

Optimizadores en redes neuronales profundas: un enfoque práctico



Luis Velasco

7 min read

Apr 26, 2020

1

Cuando nos enfrentamos al entrenamiento de una red neuronal, la decisión de que optimizador seleccionar parece estar envuelta en un halo de misterio, ya que en general la literatura alrededor de los optimizadores requiere de bastante bagaje matemático.

De cara a definir un criterio práctico, vamos a realizar una serie de experimentos para ver el comportamiento de diferentes optimizadores en problemas canónicos del aprendizaje automático. Así podremos elegir un optimizador de forma sencilla.

Marco teórico

En esencia, el objetivo del entrenamiento de redes neuronales es minimizar la función de coste encontrando los pesos adecuados para las aristas de la red (asegurando, eso sí, una buena generalización). El descubrimiento de estos pesos se lleva a cabo mediante un algoritmo numérico llamado *backpropagation* que puede resumirse así:

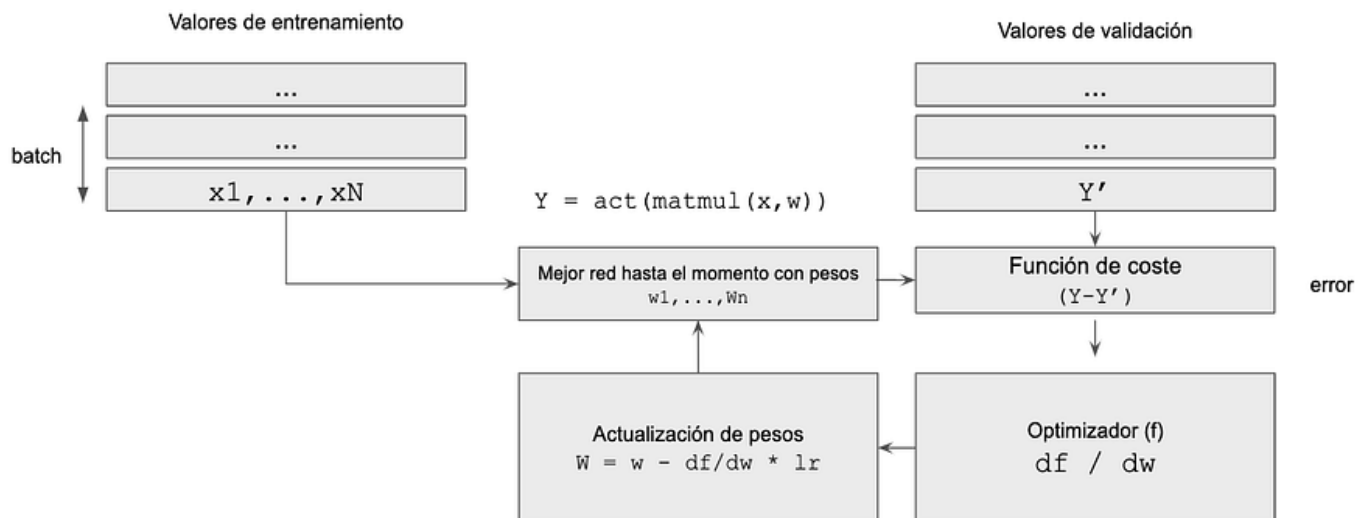


Figura 1 — Algoritmo de *backpropagation* para entrenamiento de redes neuronales

El elemento en cuestión de este estudio, el optimizador, es el encargado de generar pesos cada vez mejores: su importancia es crucial. Su funcionamiento esencial se basa en calcular el gradiente de la función de coste (derivada parcial) por cada peso (parametro/dimension) de la red. Como queremos minimizar el error, modificaremos cada peso en la dirección (negativa) del gradiente:

$$W_{t+1} = W_t - df(coste) / dW * lr$$

De cara a agilizar la convergencia de la función de coste hacia su mínimo, multiplicamos el vector de gradiente por un factor llamado factor de entrenamiento (*lr*).

El conjunto de métodos iterativos de reducción de la función de error (búsqueda de un mínimo local), son conocidos como los método de optimización basados en el gradiente descendente.

Discusión de los principales optimizadores disponibles en keras

A continuacion, presentaremos una vision intuitva de los principales optimizadores.

Stochastic Gradient Descent (SGD)

El cálculo de la derivada parcial de la función de coste respecto a cada uno de los pesos de la red para cada observación es, dado el número de diferentes pesos y observaciones, inviable. Por lo tanto, una primera optimización consiste en la introducción de un comportamiento estocástico (aleatorio). SGD hace algo tan simple como limitar el cálculo de la derivada a tan solo una observación (por batch). Existen algunas variaciones basadas por ejemplo en seleccionar varias observaciones en vez de una (mini-batch SGD). Una variación particularmente interesante es la introducción de *momentum*. De forma intuitiva, el *momentum* acelera el descenso en direcciones similares a las anteriores. Para ello, vamos a guardar un vector que representa la media en ventana de los anteriores vectores de descenso, y si el nuevo vector es similar al vector de *momentum* aceleramos su descenso. Otra variación se basa en el optimizador de gradiente acelerado de Nesterov. Para el cálculo del descenso, en un primer momento “confiamos” en el vector de *momentum* y una vez descendido en su dirección computamos el nuevo gradiente desde ese punto.

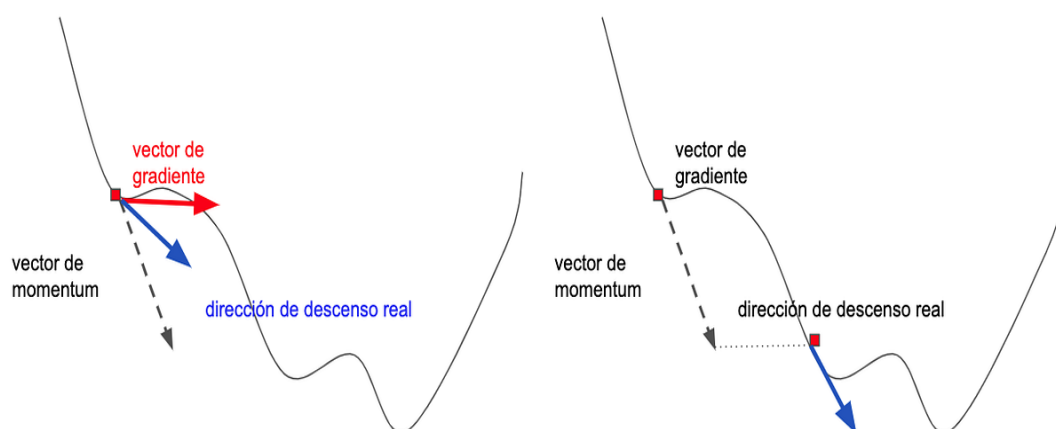


Figura 2 — SGD con aceleración y con aceleración de Nesterov

Adaptive Gradient Algorithm (AdaGrad)

El algoritmo AdaGrad introduce una variación muy interesante en el concepto de factor de entrenamiento: en vez de considerar un

valor uniforme para todos los pesos, se mantiene un factor de entrenamiento específico para cada uno de ellos. Sería inviable calcular este valor de forma específica así que, partiendo del factor de entrenamiento inicial, AdaGrad lo escala y adapta para cada dimensión con respecto al gradiente acumulado en cada iteración.

Adadelta

Adadelta es una variación de AdaGrad en la que en vez de calcular el escalado del factor de entrenamiento de cada dimensión teniendo en cuenta el gradiente acumulado desde el principio de la ejecución, se restringe a una ventana de tamaño fijo de los últimos n gradientes.

RMSprop (Root Mean Square Propagation)

RMSProp es un algoritmo similar. También mantiene un factor de entrenamiento diferente para cada dimensión, pero en este caso el escalado del factor de entrenamiento se realiza dividiéndolo por la media del declive exponencial del cuadrado de los gradientes (*glup!*)

Adam (Adaptive moment estimation)

El algoritmo Adam combina las bondades de AdaGrad y RMSProp. Se mantiene un factor de entrenamiento por parámetro y además de calcular RMSProp, cada factor de entrenamiento también se ve afectado por la media del *momentum* del gradiente.

Como se acaba de comprobar, los algoritmos mas recientes como Adam, están contruidos en base a sus predecesores, por tanto podremos esperar que su rendimiento sea superior.

Desde mi punto de vista y salvando SGD, es complicado tener una visión intuitiva del comportamiento de cada uno de los optimizadores; por ello es útil visualizar su rendimiento en

diferentes funciones de coste. Estas visualizaciones son bastante reveladoras:

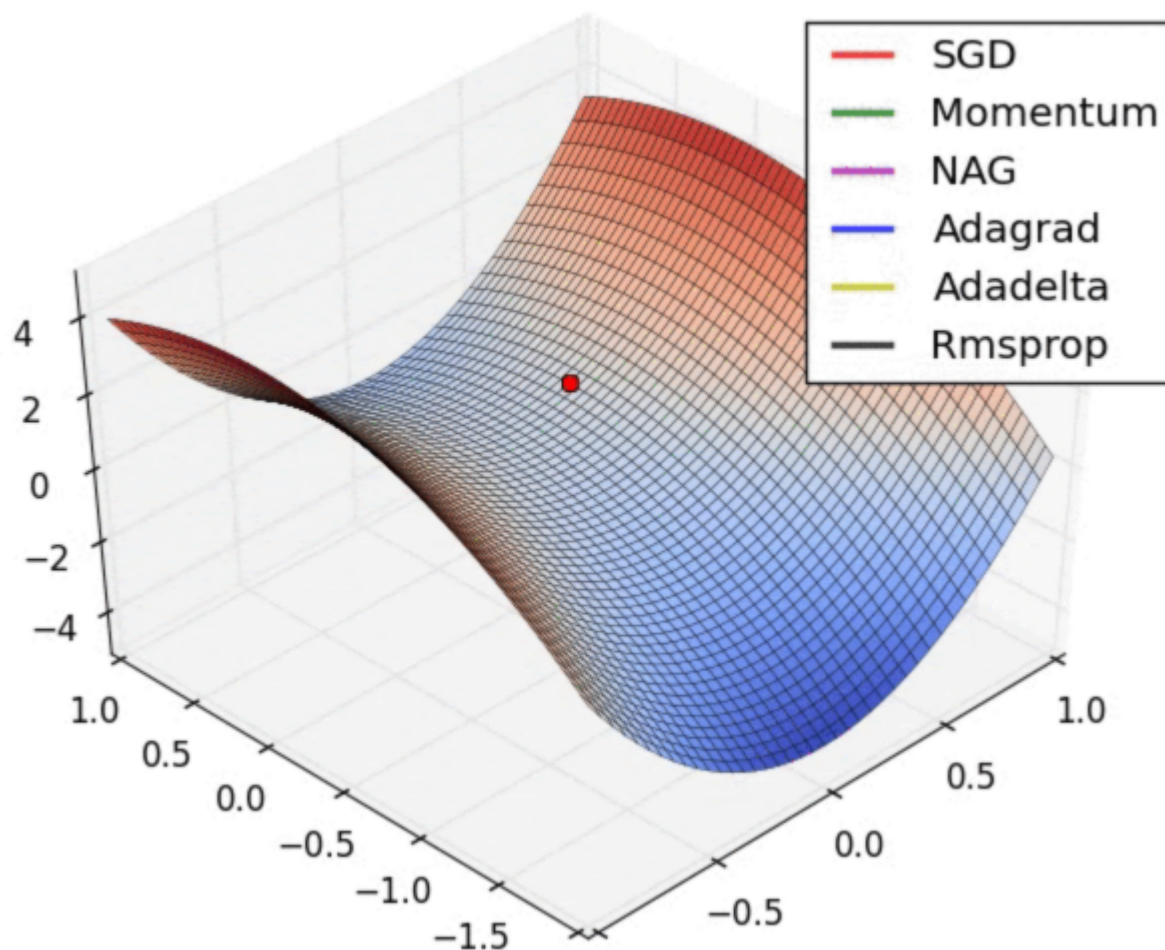


Figura 3 — Convergencia en funciones de coste de tipo valle profundo

Función de coste con valle profundo: los algoritmos que escalan el tamaño del paso basados en el gradiente convergen. Resulta interesante que SGD nunca desciende en la dirección adecuada.

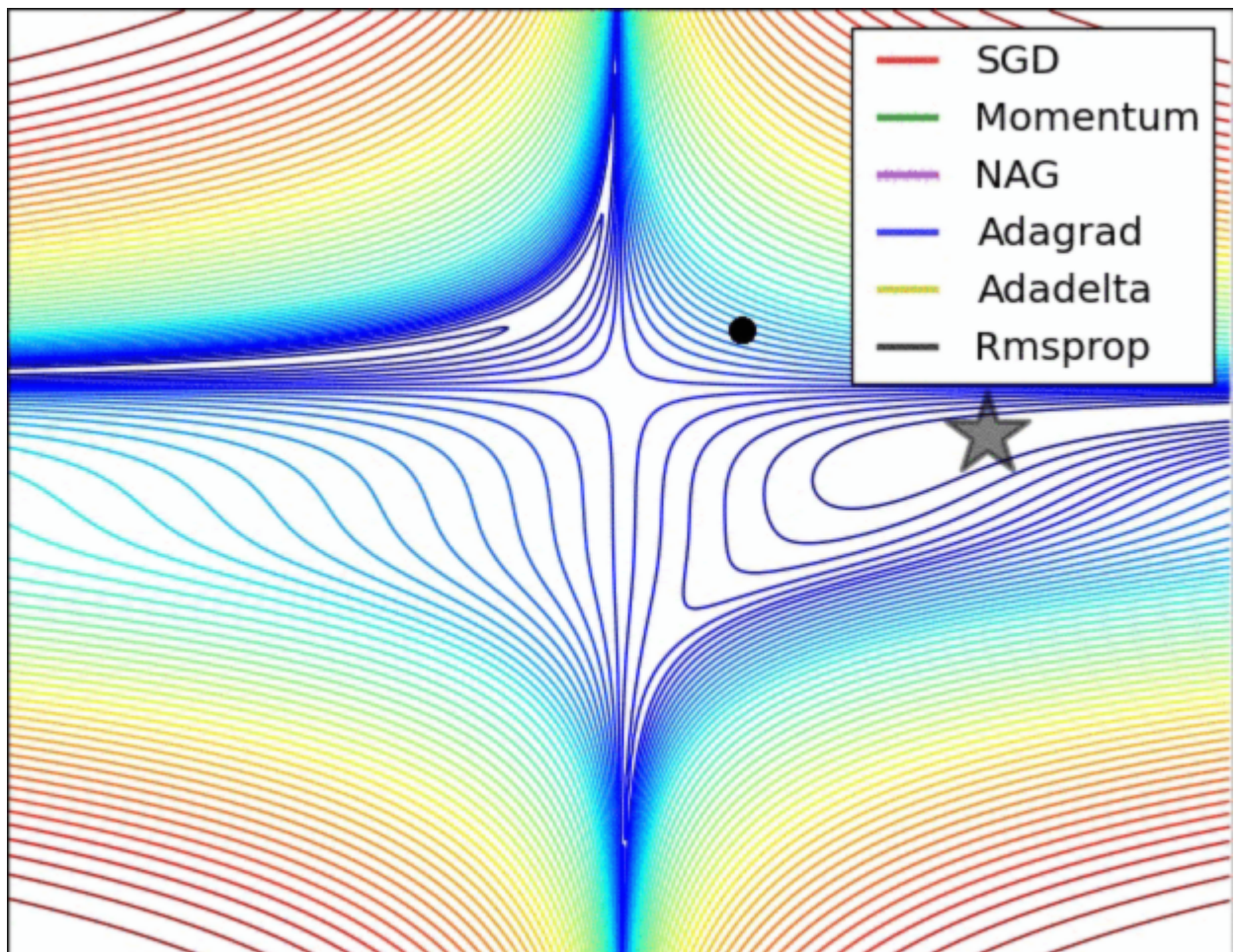


Figura 4 — Convergencia en funciones de coste tipo Beale

Función de coste en siguiendo el patrón Beale: debido a lo pronunciado del gradiente en los contornos, los algoritmos acelerados divergen, resultandoles difícil “frenar”.

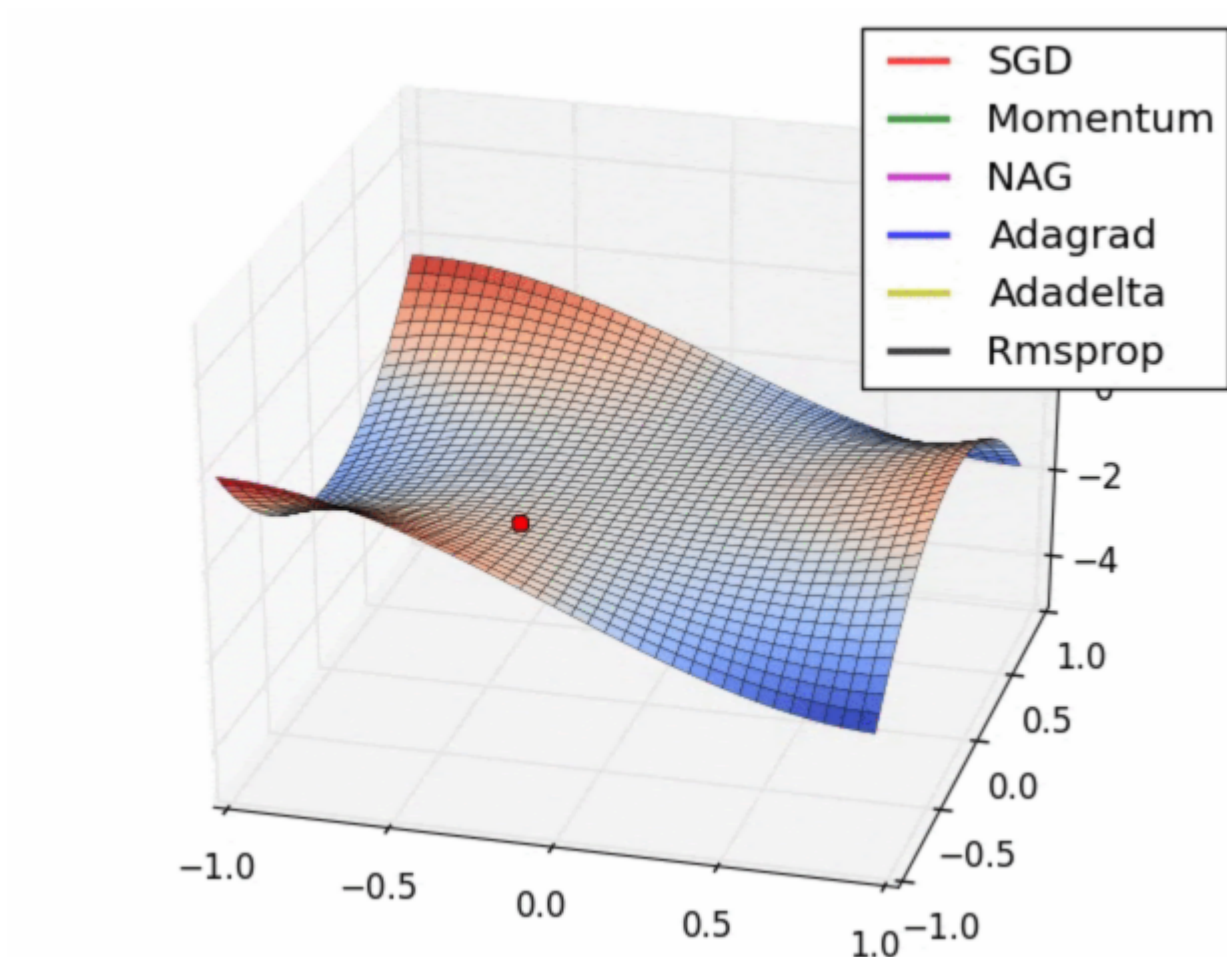


Figura 5 — Convergencia en funciones alrededor de un punto de silla

Función de coste con forma de silla de montar: si bien todos los algoritmos acaban convergiendo en las cercanías de un punto de silla, Adadelta/Adagrad/RMSProp lo hacen mucho más rápido.

Desde un punto de vista práctico, en un entorno real es imposible avanzar la taxonomía de la función de coste (al menos para mi!) dada una primera versión de una red neuronal. Por ello, creo que es mejor pasar a un experimento más cercano.

Definición del experimento

De cara a dar la mayor amplitud posible al estudio, he seleccionado los siguientes cuatro problemas clásicos en el ámbito del aprendizaje automático. Además de representar ámbitos funcionales diferentes, cada uno de ellos expone una arquitectura de red específica y representativa:


Tipo de problema	Arquitectura de red	Juego de datos
Regresion simple	Red neuronal simple	Precio de viviendas en Boston
Clasificador (multiclase) de imágenes	Red convolucional 2D	Fashion-MNIST
Clasificador (binario) de textos	Red convolucional 1D + Embeddings	Análisis sentimiento IMDB 
Forecasting de series temporales	Red neuronal recurrente con LSTM	Prediccion temperatura

Tabla 1 — Definición del experimento

Los optimizadores con los que entrenaremos cada red son los siguientes (todos los disponibles en Keras):

- Adadelta
- Adagrad
- Adam
- Adamax
- Ftrl
- Nadam
- RMSprop
- SGD (sin Nesterov y sin momentum)

En cuanto a las funciones de coste, usaremos MAE para las regresiones y forecast, binary cross entropy para el clasificador binario y categorical cross entropy para el clasificador multiclase. Mantendremos los valores por defecto del factor de aprendizaje y los parámetros específicos de cada optimizador (learning_rate=0.001).

Utilizaremos el siguiente esqueleto:


```

results = {}
history = {}
for optimizer in optimizers_list:
    model = keras.Sequential([...])
    model.compile(loss='...', optimizer=optimizer, metrics=['...'])
    optimizer_key = str(type(optimizer).__name__)
    history[optimizer_key] = model.fit(...)
    results[optimizer_key] = {}
    results[optimizer_key]["loss"] = history[optimizer_key].history['loss'][EPOCHS - 1]
    results[optimizer_key]["val_loss"] = history[optimizer_key].history['val_loss'][EPOCHS - 1]

```

Figura 6— Esqueleto del cálculo de las métricas en keras

Resultados y conclusiones

A continuación los resultados de las ejecuciones. El notebook con los experimentos se pueden encontrar en [github](#)

Experimento 1: Regresión numérica simple

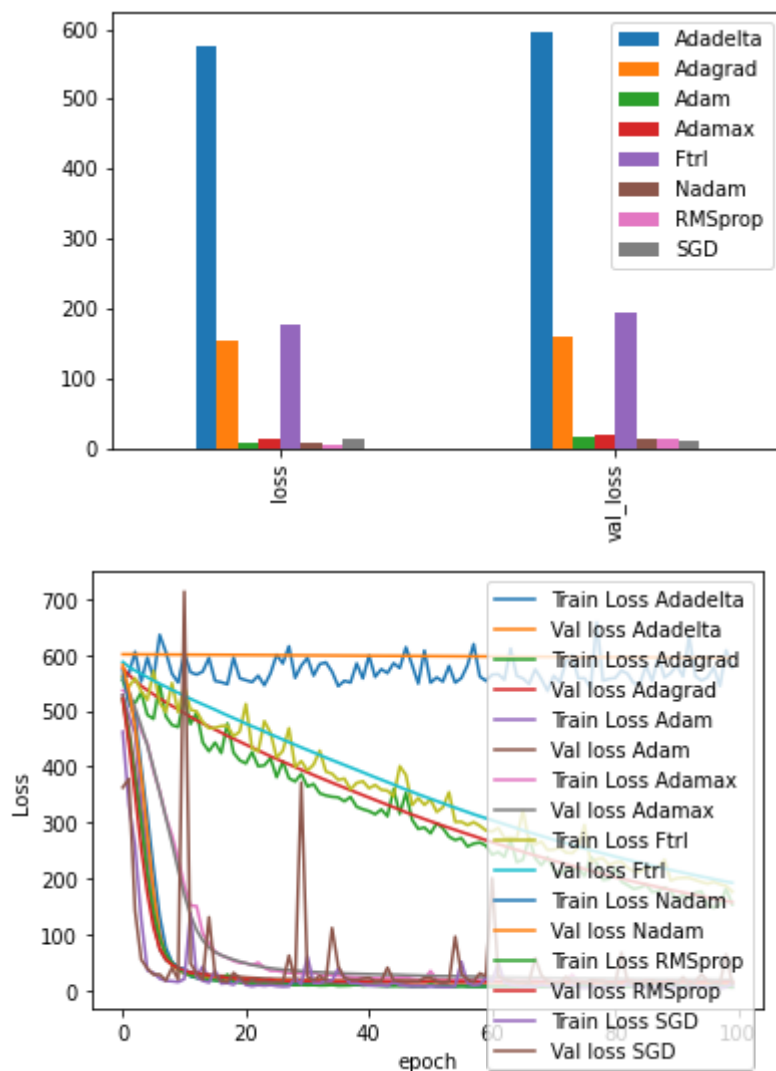


Figura 7— Convergencia de los diferentes optimizadores en una regresión numérica simple

En general, observamos tres tipos de comportamientos: (i) Adadelata no converge, (ii) Adagrad y Ftrl convergen de una forma lineal sub-óptima y (iii) los restantes algoritmos acaban encontrando un mínimo y además presentan buena generalización. Es importante destacar que el comportamiento de SGD es bastante irregular con múltiples “idas y venidas” en el proceso de convergencia. En términos absolutos, RMSProp presenta el mejor comportamiento.

Experimento 2: Clasificador (multi-clase) de imágenes

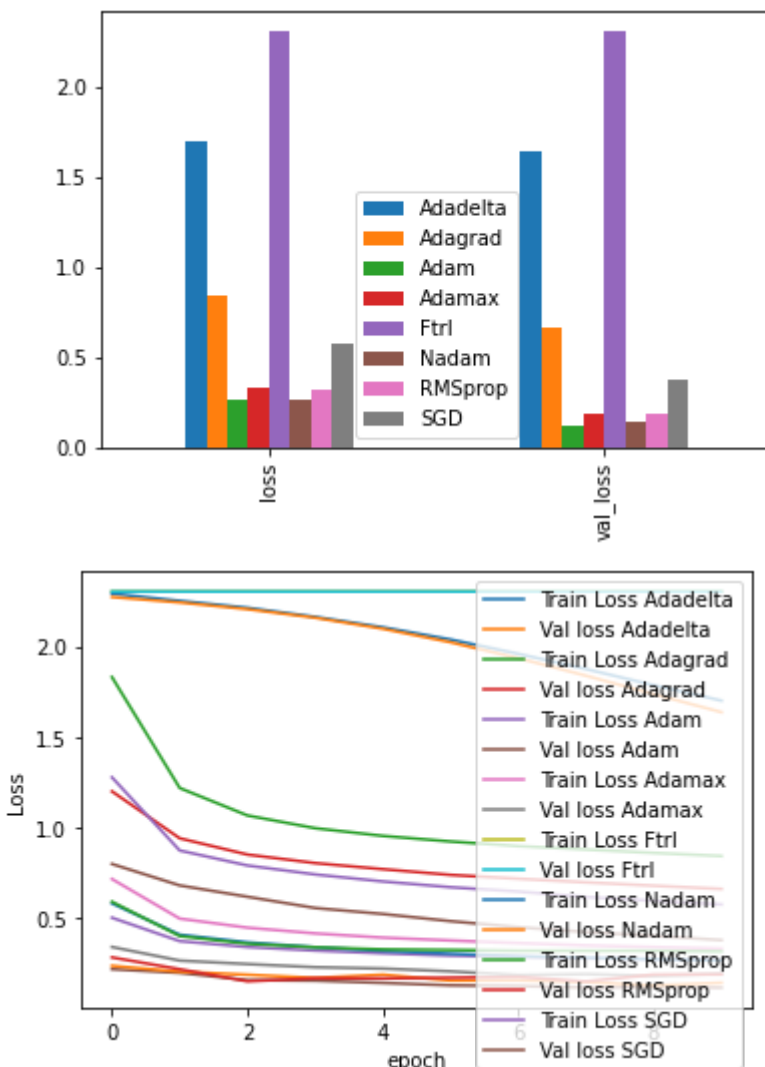


Figura 8— Convergencia de los diferentes optimizadores en clasificador CNN multiclase

En este caso, ni Adadelata ni Ftrl presentan buenos comportamientos. El resto de algoritmos sigue un patrón de

minimización del error similar, cada uno a diferente velocidad, de aumentar el número de epochs probablemente todos acabarán en una situación similar. Adam presenta el mejor comportamiento general y RMSProp la convergencia mas rapida.

Experimento 3: Clasificador (binario) de textos

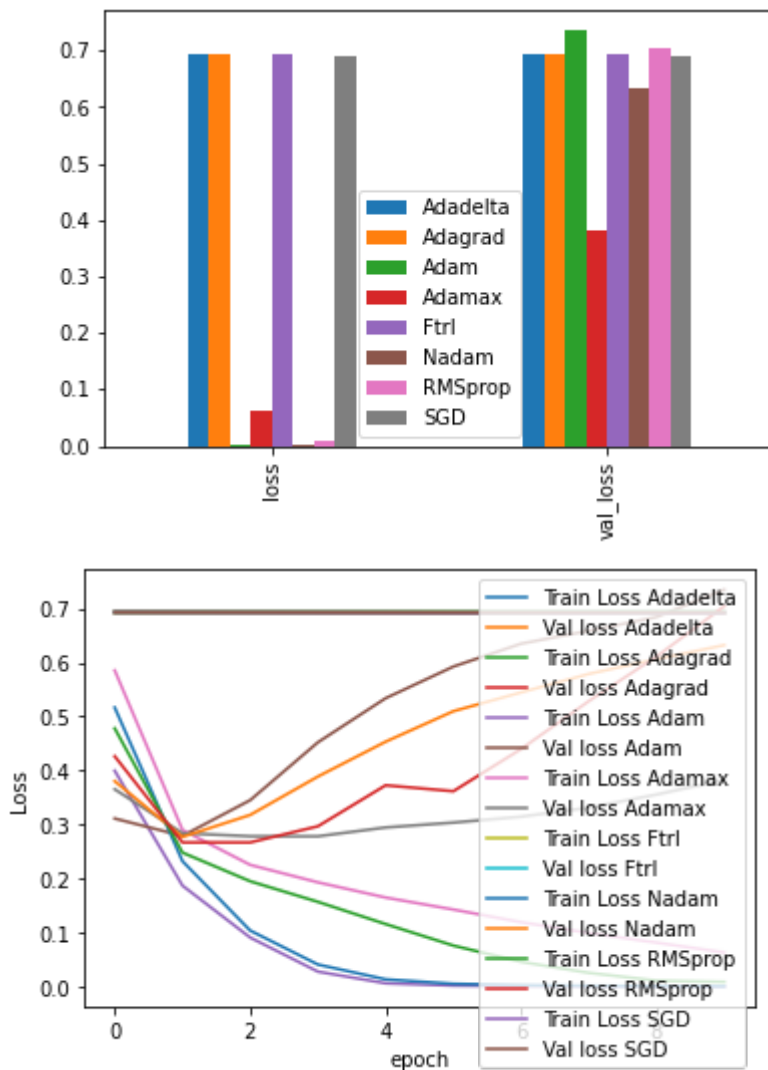


Figura 9 — Convergencia de los diferentes optimizadores en una clasificador CNN binario con Embeddings

El entrenamiento de esa red presenta un comportamiento peculiar, vemos cómo se presenta el fenómeno de overfitting con casi todos los optimizadores. Por tanto deberíamos explorar entrenar con más epochs, aplicar regularización o drop-outs. RMSProp presenta el mejor valor absoluto y Adam la convergencia mas eficiente.

Experimento 4: Forecasting de series temporales

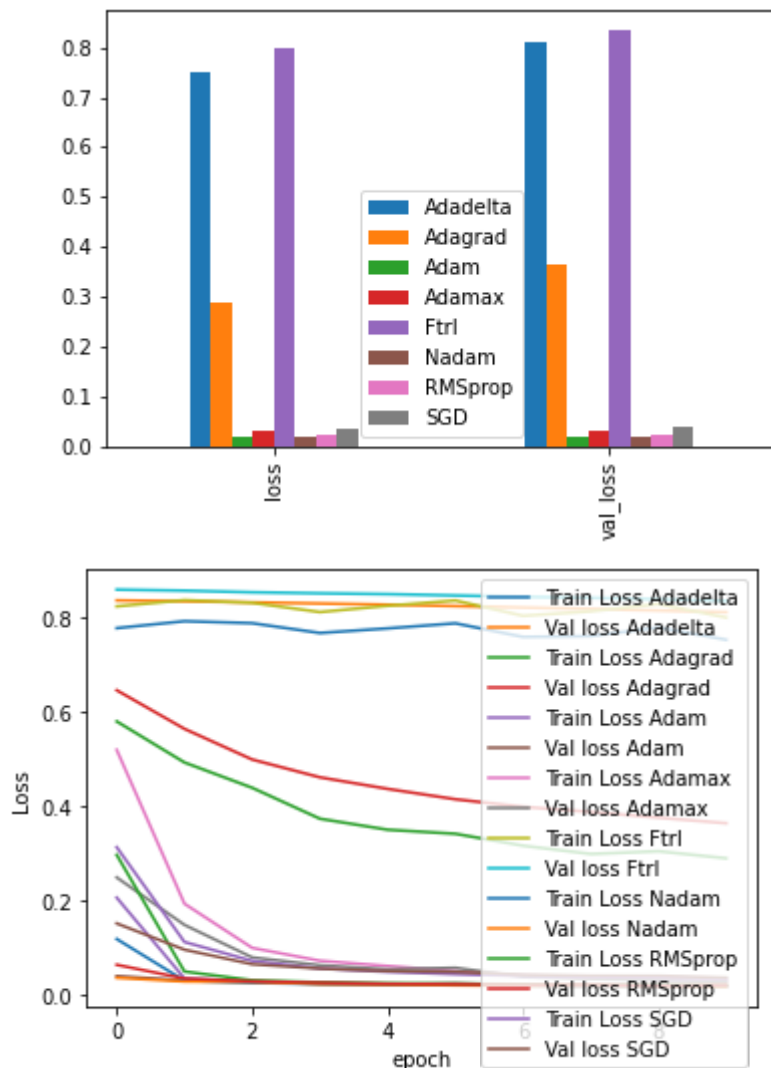


Figura 10 — Convergencia de los diferentes optimizadores con RNN LSTM

En este último escenario, de nuevo observamos tres diferentes patrones de convergencia. (i) Ftrl y Nadam no encuentran el mínimo (Ftrl es esperado, dada su particularidad aplicabilidad). (ii) Adagrad presenta una convergencia sub-óptima y (iii) el resto de algoritmo un comportamiento similar, siendo Adam y RMSProp los que presentan un mejor comportamiento.

*En conclusión hemos observado de una forma empírica cómo quizás **el algoritmo Adam presenta un comportamiento adecuado en diferentes problemas**, por tanto puede ser un buen candidato para empezar a probar en nuestros modelos.*

Un factor de extrema importancia (quizás el hyper-parámetro más importante) que no podemos pasar por alto y que hemos supuesto constante es el factor de entrenamiento. En sucesivas notas exploraremos, cómo una vez seleccionado un optimizador, podemos ajustar su factor de entrenamiento y entender el compromiso entre este factor, los tiempos de ejecución y la convergencia.

Machine Learning

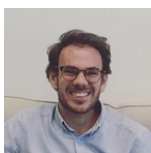
Deep Learning

AI

Keras

Adam

1



Written by Luis Velasco

274 Followers

Data, ML and everything in between. Working @ Google