ocultas a los datos de salida o . Figura tomada de (Goodfellow et al, 2016).

$$o^{(t)} = c + Vh^{(t)} \quad (3.5.6)$$

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)}) \quad (3.5.7)$$

Si la función de pérdida, $L^{(t)}$, es log-verosimilitud negativa de $y^{(t)}$ dado $x^{(1)}, \dots, x^{(t)}$, entonces

$$L(\{x^{(1)}, \dots, x^{(\tau)}\}, \{y^{(1)}, \dots, y^{(\tau)}\}) \quad (3.5.8)$$

$$= \sum_t L^{(t)} \quad (3.5.9)$$

$$= - \sum_t \log p_{model}(y^{(t)} | \{x^{(1)}, \dots, x^{(t)}\}) \quad (3.5.10)$$

donde $p_{model}(y^{(t)} | \{x^{(1)}, \dots, x^{(t)}\})$ es la entrada de $\hat{y}^{(t)}$ que corresponde a la etiqueta $y^{(t)}$. Para calcular el gradiente se hace la propagación hacia adelante al moverse en la figura (3.3) de izquierda a derecha, y luego la propagación hacia atrás se hace de derecha a izquierda de esta gráfica computacional extendida. [6]

3.5.3. Dependencias de largo plazo

El problema de las dependencias de largo plazo aparece cuando la gráfica computacional es demasiado larga, lo cual sucede fácilmente al desdoblar la gráfica en las redes neuronales recurrentes. En este caso, como los mismos parámetros se propagan hacia atrás por muchos pasos, el algoritmo de propagación hacia atrás se vuelve inestable y se puede llegar al problema del gradiente evanescente o al problema del gradiente explosivo. Se explicará a continuación el porqué. [6] [11]

Supongamos que una gráfica computacional contiene un camino que consiste en multiplicar repetidamente por una matriz W . Después de t pasos, esto es equivalente a multiplicar por W^t y si la descomposición en valores propios de W es $W^t = V \text{diag}(\lambda) V^{-1}$, entonces:

$$W^t = (V \text{diag}(\lambda) V^{-1})^t = V \text{diag}(\lambda)^t V^{-1} \quad (3.5.11)$$

Por lo que los valores propios, λ_i cuyo valor absoluto no sea cercano a 1, se harán muy grandes si son mayores que uno, causando el problema del gradiente explosivo, o se reducirán rápidamente a cero, causando el problema del gradiente evanescente. [6] [11]

Se explicará a continuación el método para resolver el problema del gradiente explosivo: recorte de gradiente. La solución para el problema del gradiente evanescente, la memoria corta de largo plazo, se explicará en la siguiente sección.

3.5.4. Recorte de gradiente

Una técnica para evitar el problema del gradiente explosivo es el recorte de gradiente. Este es un problema relevante, pues si el gradiente de un parámetro es demasiado grande, al hacer la actualización, se puede acabar muy lejos del punto original y en una región donde la función objetivo tenga un valor mayor. En la figura 3.4, tomada de (Goodfellow et al., 2016), se da un ejemplo de las complicaciones que pueden aparecer. [6] [17]

La solución a esto es recortar el gradiente. Es decir, elegir una cota máxima para las entradas del gradiente y, justo antes de hacer la actualización de parámetros, escalar al gradiente para que se respete la cota máxima. [6] [8] [17]

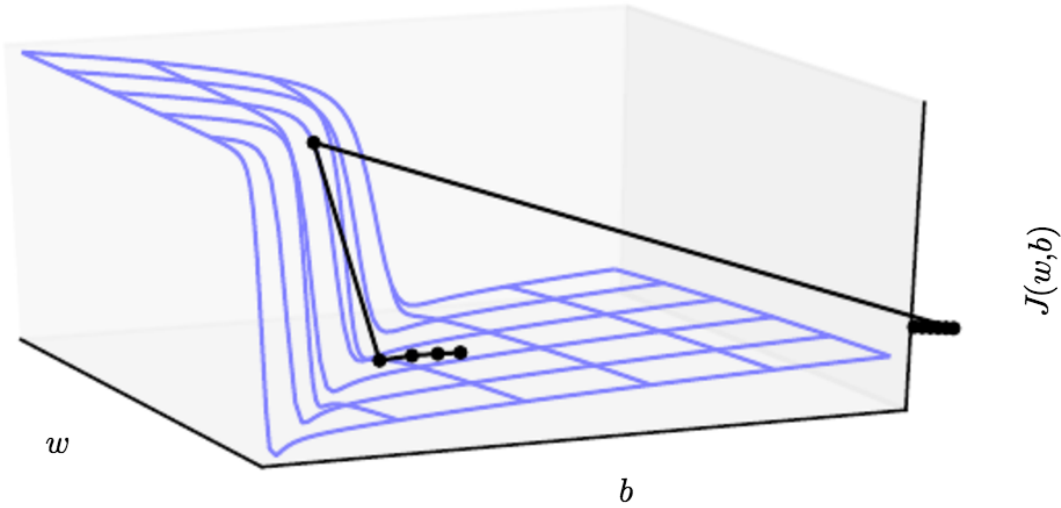


Figura 3.4: La función de pérdida $J(w, b)$ puede contener puntos que tengan derivadas muy grandes. Si los parámetros están cerca de estos puntos, al avanzar con descenso de gradiente, se llegará a un punto muy lejano, lo cual podría afectar a la optimización.

3.6. Memoria larga de corto plazo

La memoria larga de corto plazo es un tipo de unidad recurrente con “compuertas”. Estas compuertas se construyen de tal forma que los caminos de la gráfica computacional tengan derivadas que no exploten ni se desvanezcan. De esta forma se puede acumular información por un largo periodo de tiempo, y si la información ha sido usada y el sistema no la necesita más, también la puede borrar de la memoria. En la figura 3.5, tomada de (Goodfellow et al., 2016), se muestra la estructura de una “celda” de memoria larga de corto plazo. Nos referiremos a estas como LSTM de ahora en adelante, por su nombre en inglés “long short-term memory”. [5] [6] [12]

A continuación se presentarán las ecuaciones de propagación hacia adelante para la red neuronal recurrente LSTM con una capa oculta. El componente principal de una celda LSTM es el estado de la celda $s_i^{(t)}$, el cual tiene una conexión recurrente a sí mismo que es controlado por la compuerta de olvido $f_i^{(t)}$ cuyo valor se modula entre 0 y 1 por una unidad sigmoïdal. Por lo tanto si $x^{(t)}$ es el vector de datos de entrada actual, $h^{(t)}$ es el vector de la capa oculta actual que contiene los datos de salida de las celdas LSTM, y b^f , U^f y W^f son los sesgos, pesos de entrada y pesos recurrentes para la compuerta de olvido respectivamente, entonces: [6] [8] [19]

$$f_i^{(t)} = \sigma \left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)} \right) \quad (3.6.1)$$

Ahora bien, la compuerta de entrada exterior $g_i^{(t)}$ se calcula de manera similar a la compuerta de olvido pero con sus propios parámetros:

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right) \quad (3.6.2)$$

Por lo tanto si b , U y W son los sesgos, pesos de entrada y pesos recurrentes de la celda LSTM, entonces el estado $s_i^{(t)}$ se actualiza de la siguiente manera:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \tanh \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right) \quad (3.6.3)$$

Para calcular el vector de salida de la celda LSTM, $h_i^{(t)}$, se utiliza una compuerta de salida $q_i^{(t)}$ de manera similar a como se hizo con el estado de la celda: [6] [8] [19]

$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right) \quad (3.6.4)$$

$$h_i^{(t)} = \tanh \left(s_i^{(t)} \right) q_i^{(t)} \quad (3.6.5)$$

Se ha mostrado que las redes neuronales recurrentes que utilizan celdas LSTM pueden aprender y modelar sucesiones complejas con estructura de largo alcance. A continuación, en los capítulos 4 y 5, se modelarán este tipo de sucesiones utilizando estos modelos. [6] [8] [19] [22]

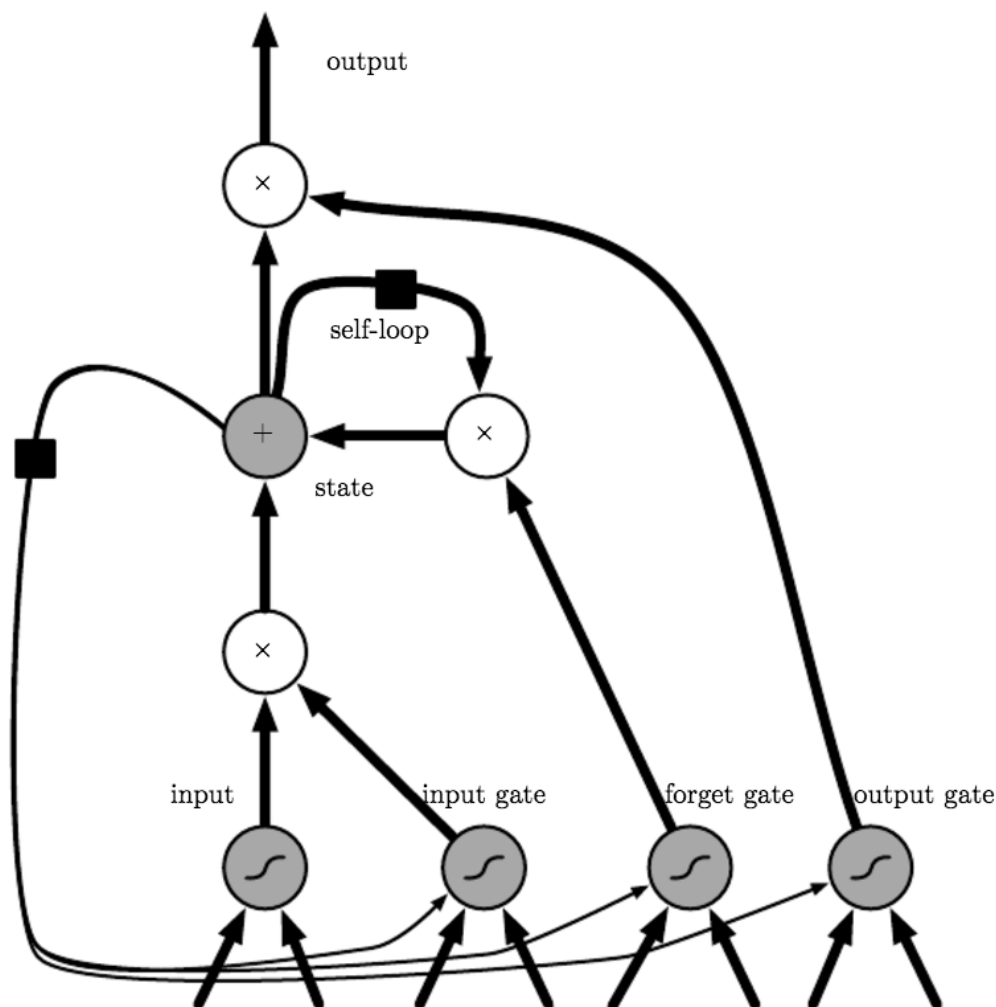


Figura 3.5: Diagrama de la celda LSTM de la red neuronal. Estas celdas están conectadas recurrentemente entre ellas, remplazando las unidades ocultas de las redes recurrentes tradicionales. Los datos de entrada pasan por una red neuronal tradicional. Después este valor puede ser acumulado en el estado de la celda si la compuerta de entrada sigmoidal lo permite. La unidad que contiene al estado de la celda tiene un estado recurrente a sí misma, cuyo peso es controlado por la compuerta de olvido. Las datos de salida de la celda pueden ser “apagados” por la compuerta de salida. Todas las unidades de compuerta tienen una función de activación no lineal y usualmente sigmoidal. Figura tomada de (Goodfellow et al. 2016).

CAPÍTULO 4

EXPERIMENTO CON TEXTO

4.1. Introducción

En este capítulo, se entrenarán modelos de texto utilizando redes neuronales recurrentes con LSTM. El objetivo será generar sucesiones de texto que parezcan reales. Para hacer esto, se procesarán y entrenarán los modelos con diferentes bases de texto y después se generarán las sucesiones de la siguiente manera: se sampleará la distribución del modelo y después la predicción se utilizará como dato de entrada para el siguiente paso. En otras palabras, como explica (Graves, 2013), el modelo trata a sus invenciones como si fueran reales, de la misma manera que lo hace una persona cuando sueña. Para decidir los hiperparámetros de los modelos se utilizó como base el artículo mencionado de Alex Graves, así como el artículo “The Unreasonable Effectiveness of Recurrent Neural Networks” escrito por Andrej Karpathy, estudiante de doctorado en aprendizaje profundo en Stanford. [8] [13]

Se utilizarán 3 bases de texto diferentes y se compararán los resultados obtenidos. El primer texto que se utilizará será el libro “Rayuela” de Julio Cortazar, el segundo, será “La Biblia” en inglés y, por último, se utilizará la base de datos del concurso Hutter.

Aunque generalmente los modelos de lenguaje se hacen a nivel palabra, en este trabajo se harán a nivel caracter. Esto se debe a varias razones. Por un lado, si el análisis se hace a nivel palabra, como el número de palabras es muy grande (a veces mayor a 100,000 incluyendo diferentes conjugaciones, nombres propios, etc.), el número de clases y parámetros es demasiado grande y eso aumenta el costo computacional. Por otro lado, predecir un caracter a la vez es más interesante desde el punto de vista de generación de sucesiones porque permite al modelo crear nuevas palabras. Por lo tanto, siguiendo el trabajo de (Graves, 2013), se predecirá con la granularidad más fina, nivel caracter, para maximizar la flexibilidad del modelo. [8]

4.2. Descripción de las bases de datos

A continuación, se describirá cada base de datos y los modelos que se utilizaron para cada análisis. El código del modelo utilizado se pueden encontrar en el siguiente repositorio: <https://github.com/GeorBelanger>

4.2.1. Rayuela

Este libro de Julio Cortazar escrito en París y publicado en 1963 es una de las obras centrales del boom latinoamericano. En ella, Cortazar invita al lector a leer sus 155 capitulos en el orden que desee, por lo que se le suele llamar una “antinovela” y esto hace que tenga una estructura interesante para generar nuevas sucesiones de ella. Esta base de datos se obtuvo de la página del Instituto Latinoamericano de la Comunicación Educativa (ILCE). Esta base tiene 972,231 caracteres y los conjuntos de validación y prueba tienen cada uno 100,000 caracteres. [2] [3]

4.2.2. La Biblia

La Biblia es uno de los libros más leídos y más famosos de la historia. Este texto está en inglés y se obtuvo de NLTK (“Natural Language Toolkit”), una plataforma dedicada a trabajar con datos de lenguaje humano. Esta base tiene 4,123,223 caracteres y los conjuntos de validación y prueba tienen cada uno 500,000 caracteres. [21]

4.2.3. Concurso Hutter

Finalmente, siguiendo el trabajo de (Graves, 2013), se consideró el conjunto de datos del concurso “Hutter”. Esta base de datos contiene los primeros 100 millones de bytes de la página de Wikipedia en inglés. El archivo no solo incluye los datos comprimidos sino también el código para implementar el algoritmo de compresión. Además de contener muchas palabras del diccionario, la base de datos contiene muchas sucesiones de caracteres que normalmente no se usarían como por ejemplo etiquetas XML para definir los meta-datos, direcciones de páginas de internet y markup para formatear los datos (encabezados, listas, etc.). Esta base de datos contiene 82,551,973 caracteres y los conjuntos de validación y prueba tienen cada uno 1,000,000 de caracteres. [8]

4.3. Resultados

A continuación se mostrarán los resultados para cada uno de los modelos.

4.3.1. Modelo Rayuela

Para esta base de datos se utilizó una red neuronal recurrente con unidades LSTM. Este modelo empleó una tasa de aprendizaje inicial de 0.03, a Adam como algoritmo de optimización, un recorte de gradiente con cota de 1, el tamaño de las batches fue de 100 y las sucesiones que se analizaron tuvieron una longitud de 50 caracteres. En todos los modelos se utilizó detención temprana.

En primer lugar se corrió una red neuronal con 1 capa oculta con 100 unidades LSTM. Después de cada época, es decir, cada corrida de entrenamiento por todos los datos, se evaluó el error de validación y se generaron sucesiones de texto. En el siguiente cuadro, se puede ver una sucesión generada después de la primera época.

corirdos endo sabisano Olivial Traveciel toster el que escialtas cucl pmen querlan y hara dera que puasonuaco etion comures convivier de Maro el mesa Oseves sestretedaran Mibutra entobso, Cocaba quirdammotercimio Maburado la a la que cicos, y en me nastolajor de laua trecos tonterderos mo boco la de preintes la de mabar Al esteros al tinimedo, omrimherica..rud la tinopo Bera los untatra en su la odas tuca aque pronterio ya palolre cuenes munonot. Manos anavoste pan, del de flo el no demanbia.

En la sucesión de texto, se puede ver que el modelo ya ha aprendido algunas palabras básicas del español: el, que, mesa, pan, etc. Aunque todavía está un poco lejos el modelo generar sucesiones en español. A continuación se muestra una sucesión generada después de 20 épocas, cuando el modelo ya ha convergido.

tener Tido, coma, de lo voviid de que todo al meo. Yutropir la agumo, de todo al contundada. Soimente taleca caguas, aliver, armanlo atacasiora gente de piar el lanicaba esos caticado a fractores como Subera de tantos parto a creer. Nerliner as aquedalso temparrondo dos amobano esfunta a viejo. Oliveira de treinta a y es mi hiciente, protafe en el napecio. Meros gusiado como si me retorama, con timiizas. Puadero tan lado y por perregitas del mitio. Travoro es fuentes nada la contra en la co

En esta sucesión, el modelo ya genera palabras más complejas en español y se puede ver que incluso ha aprendido el nombre del personaje principal de la obra: Oliveira. No obstante, el modelo sigue lejos del desempeño esperado. El error de prueba fue menor que el error de validación, por lo que no se está sobreajustando. Se intentará utilizar modelos con más parámetros para mejorar el desempeño.

Para mejorar la calidad de las sucesiones, se aumentó la capacidad del modelo y luego se utilizó regularización para evitar sobreajustar. Se optó por un modelo de dos capas ocultas y

500 unidades ocultas y se utilizó dropout después de cada capa oculta. El error de validación con detención temprana resultó ser 1.64 y el error de prueba 1.60, por lo que parece ser que todavía no se está sobreajustando. A continuación se muestra una sucesión generada con este modelo.

A Pola como un estecho una salia esos fodido de cosas un dido... A nadie que a los caleungo de Oliveira. A oble Mont reputados que mi cientos dijo Tenrico de que vez. Mane, un estobadon, la busto estus fudente. Hecidos ejemposo dijo Talita y ventan En sido nos parata -laguo opirosa. Un que rearder. Remuodo esperana como Mato de tilonis, que fuera realmente Arren con gris, y asmiento el piersko quina si Se la mas hucha hasta los acorrino un bubio. Es salibles en pasa

Se puede ver que este modelo genera muchas palabras en español o con estructura del español y las frases hacen más sentido que en el modelo anterior. Una de las razones que pueden estar frenando un mejor desempeño es lo pequeño de la base de datos. En el siguiente experimento se utilizará una base de datos 4 veces más grande y se compararán los resultados.

4.3.2. Modelo Biblia

Para esta base de datos se utilizó el mismo modelo que con los datos de Rayuela. Es decir, una red neuronal recurrente con unidades LSTM, una tasa de aprendizaje inicial de 0.03, a Adam como algoritmo de optimización, un recorte de gradiente con cota de 1, el tamaño de las batches fue de 100 y las sucesiones que se analizaron tuvieron una longitud de 50 caracteres. En todos los modelos se utilizó detención temprana y dropout como regularización.

Esta base de datos es bastante más grande que la de Rayuela, por lo que el tiempo de entrenamiento aumenta también. Como primer modelo se utilizará 1 capa oculta con 100 unidades LSTM y a dropout como algoritmo de regularización después de cada capa oculta. Este modelo converge rápidamente y entrega mejores resultados que los experimentos con Rayuela. A continuación se verá una sucesión que se generó con este modelo.

Sheb, of kings your childres. The midst that and thou fall they say a rain stangues, with thou and seadwe, that that he for the leses, I mave knet and is somongel; unto him, besought this shoot. And the inholire she blood savids ye also tendenshaln my fompest; and the tritiel merle who einssyres as all usterly not which all man out over, even the word givity at he be it awonk whom even out whous him

Este modelo ya produce casi todas las palabras en inglés o a veces palabras que no existen, pero que tienen una estructura similar a la del idioma inglés. A continuación, se utilizó un modelo con una mayor capacidad: 1 capa oculta con 500 unidades LSTM.

El modelo con 1 capa oculta y 500 unidades LSTM convergió en 6 épocas y entregó muy

buenos resultados. El error de validación fue de 1.17 y el de prueba de 1.21, y a continuación se puede ver una sucesión que se generó con este modelo:

But Jeah said ut into the LORD God and these day of thine of that dwelt my God because of the priests swalling is also not gate of thiel, have spirit if us weed, but with them put her's lasting, and their judgest covered their tarrim to them, and please God which connembrory. But the first before him, and in my servant and sinate;

Este modelo ya escribe inglés bastante bien y algunas partes de las frases empiezan a tener sentido. La tabla 4.1 contiene los tiempos de entrenamiento y los errores de validación para cada época. Se puede ver que el valor de validación desciende hasta la sexta época y después empieza a aumentar. Debido a que se está utilizando detención temprana, el modelo de que se tenía en ese momento es con el que se evalúa el error de prueba.

Final de la época 1	tiempo: 983.04s	error valid 1.62	ppl valid 5.07
Final de la época 2	tiempo: 1018.08s	error valid 1.39	ppl valid 4.02
Final de la época 3	tiempo: 958.67s	error valid 1.32	ppl valid 3.76
Final de la época 4	tiempo: 972.85s	error valid 1.26	ppl valid 3.52
Final de la época 5	tiempo: 984.16s	error valid 1.19	ppl valid 3.28
Final de la época 6	tiempo: 1084.37s	error valid 1.17	ppl valid 3.23
Final de la época 7	tiempo: 984.27s	error valid 1.17	ppl valid 3.24
Final de la época 8	tiempo: 1065.81s	error valid 1.18	ppl valid 3.25
Final de la época 9	tiempo: 2404.71s	error valid 1.23	ppl valid 3.41
Final de la época 10	tiempo: 4644.00s	error valid 1.41	ppl valid 4.09
Final del entrenamiento		error de prueba 1.21	ppl prueba 3.34

Tabla 4.1: Resultados del entrenamiento del modelo

La tabla 4.2 contiene generaciones de sucesiones para las 6 primeras épocas, momento en el que el modelo alcanza su mejor desempeño. Se puede ver como las sucesiones van mejorando conforme el modelo disminuye su error de validación.

Se obtendrán más sucesiones de este modelo al hacer variar la temperatura de la softmax de salida. La temperatura es simplemente un número por el que se divide el vector de salida antes de aplicarle la función softmax. Por lo tanto, al disminuir la temperatura de 1 a 0.1, aumenta la confianza de las predicciones de la red, pero también será más conservadora. A continuación se muestra una muestra utilizando 0.1 como temperatura. [13]

qum noth, I wald ame is he wils. Befemod; whe vied that
the camem by in the entick not will the draad, fore priecy of were
ual to gindes ye depers will come cannted of his natering
whines I will s happerats. But God of his couse a scaight against
and ye there servens tought the land of a bread, when me
As come Behold, neither Ham, all not done offer on his dead wrire
e hear he lift firdness bour mine pon, and Shew as one
uncity, and Gad what unto commandmenteth in the intip Joht it to
e to Hali, saying, Gameth, are dwelling place the irmine
of him. For in truck in thy Surely, and all all Israel, We doth
rives seekand king's bedieffiseth, and I sike them being
even them fear in Ghist have my people had we shall see the mess

Tabla 4.2: Generacion de sucesiones para cada época

uhus thine sin to the servone of the son of King is shall the morning and the same
of the son of King and the son of Assyria the son of King is shall the servont to
the same the same that said unto them, The LORD shalt the same the same of
the same the same the same that he said unto them, The LORD said unto them,

Se puede ver que como el modelo es más conservador, entrega sucesiones que son más probables que estén escritas en la Biblia, pero, al mismo tiempo, esto hace que se generen ciclos de frases que aparecen mucho en el conjunto de entrenamiento como por ejemplo “the son of King”. Este problema se puede solucionar elevando un poco la temperatura, como se muestra a continuación con una sucesión generada con 0.5 de temperatura.

And in the house of the LORD in the corner in the mercy of the stones and the
children of Israel the prophesy and how will seek a litter is sin. And the mighty
of it is a standed me with the things the first the wicked in the sons of Saul the
play of the word of the son of Kanah, and at the singer of the morning that have
in the word of the LORD and the word of the LORD and the standing of the
LORD, and went in the people came out of God.

Las palabras ahora son casi todas del idioma inglés y las frases ya empiezan a tener sentido. Al ver estos resultados, se puede ver que el modelo ya genera sucesiones que se aproximan razonablemente al texto analizado. Ahora, se probarán estos modelos con el texto del concurso Hutter, el cual tiene una estructura un poco más complicada que los primeros dos.

4.3.3. Modelo Hutter

Esta base de datos utilizó un modelo muy parecido a los dos anteriores. Es decir, una red neuronal recurrente con unidades LSTM, una tasa de aprendizaje inicial de 0.03, a Adam como algoritmo de optimización, un recorte de gradiente con cota de 1. Para este modelo, debido a sus mayores dimensiones, el tamaño de las batches que se utilizaron fue de 1000 y las sucesiones que se analizaron tuvieron una longitud de 50 caracteres. En todos los modelos se utilizó detención temprana y dropout como regularización.

Esta base de datos es bastante mayor que las otras dos, por lo que cada época tiene una mayor cantidad de datos. En primer lugar, se empleó un modelo con 1 capa oculta y 100 unidades LSTM. El modelo convergió en 3 épocas. El error de validación al final de la tercera época fue de 1.77 y el de prueba fue de 1.73, por lo que parece ser que no se está sobreajustando. Estos resultados están lejos del desempeño esperado por lo que se probarán modelos con más parámetros. A continuación, en la tabla 4.3, se muestran algunas sucesiones generadas por este modelo.

Shelt be *Arlant tech-th leanve's in [[Emperies Gensrotuc'continonicums Joker]] <sup> Wicywarth ISBN]] *[[Ansorsurine Ass anner Highn]]
Austriality woul mentioned by [[Brath]] house special [[Whes Cack" anver prefine datasen]] Wicard.a;/text tiems] [[Ajfhs]]
* [http://www.abrithul.lolmarc" was throrss parer
 pound with inderon be unteendicity and invernity
[[Qumose jurn—[[meltorit]], [[Nature leathers plits]]—hill Effic]] *[[mplet;, Aarc, Strege United WPTR, Afn Husi]]

Tabla 4.3: Sucesiones con primer modelo de Hutter

Se puede ver en estas sucesiones que el modelo ya esta aprendiendo de los datos a escribir inglés y a abrir y cerrar paréntesis bastante bien. Incluso genera direcciones web imaginarias. Se va a utilizar un modelo con más parámetros para aprender más sobre la base de datos.

El siguiente modelo que se probó utilizó una capa oculta y 500 unidades LSTM ocultas. También se utilizaron batches de tamaño 1000 en esta ocasión. En la tabla 4.4 se muestran los errores de validación y los tiempo de entrenamiento para las primeras 4 épocas.

Como se puede ver en la tabla, el error de validación empieza a converger después de la época 4. Se puede ver que el tiempo de entrenamiento para cada época ha aumentado a más

Final de la época 1	tiempo: 23473.43s	error valid 1.60	ppl valid 4.95
Final de la época 2	tiempo: 34589.61s	error valid 1.54	ppl valid 4.68
Final de la época 3	tiempo: 31840.15s	error valid 1.49	ppl valid 4.43
Final de la época 4	tiempo: 42349.47s	error valid 1.48	ppl valid 4.41
Final del entrenamiento		error de prueba 1.44	ppl prueba 4.22

Tabla 4.4: Resultados del entrenamiento del modelo

de 30,000 segundos, es decir, más de 8 horas. Esto se debe en parte al tamaño de la base de datos y al número de parámetros que se tienen que actualizar. En la tabla 4.5, se muestra varios ejemplos de sucesiones generadas con el modelo.

[[Image:Sera Jurischoshaavireson in Expexed a name, diversion of Oe Tagtonninans Players Lemen]] [[Category:Ort Category:Stera de Carregrowing]] Heaps such [[Category:Horn—General General]]
The [[first sea]] and the glast level major most bally on the gla the antitrone hard, the Thind national Depass or earnenton, entropy and a” et teams of ”Ñational fargular)
Gallamorboniaulist on the did not combinative name ones and physically early " has " of the environt, and support as a [[Each chase]] whyrigal later term in thome, each ancient authoring as a particularly subspace

Tabla 4.5: Sucesiones con segundo modelo de Hutter

Al examinar las sucesiones de la tabla 4.5, se puede ver que el modelo no solo ha aprendido palabras en inglés, sino que también las estructuras de sus frases empiezan a tener sentido y ha aprendido a abrir y cerrar brackets así como otros símbolos como el hashtag y hacer ". En la tabla 4.6, se analizarán algunas sucesiones al modificar la temperatura a 0.5.

En las sucesiones de la tabla 4.6, se puede ver que el modelo ya escribe bastante bien inglés, sabe abrir y cerrar paréntesis y tienen intuiciones del language markdown que utiliza el documento, como en el primer ejemplo: “align="left"”.

Los resultados aquí mostrados están lejos aquellos mostrados por (Graves, 2013). Sin embargo, esto se puede explicar por la capacidad computacional a la que se tiene acceso. Graves, en su artículo, al analizar la base de datos del Premio Hutter, utiliza 7 capas ocultas

CAPÍTULO 5

EXPERIMENTO CON CALIGRAFÍA EN LÍNEA

5.1. Introducción

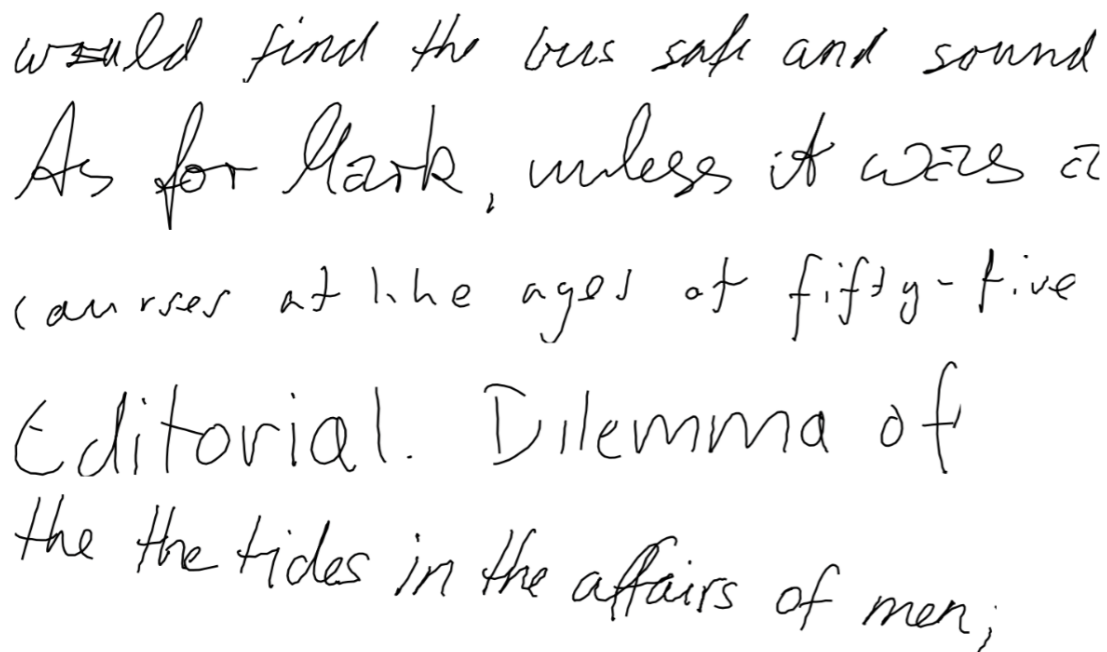
En este capítulo, se usará el modelo de redes neuronales recurrentes con unidades LSTM para analizar la distribución de sucesiones cuyos valores sean reales. Por lo tanto, siguiendo el trabajo de (Graves, 2013), se utilizará la base de datos de caligrafía en línea de IAM (IAM-OnDB) [23]. Se llama caligrafía en línea porque la escritura está registrada como sucesiones de la ubicación de la punta de la pluma. Estos datos son atractivos para la generación de sucesiones debido a su baja dimensionalidad, dos números reales por cada punto, y su facilidad de visualización. [23] [8]

Siguiendo a (Graves, 2013), el objetivo del capítulo será de generar caligrafía en línea con buena legibilidad. Por esta razón y porque no existen “benchmarks” relevantes, los resultados de validación pasarán a segundo plano. Por otro lado, determinar una distribución predictiva que funcione bien con los datos de entrada fue el principal reto para Graves. El modelo que decidió utilizar el científico tiene tres componentes principales: las unidades de memoria LSTM, un mecanismo de atención y una mezcla de gaussianas como capa de salida para generar una red de densidad mixta.

En el siguiente capítulo, se comenzará por dar más detalles sobre la base de datos utilizada, después se explicará el modelo empleado y, finalmente, se mostrarán los resultados obtenidos.

5.2. Descripción de la base de datos

La base de datos IAM-OnDB consiste de enunciados de caligrafía en línea registrados a partir de 221 escritores diferentes utilizando un “pizarrón inteligente”. A los escritores se les pidió escribir enunciados contenidos en la base de texto Lancaster-Oslo-Bergen y la posición de su pluma fue registrada utilizando un dispositivo infrarojo en la esquina del pizarrón. Los datos originales contienen las coordenadas (x, y) de los puntos además de otra variable que indica en qué puntos se levantó la pluma del pizarrón. Predecir el movimiento de la pluma un punto a la vez permite al modelo tener máxima flexibilidad para inventar caligrafía nueva, pero también requiere mucha memoria. La letra promedio ocupa más de 25 periodos y el enunciado promedio alrededor de 700. [8]



would find the bus safe and sound
As for Mark, unless it were a
cancer at the age of fifty-five
Editorial. Dilemma of
the the tides in the affairs of men;

Figura 5.1: Muestra de la base de datos IAM. Figura tomada de (Graves, 2013)

5.3. Descripción del modelo

Como se mencionó en la introducción del capítulo, el modelo utilizado para este experimento tiene 3 partes principales: las unidades de memoria LSTM, un mecanismo de atención y una red de densidad mixta. Las unidades de memoria LSTM se explicaron en el capítulo

3, por lo que, en esta sección, se explicará cómo funciona la densidad mixta en el modelo y después se describirá el mecanismo de atención.

5.3.1. Red de densidad mixta

En la sección 3.2, se dio una breve introducción a redes de densidad mixtas. El objetivo de estas es usar los datos de salida de una red neuronal para parametrizar una mezcla de distribuciones. Una parte de los datos de salida se usan para definir los pesos de mezcla y el resto para parametrizar las componentes de las mezclas individuales. Los pesos de mezcla se normalizan con una función softmax y a los demás datos de salida se les aplican funciones adecuadas para mantener sus valores en un rango adecuado. La red de densidad mixta se entrena maximizando la log probabilidad de los objetivos bajo las distribuciones inducidas. [8]

Cada vector de entrada x_t consiste de un par de números reales x_1, x_2 que define la desviación de la pluma con respecto al dato de entrada anterior, y una variable binaria x_3 que tiene valor 1 cuando el vector termina un trazo, es decir, si la pluma fue levantada del pizarrón antes que el siguiente vector fuera registrado. Una mezcla de gaussianas bivariadas fue usada para predecir x_1 y x_2 , mientras que una distribución Bernoulli fue usada para x_3 . Cada vector de salida y_t consiste de la probabilidad de fin de trazo e , un conjunto de medias μ^j , desviaciones estandar σ^j , correlaciones ρ^j y pesos de mezcla π^j para las M componentes de mezcla. A continuación se pueden ver las ecuaciones que se describieron siguiendo a (Graves, 2013). [8]

$$x_t \in \mathbb{R} \times \mathbb{R} \times \{0, 1\} \quad (5.3.1)$$

$$y_t = (e_t, \{\pi_t^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_{j=1}^M) \quad (5.3.2)$$

La media y desviación estandar son vectores de dimensión 2, mientras que el peso, la correlación y la probabilidad de término de trazo son escalares. Los vectores y_t se obtienen de los datos de salida de la red \hat{y}_t de la siguiente manera: [8]

$$\hat{y}_t = (\hat{e}_t, \{\hat{w}_t^j, \hat{\mu}_t^j, \hat{\sigma}_t^j, \hat{\rho}_t^j\}_{j=1}^M) = b_y + \sum_{n=1}^N W_{h^n y} h_t^n \quad (5.3.3)$$

$$e_t = \frac{1}{1 + \exp(\hat{e}_t)} \quad (5.3.4)$$

$$\pi_t^j = \frac{\exp(\hat{\pi}_t^j)}{\sum_{j'=1}^M \exp(\hat{\pi}_t^{j'})} \quad (5.3.5)$$

$$\sigma_t^j = \exp(\hat{\sigma}_t^j) \quad (5.3.6)$$

$$\rho_t^j = \tanh(\hat{\rho}_t^j) \quad (5.3.7)$$

La función de densidad $Pr(x_{t+1}|y_t)$ para el siguiente dato de entrada x_{t+1} dado el vector de salida y_t se define de la siguiente manera:

$$Pr(x_{t+1}|y_t) = \sum_{j=1}^M \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j) \begin{cases} e_t & (x_{t+1})_3 = 1 \\ 1 - e_t & \text{eoc} \end{cases} \quad (5.3.8)$$

donde

$$\mathcal{N}(x|\mu, \sigma, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left[\frac{-Z}{2(1-\rho^2)}\right] \quad (5.3.9)$$

con

$$Z = \frac{(x_1 - \mu_1)^2}{\sigma_1^2} + \frac{(x_2 - \mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1 - \mu_1)(x_2 - \mu_2)}{\sigma_1\sigma_2} \quad (5.3.10)$$

Por lo tanto, si se utiliza la función de pérdida logarítmica, al sustituir las ecuaciones (5.3.9) y (5.3.10), se obtendrá la siguiente función de pérdida

$$\mathcal{L}(x) = \sum_{t=1}^T -\log\left(\sum_j \pi_t^j \mathcal{N}(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j)\right) - \begin{cases} \log e_t & (x_{t+1})_3 = 1 \\ \log(1 - e_t) & \text{eoc} \end{cases} \quad (5.3.11)$$

En la figura 5.2, se puede ver que hay dos tipos de predicción en la capa de densidad mixta: las pequeñas burbujas que denotan los trazos que se hacen al escribir las letras, y unas burbujas más grandes que son las predicciones al terminar un trazo y para comenzar el siguiente. Las predicciones al final de los trazos tienen una varianza más alta porque la posición de la pluma no fue registrada cuando estuvo despegada del pizarrón y, por lo tanto, puede haber una distancia más grande entre el final de un trazo y el siguiente. [8]

La capa de densidad mixta es muy útil para generar caligrafía convincente. Sin embargo, todavía hace falta la parte del modelo que permitirá generar las sucesiones de texto que se requieran. Esta parte del modelo, llamada mecanismo de atención, se presentará a continuación.

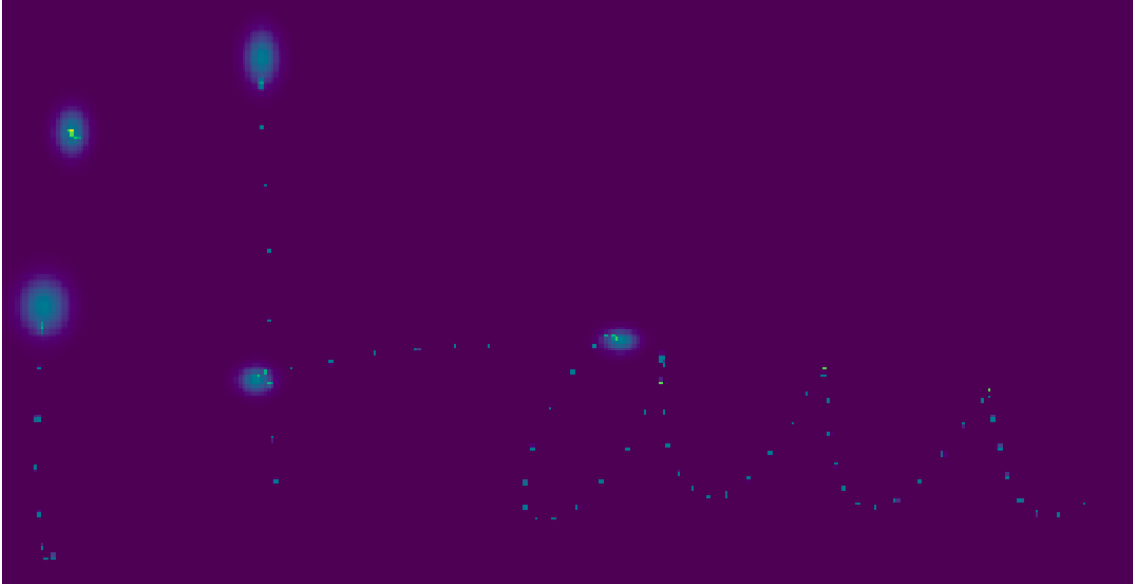


Figura 5.2: La operación de la capa de densidad mixta aplicada a una predicción de caligrafía al escribir a palabra “itam”.

5.3.2. Mecanismos de atención

Para poder generar caligrafía para un cierto texto dado, se condicionarán las predicciones de la caligrafía en el texto que se quiere escribir. Sin embargo, uno de los problemas más importantes al tratar de hacer esto es que la sucesión de caligrafía es 25 veces más grande en promedio que el texto. Otro reto es que la alineación entre el texto y la caligrafía no se conoce hasta que los datos son generados debido a que el número de coordenadas usadas para escribir cada caracter cambia de manera importante de acuerdo al estilo, tamaño, velocidad de la pluma, etc. [8]

Para resolver este problema, siguiendo a (Graves, 2013), en este trabajo se creará una mecanismo de atención que consiste de una “ventana suave” que es convolucionada con el texto y luego alimentada como dato de entrada para la red de predicción. Los parámetros de la ventana se obtienen del modelo al mismo tiempo que se hacen las predicciones, para que dinámicamente se determine una alineación entre la ubicación del texto y de la pluma. En otras palabras, el modelo aprende a decidir que caracter escribir después.

La ventana suave se define a continuación. Dada una sucesión c con U caracteres, la ventana suave w_t con respecto al paso de tiempo t ($1 \leq t \leq T$) se define mediante la siguiente convolución con una mezcla de K funciones gaussianas donde $\phi(t, u)$ es el peso de la ventana de c_u en el tiempo t . [8]

$$\phi(t, u) = \sum_{k=1}^K \alpha_t^k \exp(-\beta_t^k (\kappa_t^k - u)^2) \quad (5.3.12)$$

$$w_t = \sum_{u=1}^U \phi(t, u) c_u \quad (5.3.13)$$

Intuitivamente, explica Graves, el parámetro κ_t controla la ubicación de la ventana, los parámetros β_t controlan el ancho de la ventana y los parámetros α_t controlan la importancia de la ventana dentro de la mezcla. Por otro lado, los parámetros de la ventana se calculan de la siguiente manera a partir de los datos de salida de la primera capa oculta del modelo. [8]

$$(\hat{\alpha}_t, \hat{\beta}_t, \hat{\kappa}_t) = W_{h^1 p} h_t^1 + b_p \quad (5.3.14)$$

$$\alpha_t = \exp(\hat{\alpha}) \quad (5.3.15)$$

$$\beta_t = \exp(\hat{\beta}) \quad (5.3.16)$$

$$\kappa_t = \kappa_{t-1} + \exp(\hat{\kappa}) \quad (5.3.17)$$

Los parámetros de ubicación κ_t se definen como la diferencia con respecto a la ubicación anterior κ_{t-1} para poder alinear el texto con el trazo de la pluma. Después de esto, los vectores w_t se pasan hacia las siguientes capas ocultas en el tiempo t y hacia la primera capa oculta en el tiempo $t + 1$ como datos de entrada.

5.4. Resultados

A continuación se mostrarán los resultados que entregó el algoritmo implementado por el autor. El código se puede encontrar en <http://github.com/GeorBelanger/>

Para elegir los hiperparámetros del modelo, se tomó como base (Graves, 2013). Sin embargo, debido a que no se contaba con la suficiente capacidad computacional, se redujeron algunos parámetros para poder entrenar el modelo en un tiempo razonable, como por ejemplo el número de unidades LSTM en cada capa oculta.

El modelo utilizó 3 capas ocultas con 200 unidades LSTM en cada una, los batches con tamaño 64, 8 gaussianas en la capa de salida, a rmsprop como algoritmo de optimización y recorte de gradiente a 10. La red recurrente se desenrolló 150 pasos de tiempo y el promedio de pasos de tiempo o trazos de cada letra se fijó en 25. Por lo que el algoritmo se fija

en promedio en las 6 letras anteriores para hacer el siguiente trazo. Para la generación de sucesiones, se escogieron frases que tuvieran que ver con la misión, filosofía o mercadotecnia del ITAM.

El algoritmo convergió después de 13,000 iteraciones. En las figuras 5.3 y 5.4 se pueden ver algunos resultados al generar la caligrafía condicionada en la frase “solo hay un itam”.

A handwritten cursive phrase "Solo hay un itam" in blue ink on a white background. The letters are connected in a fluid, cursive style.

Figura 5.3: Generación de caligrafía de nuestro modelo condicionado con respecto a la frase “solo hay un itam”



Figura 5.4: Operación de la capa de densidad mixta al generar caligrafía condicionada con respecto a la frase “solo hay un itam”

En la figura 5.5, se puede ver la generación de caligrafía basada en la frase “i love deep learning”. Se puede ver que en última parte de la frase, hay demasiado espacio entre las letras. Esto se debe a que el mecanismo de atención todavía tiene a veces problemas para alinear la caligrafía con el texto que se quiere generar.

A handwritten cursive phrase "i love deep learning," in blue ink on a white background. The letters are connected in a fluid, cursive style. There is noticeable spacing between the letters in the final part of the phrase.

Figura 5.5: Generación de caligrafía de nuestro modelo condicionado con respecto a la frase “i love deep learning”.

A continuación, se entrenará un modelo más compacto por más tiempo y se compararán los resultados. La arquitectura de este segundo modelo utilizó solamente 2 capas ocultas más

el mecanismo de atención presentado. Además, cada capa oculta contó con 100 unidades LSTM, las capa de salida utilizó 4 gaussianas y el mecanismo de atención 2 gaussianas. Este modelo se entrenó por 60,000 épocas. En la figura 5.6, se puede ver la primera generación de caligrafía de este modelo condicionada en la frase “Por un México más libre”.



Figura 5.6: Generación de caligrafía de nuestro modelo condicionado con respecto a la frase “Por un México más libre”

De igual forma que se hizo en el capítulo 4, en este modelo también podemos hacer que el modelo entregue elementos más probables de $P(x|c)$ para que las sucesiones sean más estables. Para hacer esto, se sesgará el muestreo hacia predicciones más probables en cada paso. Si b es el sesgo de probabilidad, un real mayor o igual a cero, antes de muestrear de $Pr(x_{t+1}|y_t)$, cada desviación estandar σ_t^j en la mezcla gaussiana de la ecuación 5.3.6, se puede recalcular de la siguiente manera: [8]

$$\sigma_t^j = \exp(\hat{\sigma}_t^j - b) \quad (5.4.1)$$

Y los pesos de la mezcla se pueden recalcular de la ecuación 5.3.5 de la siguiente manera:

$$\pi_t^j = \frac{\exp(\hat{\pi}_t^j(1+b))}{\sum_{j'=1}^M \exp(\hat{\pi}_t^{j'}(1+b))} \quad (5.4.2)$$

Esto reduce artificialmente la varianza tanto en la elección del componente de mezcla como en la distribución del componente. Cuando $b = 0$ se obtiene un muestreo insesgado y, conforme b tiende a infinito, la varianza en el muestreo desaparece. En las figuras 5.7 y 5.8, se puede ver la generación de sucesiones sesgadas basadas en la frase “Por un Mexico más libre, más justo y más próspero”.

Los resultados aquí expuestos muestran el poder que tienen los modelos explicados para generar sucesiones complejas de dependencias de largo plazo. La calidad de la escritura todavía podría mejorar si como (Graves, 2013) se utilizara un modelo más grande y se entrenara por más tiempo. Sin embargo, no se cuenta con la capacidad computacional necesaria y para este trabajo bastará mostrar el potencial que tienen los modelos. [8]

The image shows the phrase "por un Mexico mas libre" written in a fluid, cursive script. The letters are connected, and the overall style is elegant and handwritten. The color is a vibrant blue.

Figura 5.7: Generación de caligrafía de nuestro modelo condicionado con respecto a la frase “Por un México más libre”

The image shows the phrase "mas justo y mas prospero" written in a fluid, cursive script. The letters are connected, and the overall style is elegant and handwritten. The color is a vibrant blue.

Figura 5.8: Generación de caligrafía de nuestro modelo condicionado con respecto a la frase “más justo y más próspero”

CAPÍTULO 6

CONCLUSIONES

Para concluir este trabajo, se dará un pequeño resumen de lo que se ha hecho, se darán algunas conclusiones generales, se describirán un poco los rumbos que ha tomado esta investigación en aprendizaje profundo y, finalmente, se darán algunos comentarios de cierre.

En este trabajo, en primer lugar, en el capítulo 2, se revisaron los conceptos básicos y que se consideraron necesarios de aprendizaje de máquina. En segundo lugar, se revisaron los conceptos de aprendizaje profundo para dar una buena introducción y un marco teórico necesario para entender a las redes neuronales y redes neuronales recurrentes. Después en el capítulo 4 y 5 se realizaron experimentos de generación de sucesiones prediciendo un paso a la vez los siguientes datos: en el capítulo 4, el carácter que sigue basado en los anteriores y, en capítulo 5, el trazo que sigue basado en los anteriores.

Con los experimentos, se pudo demostrar, siguiendo a (Graves, 2013), que las redes neuronales recurrentes con unidades LSTM pueden generar sucesiones discretas o reales con estructura compleja y de largo alcance al predecir un paso a la vez. También se presentó, siguiendo a Graves, el mecanismo de atención que permite sintetizar caligrafía en línea al condicionar las predicciones del modelo en un texto dado. Al mismo tiempo, también se comprobó que para hacer que los algoritmos aprendan y tengan un desempeño altamente competitivo, será necesario tener la capacidad computacional necesaria. En este trabajo, se utilizó una computadora personal y aunque los resultados no son altamente competitivos, ya se deja ver el poder que tienen los algoritmos estudiados. [8]

Graves propone dos direcciones para continuar su trabajo. La primera tiene que ver con entender mejor la representación interna de los datos para manipular la distribución de muestreo directamente para hacer mejores predicciones. En esta misma línea, Graves sugiere desarrollar un mecanismo que extraiga anotaciones de alto nivel de los datos secuenciales para poder tener resultados con más estructura y poder tener información sobre los diferentes

estilos, las diferentes formas de las letras y el orden de los trazos. [8]

La segunda dirección que propone Graves es utilizar el modelo para hacer síntesis de voz, la cual es un poco más desafiante que la caligrafía pues hay una mayor dimensionalidad de los datos. Graves ya ha continuado esta investigación el mismo. En 2013, publicó el artículo “Hybrid speed recognition with deep bidirectional LSTM” en el que se demuestra que su modelo también tiene buenos resultados en síntesis de voz. [8]

La inteligencia artificial promete traer una revolución en muchas áreas de la vida. En los siguientes 10 o 20 años, se cree que habrá una nueva “revolución industrial”: la inteligencia artificial ayudará al humano en muchas tareas y, por otro lado, habrá muchos trabajos que dejarán de existir. Sin embargo, es importante recalcar que todos seremos más ricos pues será más barato hacer muchos productos. No obstante, será esencial crear un marco económico en el que todos se puedan beneficiar de esto y los beneficios no se vayan a unas cuantas manos. [6]

BIBLIOGRAFÍA

- [1] C. M. BISHOP, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [2] J. CORTAZAR, *Rayuela — instituto latinoamericano de la comunicación educativa*, 2017. red.ilce.edu.mx/sitios/micrositios/cortazar_aniv/pdf/8_Cielo_Rayuela_libro.pdf.
- [3] J. CORTÁZAR, *Rayuela*, Biblioteca Cortázar, Alfaguara, 1996.
- [4] L. DENG AND D. YU, *Deep learning: Methods and applications*, tech. rep., May 2014.
- [5] F. A. GERS, J. SCHMIDHUBER, AND F. CUMMINS, *Learning to forget: Continual prediction with LSTM*, Neural Computation, 12 (2000), pp. 2451–2471.
- [6] I. GOODFELLOW, Y. BENGIO, AND A. COURVILLE, *Deep Learning*, MIT Press, 2016. <http://www.deeplearningbook.org>.
- [7] A. GRAVES, *Deep learning lecture 13: Alex graves on hallucination with rnns*. <https://www.youtube.com/watch?v=-yX1SYeDHbg>.
- [8] —, *Generating sequences with recurrent neural networks*, CoRR, abs/1308.0850 (2013).
- [9] T. HASTIE, R. TIBSHIRANI, AND J. FRIEDMAN, *The Elements of Statistical Learning*, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.
- [10] J. HAUGELAND, *Artificial Intelligence: The Very Idea*, Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.
- [11] S. HAYKIN, *Neural Networks: A Comprehensive Foundation*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd ed., 1998.

- [12] S. HOCHREITER AND J. SCHMIDHUBER, *Long short-term memory*, Neural Comput., 9 (1997), pp. 1735–1780.
- [13] A. KARPATY, *The unreasonable effectiveness of recurrent neural networks*. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>.
- [14] T. M. MITCHELL, *Machine Learning*, McGraw-Hill, Inc., New York, NY, USA, 1 ed., 1997.
- [15] K. P. MURPHY, *Machine Learning: A Probabilistic Perspective*, The MIT Press, 2012.
- [16] V. ÑAIR AND G. E. HINTON, *Rectified linear units improve restricted boltzmann machines.*, in ICML, J. Fürnkranz and T. Joachims, eds., Omnipress, 2010, pp. 807–814.
- [17] R. PASCANU, T. MIKOLOV, AND Y. BENGIO, *Understanding the exploding gradient problem*, CoRR, abs/1211.5063 (2012).
- [18] D. E. RUMELHART, G. E. HINTON, AND R. J. WILLIAMS, *Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1*, MIT Press, Cambridge, MA, USA, 1986, ch. Learning Internal Representations by Error Propagation, pp. 318–362.
- [19] H. SAK, A. W. SENIOR, AND F. BEAUFAYS, *Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition*, CoRR, abs/1402.1128 (2014).
- [20] N. SRIVASTAVA, G. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, AND R. SALAKHUTDINOV, *Dropout: A simple way to prevent neural networks from overfitting*, J. Mach. Learn. Res., 15 (2014), pp. 1929–1958.
- [21] E. K. STEVEN BIRD AND E. LOPER, *Natural language toolkit*. http://www.nltk.org/nltk_data/.
- [22] I. SUTSKEVER, O. VINYALS, AND Q. V. LE, *Sequence to sequence learning with neural networks*, CoRR, abs/1409.3215 (2014).
- [23] H. B. U. MARTI AND M. ZIMMERMAN, *Iam handwriting database*. <http://www.fki.inf.unibe.ch/databases/iam-handwriting-database>.
- [24] V. VANHOUCKE AND A. CHACRKABORTY, *Udacity course: Deep learning by google*. <https://www.udacity.com/course/deep-learning--ud730>.
- [25] D. WOLPERT, *The lack of a priori distinctions between learning algorithms*, Neural Computation, (1996).