Introducción a las redes neuronales artificiales

Ignacio Haeussler

2 de noviembre de 2017

1. Introducción

Las redes neronales artificiales (ANNs) son sistemas de computación inspirados en las redes neuronales biológicas que constituyen los cerebros de los animales. Tales sistemas aprenden (mejoran progresivamente su rendimiento) para realizar tareas por medio de la consideración de ejemplos, sin requerir una programación que defina los pasos específicos para llevarlas a cabo. El objetivo de este informe es presentar el análisis introductorio realizado sobre distintas componentes esenciales en la creación de una red neuronal. Para ello se utilizaró la famosa base de datos MNIST de imágenes de números escritos a mano de 28x28 pixeles y la biblioteca de código abierto Tensorflow para procesar datos y desarrollar las redes neuronales.

2. Metodología

Las redes neuronales artificiales desarrolladas consisten de 3 capas, donde la capa de entrada posee una neurona por píxel (28x28) y la capa de salida 2 neuronas. Las capas están conectadas entre sí mediante sinapsis que conectan a cada neurona de la capa anterior con cada una de la capa posterior. Además, todas las neuronas poseen una función de activación sigmoidal logística, excepto la de salida que posee una softmax. Con respecto a las imágenes utilizadas, resultaron ser un subconjunto de la base de datos MNIST de 70000 imágenes, pues se acotó el problema a la detección del número 3 (número verificador del rut del alumno). Para ello fueron filtradas las imágenes de este número en la base de datos, junto con otras en igual cantidad elegidas de manera aleatoria y uniforme entre el resto de las imágenes. Este procedimiento fue realizado para el training, validación y testing, obteniendo un total de 11276, 986 y 2020 imágenes para cada etapa respectivamente. Como parámetros variables, fue empleado gradiente descendiente estocástico (SGD) con momentum, con

un minibatch de tamaño 32, entropía cruzada como función de costo, 25 neuronas en la capa escondida y una tasa de aprendizaje $\mu = 0.1$, los que se han analizado, discutido y/o adaptado a un valor más adecuado durante el informe. Por otro lado, se utilizó un criterio de detención "early stoping" con un máximo de 15 validaciones con error creciente consecutivas (a menos que se especifique un valor distinto). Finalmente ciertos criterios fueron adoptados para la consideración de los resultados. Los indicadores de testing fueron los que se consideraron concluyentes, debido a expresar las capacidades adquiridas de generalización de la red (a diferencia del training) y a poseer un mayor número de ejemplos que la validación. Adicionalmente, los promedios y desviaciones estándar de los indicadores fueron aproximados a la unidad, excepto que luego este criterio cambió al primer decimal, debido a la disminución en la variabilidad adquirida. A continuación se presentan los resultados obtenidos de los distintos experimentos realizados, luego un análisis de los mismos, y finalmente una discusión acerca de los algoritmos de aprendizaje de las redes neuronales utilizadas.

3. Resultados

3.1. CE vs MSE

	Exactitud	Precisión	Sensibilidad	Iteraciones	F1 score
CE	$96 \pm 2\,\%$	$99 \pm 1\,\%$	$93 \pm 5\,\%$	5386 ± 2215	$96 \pm 2\,\%$
MSE	$91 \pm 2\,\%$	$98 \pm 1\,\%$	$85 \pm 3\,\%$	2850 ± 611	$91 \pm 1\,\%$

Tabla 1: Comparando CE y MSE (testing)

3.2. Tasa de aprendizaje

	Exactitud	Precisión	Sensibilidad	Iteraciones	F1 score
10^{-2}	$90 \pm 4\,\%$	$97 \pm 3\,\%$	$85 \pm 7\,\%$	8502 ± 3462	$91 \pm 3\,\%$
10^{-1}	$96 \pm 2\,\%$	$99 \pm 1\,\%$	$93 \pm 5\,\%$	5386 ± 2215	$96 \pm 2\,\%$
10^{0}	$97 \pm 1\,\%$	$99 \pm 1\,\%$	$96 \pm 2\,\%$	1308 ± 165	$97 \pm 1\,\%$
10^{1}	$75 \pm 20\,\%$	$56 \pm 46\,\%$	$54 \pm 44\,\%$	386 ± 95	$55 \pm 45\,\%$

Tabla 2: Comparando tasas de aprendizaje (testing)

3.3. Neuronas en capa escondida

	Exactitud	Precisión	Sensibilidad	Iteraciones	F1 score
0	$92.3 \pm 0.9\,\%$	$90.5 \pm 4.6\,\%$	$94.0 \pm 2.6\,\%$	10504 ± 384	$92.1 \pm 1.2\,\%$
1	$94.1 \pm 0.2\,\%$	$96.7 \pm 0.7\,\%$	$91.9 \pm 0.9\,\%$	11538 ± 689	$94.2 \pm 0.2\,\%$
5	$96.5 \pm 0.3\,\%$	$95.8 \pm 1.2\,\%$	$97.1 \pm 0.6\,\%$	12300 ± 651	$96.4 \pm 0.3\,\%$
10	$97.9 \pm 0.2\,\%$	$98.6 \pm 0.2\,\%$	$97.3 \pm 0.4\,\%$	13084 ± 821	$97.9 \pm 0.2\,\%$
25	$98.5 \pm 0.1\,\%$	$98.7 \pm 0.5\,\%$	$98.4 \pm 0.3\,\%$	12340 ± 280	$98.5 \pm 0.1\,\%$
100	$98.4 \pm 0.1\,\%$	$98.7 \pm 0.2\,\%$	$98.2 \pm 0.3\,\%$	12280 ± 398	$98.4 \pm 0.1\,\%$

Tabla 3: Neuronas en capa escondida (con sobreajuste)

3.4. SGD vs ADAM

	Exactitud máxima (T)	Exactitud máxima (V)
SGD	$97.9\,\%$	$97.6\,\%$
ADAM	$98.3\,\%$	$96.6\,\%$

Tabla 4: Exactitudes máximas SGD vs ADAM (T: entrenamiento, V: validación)

	Exactitud	Precisión	Sensibilidad	F1 score	Iteraciones
SGD (T)	$97.6 \pm 0.4\,\%$	$99.4 \pm 0.2\,\%$	$96.0 \pm 0.9\,\%$	$97.6 \pm 0.4\,\%$	1246 ± 151
ADAM (T)	$95.7 \pm 2.5\,\%$	$98.3 \pm 1.8\,\%$	$93.6 \pm 4.7\,\%$	$95.8 \pm 3.4\,\%$	2978 ± 1157
SGD (V)	$97.4 \pm 0.1\,\%$	$98.9 \pm 0.4\,\%$	$96.1 \pm 0.5\,\%$	$97.5 \pm 0.1\,\%$	-
ADAM (V)	$95.3 \pm 1.8\,\%$	$97.6 \pm 1.1\,\%$	$93.3 \pm 2.9\,\%$	$95.4 \pm 1.7\,\%$	-
SGD (TE)	$97.2 \pm 0.4\,\%$	$99.1 \pm 0.8\,\%$	$95.6 \pm 1.3\,\%$	$97.3 \pm 0.4\,\%$	-
ADAM (TE)	$96.3 \pm 1.1\,\%$	$98.5 \pm 0.6\,\%$	$94.5 \pm 1.8\,\%$	$96.4 \pm 1.0\,\%$	-

Tabla 5: Indicadores SGD vs ADAM (T: entrenamiento, V: validación, TE: testing)

3.5. Sobreajuste

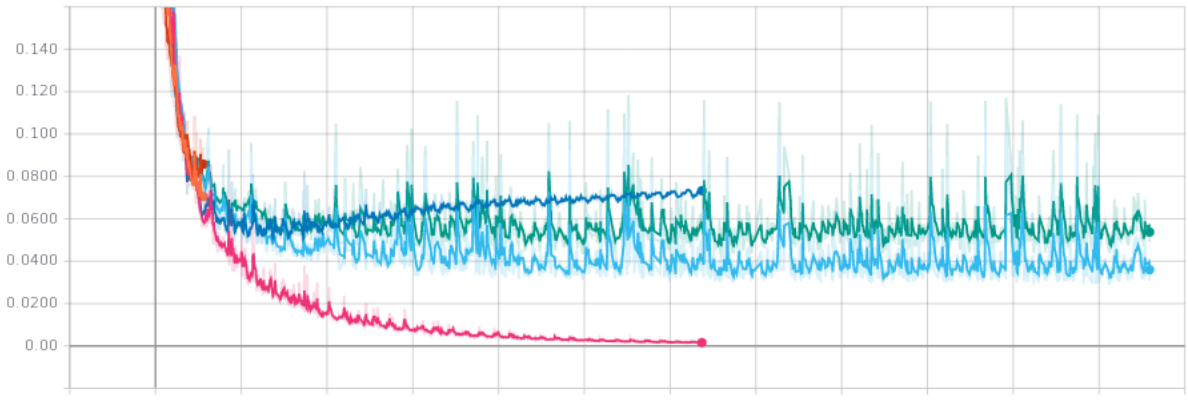

Figura 1: Error de entrenamiento y validación: sobreajustar (rojo,azul) y regularizar (celeste,verde)

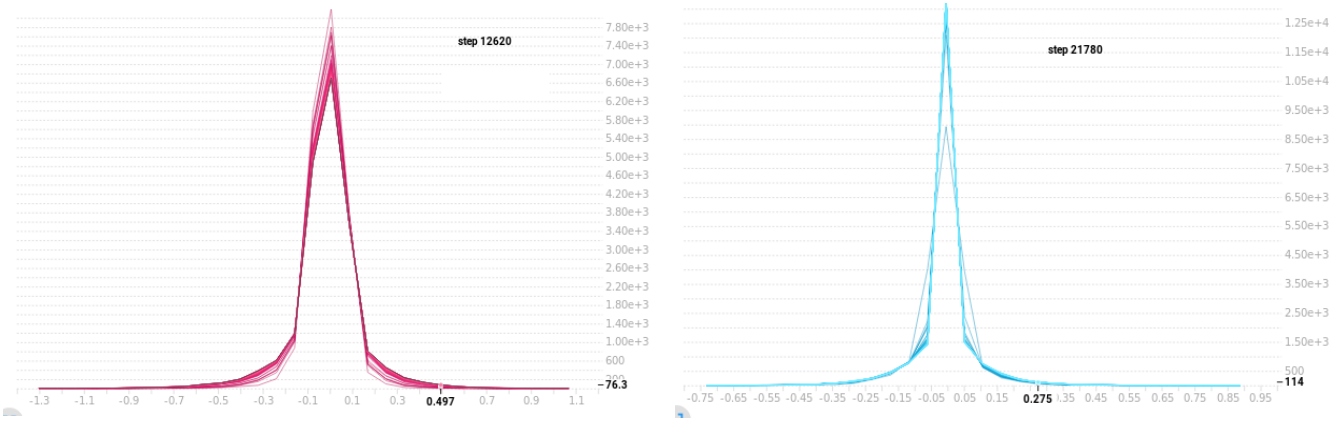

Figura 2: Comparación de pesos en red sobreajustada (rojos) vs regularizada (azules)

	Exactitud	F1 score
Sobreajustada	$98.2 \pm 0.1\,\%$	$98.2 \pm 0.1\,\%$
Regularizada	$97.9 \pm 0.4\,\%$	$97.9 \pm 0.3\,\%$

Tabla 6: Indicadores sobreajustados vs regularizados (testing)

4. Análisis de resultados

4.1. CE vs MSE

Las diferencias identificadas entre el uso de la entropía cruzada y el error cuadrático medio como funciones objetivo, fijando $\mu = 0.1$, SGD y 25 neuronas en la capa escondida se muestran en la Tabla 1. Como se puede observar, la entropía cruzada posee mejor exactitud, sensibilidad y F1 score, mientras que el error cuadrático medio, precisión e iteraciones requeridas. A pesar de que MSE requiere casi la mitad de iteraciones que CE, la exactitud y el F1 score son mayores, por lo que concluimos que CE es mejor función de costo para esta situación. Esto coincide con la teoría, pues ella explica que esto se debe a que estamos en un escenario de clasificación, en el que la derivada de MSE c.r.a w es pequeña cuando hay errores de clasificación, pues ésta no logra manejar adecuadamente las pequeñas derivadas de las funciones sigmoidales. Sin embargo, en la presencia de clasificaciones erróneas, CE recibe las pequeñas derivadas dentro de los logaritmos que la componen, generando errores grandes, exigiendo a la red aprender.

$$CE = -\frac{1}{n}\sum_x [y \ln \hat{y} + (1-y)\ln(1-\hat{y})]; \qquad MSE = \frac{1}{n}\sum_x (y-\hat{y})^2$$

(*Donde se tiene* $y, \hat{y}$ *distintos para cada patrón* x*, con* $\hat{y} = \frac{1}{1+e^{-w\cdot x}}$ $(= sigm(w^T x))$)

4.2. Tasa de aprendizaje

Se midió el desempeño promedio de 5 entrenamientos para distintas tasas de aprendizaje, utilizando SGD, CE, 25 neuronas en la capa escondida y un máximo de 1000 validaciones consecutivas con aumento de error, obteniendo los resultados de la Tabla 2. Los indicadores van mejorando hasta llegar a $\mu = 10$, en donde los promedios son inferiores y las desviaciones mayores. Cabe señalar que en dos de los entrenamientos, la última tasa generó inicialmente indicadores indefinidos, a causa de la inexistencia de verdaderos positivos y de falsos negativos. Estos fueron interpretados como nulos, obteniendo finalmente bajos indicadores para tal μ.

En base a estos resultados, $\mu = 1$ resulta ser la más atractiva de las tasas. 15 entrenamientos adicionales fueron realizados sin obtener señales de inestabilidad, por lo que se decidió utilizar esta tasa para los siguientes puntos. Esto concuerda con las tasas estáticas recomendadas: $\mu \in [10^{-4}, 10^{0}]$. Por otro lado, estos resultados sugieren que la forma subyacente de la función de costo es amplia

para todas las entradas, es decir, la variación del error para pequeños cambios de los pesos no es grande, por lo que una tasa de aprendizaje "elevada" ($\mu = 1$) sería adecuada.

4.3. Neuronas en capa escondida

Utilizando CE, SGD y $\mu = 1$, se obtuvieron los resultados de la Tabla 3, al variar el número de neuronas en la capa escondida. Desde 0 a 25 neuronas, se observa un crecimiento en todos los indicadores (solo con algunas excepciones dentro del margen de error), manteniéndose la regla de que al considerar la exactitud y F1, cada red sucesora es superior a la anterior. Sin embargo, entre 25 y 100, las similitudes superan el criterio combinado de exactitud y F1, no pudiendo establecerse superioridad alguna en ninguno de estos dos indicadores.

Por otro lado, la red de 10 neuronas vs la de 25 (en capa escondida) es apenas inferior en estos dos indicadores. Por ello se repitió el experimento para estas dos redes, obteniéndose resultados muy similares y reafirmando la conclusión de que la de 25 es estadísticamente superior. Esto al superar fuera del margen de error a la de 10 neuronas en la capa escondida, con lo que concluimos que el número inicialmente elegido es el adecuado y el que se utilizará de aquí en adelante.

4.4. SGD vs ADAM

Utilizando CE, $\mu = 1$ y 25 neuronas en la capa escondida, se midió el rendimiento del gradiente descendiente estocástico vs el algoritmo de la estimación de momentos adaptativa (ADAM), obteniendo los resultados de la Tabla 4. Observando los valores máximos de exactitud, ADAM supera a SGD en entrenamiento, pero no en validación. Esto nos lleva a preguntarnos si ADAM podría superar a SGD en otros aspectos. Luego, observando los indicadores agregados de entrenamiento en la Tabla 5, nos percatamos de que en realidad, SGD supera en promedio a todos los indicadores de ADAM, a pesar de que esto (debido a la gran dispersión de los resultados de ADAM) solo suceda si no es considerado el margen de error. Sin embargo, en el caso de la validación, SGD sí supera a ADAM en exactitud y en F1.

Finalmente, recurriendo a los datos de testing para encontrar una confirmación, nos percatamos de que ADAM solo logra ser superado en el número de iteraciones requeridas, si consideramos su margen de error. Por ello, no podemos afirmar en el presente experimento y con los datos utilizados que ADAM sea inferior a SGD, solo conjeturar que ADAM generalizaría peor que SGD. Esto puede

deberse a que utilizamos Adam con los parámetros por defecto (posiblemente inadecuados para este problema), y a que Adam está diseñado para problemas de más difícil tratamiento, que involucren varias dimensiones con gradientes dispersos, como lo es el reconocimiento de lenguaje natural (más de esto en discusión).

4.5. Sobreajuste

Utilizando SGD, CE, $\mu = 1$ y un límite de 1000 validaciones consecutivas con aumento de error, se obtuvieron las curvas de error de la Figura 1. En el sobreajuste, se puede observar cómo la curva azul de validación se separa de la roja de entrenamiento, donde el entrenamiento continúa disminuyendo su error, mientras que el de la validación comienza a aumentar. Luego, mediante la regularización, la validación logra obtener un menor error "transfiriendo" parte de él al entrenamiento. Además, la curva regularizada tiende a seguir por más tiempo sin alcanzar el límite de las 1000 validaciones consecutivas. Por otro lado, el sobreajuste se caracteriza por generar una mayor proporción de pesos de mayor valor absoluto, lo que de una forma más estrecha a los histogramas de pesos de las redes sobreajustadas. En la Figura 2 se puede apreciar que ambas redes poseen características de sobreajuste, pero que la aparición de pesos de "gran" valor absoluto es más infrecuente en la regularizada. Estos resultados concuerdan con el esperado funcionamiento de los regularizadores, los agregan costo a la función de consto según la norma l2 (euclidiana) de los pesos, por lo que éstos buscarán tener componentes de bajo valor absoluto. Esto evitaría que se consideren ciertas características de alto detalle existentes en los datasets de entrenamiento, enfocándose en patrones más generales y comunes al problema.

Finalmente, desde un punto de vista de los indicadores de los dos tipos de redes generadas con los hiperparámetros utilizados, las redes regularizadas no logran desempeñarse mejor (Tabla 6), pues ni su exactitud ni su F1 score logran ser significativamente mejores. De hecho, sin considerar el margen de error, tendría indicadores inferiores. Desde luego, no podemos afirmar su inferioridad al considerar los márgenes de error obtenidos.

5. Discusión

5.1. Gradiente descendiente con minibatch

En el desarrollo de redes neuronales, distintas implementaciones han sido desarrolladas, y en base a ellas, nuevas y con mejores características prácticas han sido descubiertas. El algoritmo del gradiente descendiente con batch consiste en realizar la actualización de los pesos, calculando el gradiente de una función de costo utilizando la totalidad del set de entrenamiento, mientras que el algoritmo de gradiente descendente estocástico calcula el gradiente de la función en base a únicamente el ejemplo actual del set de entrenamiento. El primero realiza computaciones redundantes para datasets grandes, calculando varias veces los gradientes de ejemplos similares y tardando considerablemente por cada iteración (y siendo intratable para datasets que no caben en memoria), no puede funcionar de forma online, y tiende a quedarse estancado en mínimos locales. Por otro lado, el segundo sufre grandes variaciones debido a la varianza que enfrentan los gradientes estimados, complicando más de la cuenta la llegada al máximo global y requiriendo de una tasa de aprendizaje menor para asegurar una adecuada convergencia.

Por otro lado, el algoritmo del gradiente descendiente con minibatch combina las ventajas proporcionadas de ambos mundos, reduciendo la varianza de las actualizaciones de los pesos, lo que lleva a convergencias más estables, y puede utilizar operaciones de matrices altamente optimizadas, comunes en las bibliotecas de deep learning en el estado del arte. El tamaño del minibatch deberá ser elegido teniendo en cuenta el tradeoff entre flexibilidad de escape frente a los mínimos locales (consideración de valores de ejemplos particulares, lo que es entregado por batches pequeños), y una varianza que no impida una estable convergencia. Los tamaños típicos varian entre 50 y 256, pero pueden variar para distintas aplicaciones.

5.2. Optimizaciones

SGD (con minibatch) no garantiza buena convergencia, sino que ofrece distintos desafíos que deben ser considerados, entre ellos el problema asociado a navegar superficies que se curvan de forma mucho más empinada en unas dimensiones que en otras, las que son comunes cerca de mínimos locales, que generan oscilaciones en el camino hacia el mínimo global entre un mínimo local y otro, y que no logran ser superadas efectivamente realizando solo una elección pequeña del minibatch. Esto ha generado propuestas para mejorar el algoritmo SGD, y una de ellas es el

método del momentum, el que recuerda las actualizaciones previas de los pesos (en una media móvil), y determina la siguiente iteración como una combinación lineal convexa del gradiente y su actualización previa:

$$\Delta w(k+1) = \mu(-\nabla_w C(k)) + \alpha \Delta w(k)$$

(C *función de costo y* $\alpha \in [0,1)$*, típicamente 0.9*)

Su objetivo es lograr acelerar la convergencia de los pesos, superando más rápidamente las oscilaciones. Esto lo logra asignando de forma segura un μ pequeño, pero (como insinúa su fórmula de actualización) aumentando el ritmo de cambio de las componentes que llevan un historial de cambio previo, y disminuyendo el de las que recién han invertido su signo de cambio (de ahí el nombre momentum). Sin embargo, tal aceleración es ciega y no es consciente de cuando debe detenerse, incluso si se está acercando al objetivo, sino que desacelera una vez que percibe que se ha alejado de la dirección correcta (e invierte su signo de cambio). El algoritmo del gradiente acelerado de Nesterov (NAG) aproxima el lugar donde se encontrarán los pesos en la siguiente iteración, y luego tomar eso en cuenta para poder desacelerar antes de que sea costoso. Por otro lado, es común la existencia de gradientes dispersos, los que generan la necesidad de que las distintas componentes deban cambiar a un ritmo distinto a las demás para poder así adaptarse a esos cambios locales infrecuentes.

Adam (estimación adaptativa de momentos) es una adaptación de SGD que combina las capacidades de tratar con gradientes dispersos y objetivos no estacionarios, mediante el uso de tasas de aprendizaje adaptativas individuales basadas en estimaciones del primer y segundo momentos de los gradientes (de ahí su nombre):

$$m_w(k+1) = \beta_1 m_w(k) + (1-\beta_1)\nabla_w C(k), \quad m_w(0) = 0$$

$$v_w(k+1) = \beta_2 v_w(k) + (1-\beta_2)(\nabla_w C(k))^2, \quad v_w(0) = 0$$

$$\hat{m}_w(k+1) = \frac{m_w(k+1)}{1-\beta_1^k}, \quad \hat{v}_w(k+1) = \frac{v_w(k+1)}{1-\beta_2^k}$$

$$\Delta w = \mu \frac{-\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

($\beta_i \in [0,1)$*,* $\epsilon > 0$*, para evitar división nula, y* $()^2$ *dado por el producto de Hadamad.*
Los autores proponen $\beta_1 = 0.9,\ \beta_2 = 0.999,$ *y* $\epsilon = 10^{-8}$)

Las ventajas de este método son que la magnitud de cambio de los pesos se ajusta automáticamente, es invariante a escalamientos del gradiente y está aproximadamente limitada por la tasa de aprendizaje como hiperparámetro, que no requiere un objetivo estacionario y que funciona en presencia de gradientes dispersos. Como puede apreciarse en las fórmulas, Adam no solo utiliza una media exponencialmente descendente de los pasados gradientes cuadrados ($v_w(t)$'s), sino que también una media exponencialmente descendente de los gradientes pasados ($m_w(t)$'s), similar a SGD con momentum. Sin embargo, nuevamente Adam posee la desventaja de SGD con momentum, en la que su media exponencialmente decreciente de los gradientes pasados no considera la posición de los pesos dentro de el espacio en la futura iteración, y no desacelera sin haber generado costos. Por ello NAG puede ser incorporado a Adam, generando el algoritmo NADAM, el que también estima la posición en que estarán los pesos, evitando así el sobrecosto generado por aceleración sin control.

6. Conclusiones

SGD con momentum ha resultado ser una poderosa herramienta en el problema de reconocimiento de dígitos escritos a mano, alcanzando exactitudes de 97.2 ± 0.4 y F1 score de 97.3 ± 0.4, utilizando CE, $\mu = 1$, 25 neuronas en capa escondida, un minibatch de tamaño 32 y un máximo 15 validaciones consecutivas (cada 10 iteraciones) con error ascendente. Sin embargo, las exactitudes en el estado del arte en la clasificación de MNIST rondan los $99.5 \pm 0.3\,\%$, por lo que probablemente los desafíos y optimizaciones asociados a esta red no hayan terminado. Una adaptación a NAG podría ser interesante.

Por otro lado, no hemos visto el potencial de algoritmos como Adam para enfrentar problemas con gradientes dispersos que requieran tasas de aprendizaje adaptativas para funcionar, o unos en los que solo una mayor cantidad de capas serían requeridas, como procesamiento de lenguaje natural. Queda mucho por aprender.

7. Referencias

- Cross entropy: http://neuralnetworksanddeeplearning.com/chap3.html
- Paper Adam: https://arxiv.org/pdf/1412.6980.pdf
- Minibatch & optimizaciones SGD: http://ruder.io/optimizing-gradient-descent/