



**Tecnológico
de Monterrey**

Proyecto Final: Deep Learning Neural Network Application.

Deep Learning EM 2019 GPO 1

Profesor: Dr. Leonardo Garrido.

Roberto Pérez Barrera. A01380452

Guillermo J. Astiazarán. A00226365

15 de mayo del 2019

Campus Monterrey

Introducción

Hoy en día las tecnologías de la información forman una gran parte de nuestra vida cotidiana, pero la Inteligencia Artificial es algo que paso a paso esta agarrando fuerza no solo en el sector doméstico, si no en la industria en general.

Se han desarrollado distintas ramas como lo son algoritmos de búsqueda, sistemas multiagentes, aprendizaje automático, etc. Pero lo más reciente es el Aprendizaje Profundo. Un campo de investigación donde le da la oportunidad a los desarrolladores a imitar las funcionalidades de un cerebro humano. Modificando pesos entre las distintas capas conformadas por neuronas para al final arrojar un resultado de predicción.

En este proyecto nos enfocamos en la descripción de imágenes, específicamente en los gestos de las personas. El sistema recibe un dataset de 600 imágenes protagonizadas por personas sonriendo y no sonriendo. Al final el software realiza un auto entrenamiento para predecir cualquier tipo de imagen que contenga el rostro de una persona y arroja el resultado de su precisión.

Data Set

Fuente:

<https://www.kaggle.com/iarunava/happy-house-dataset/>

El Dataset que seleccionamos fue “Happy House Dataset”. Contiene un training set con 600 ejemplos y un test set de 150 imagenes.

El dataset que seleccionamos cuenta con imágenes de personas sonriendo y no sonriendo. Nuestra labor en este proyecto es ajustar los hiperparametros de la red neuronal de 2 capas y de L capas con la finalidad de encontrar la mejor exactitud al momento de clasificar la imagen.

Decidimos usar este dataset ya que es poco pesado (dado que al igual que el dataset de gatos, cuenta con imágenes de 64x64 pixeles) y es perfecto para demostrar la variación que puede ver en la exactitud de la clasificación dependiendo de los valores de los parámetros.

Metodología

Nuestra metodología para determinar la mejor exactitud, será mediante la modificación sistemática de cada hiper parámetro individualmente, para ver cómo se va comportando en ambas redes. Para esto modificaremos lo siguiente:

Learning rate:

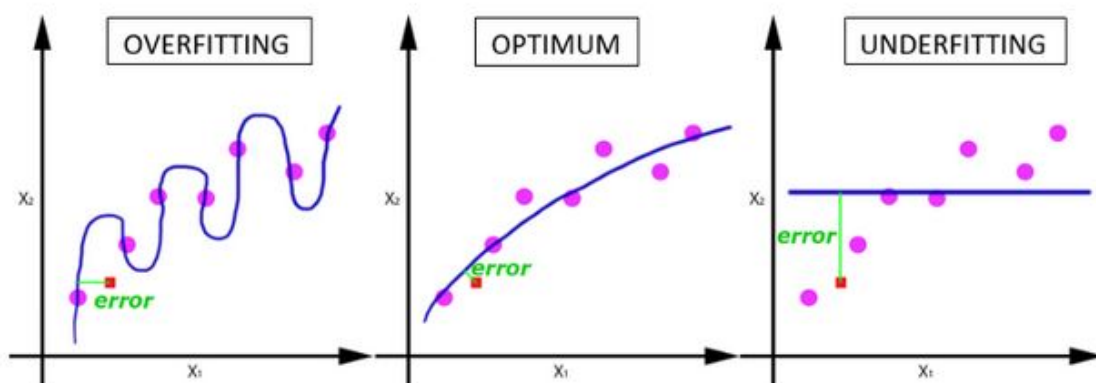
Originalmente en 0.0075 iremos variando el valor partiendo desde los extremos, probando con un valor grande (0.01) y después con otros valores exponencialmente más pequeños (0.01, 0.001 y 0.0001) para ver cómo se comporta en esos casos extremos. Nuestra metodología se basará en empezar con un valor pequeño e irlo incrementando exponencialmente por cada lote

Sabemos que debemos prestar particular atención al momento de decidir el mejor valor, ya que si este es muy pequeño, el descenso por gradiente será mucho más lento, en cambio si es muy grande puede sobrepasar el mínimo. Y puede fallar al converger.

Número de Iteraciones:

Por otro lado sabemos que elegir el número de iteraciones tampoco es sencillo ya que no hay un número de épocas predeterminadas que se deban hacer en base al tamaño del dataset. Por lo general suele depender de muchos más factores.

Es importante elegir bien el número de épocas ya que conforme el número de iteraciones va creciendo, por ende se cambia más el número de veces que se modifica el peso en la red neuronal y esta curva puede ir desde un ajuste insuficiente (underfitting), el óptimo que buscamos y si tenemos un ajuste excesivo podemos caer en overfitting.



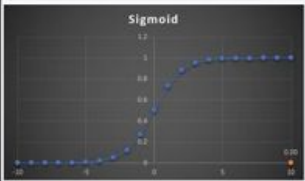
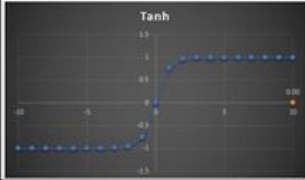
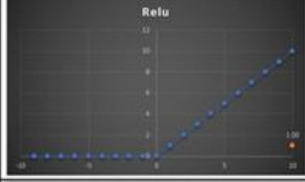
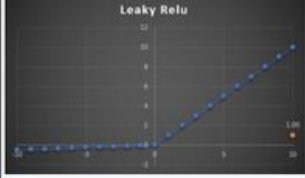
Número de capas:

Tal como se hizo en la tarea 9, nosotros probaremos la modificación de los parámetros antes mencionados en una red neuronal de dos capas y en otra de 4 capas.

Una vez que hayamos obtenido los resultados de estas dos redes, determinaremos en base al valor obtenido del accuracy si es prudente o no, seguir agregando más capas a la red.

Funcion de activacion:

Al igual que el número de capas probaremos con las funciones de activación predeterminadas en la tarea. Tanto la sigmoid como la relu. Sin embargo también procederemos a probar con otra función de activación (leaky relu).

| Name | Plot | Equation | Derivative |
|--|---|--|--|
| Sigmoid |  | $f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ | $f'(x) = f(x)(1 - f(x))$ |
| Tanh |  | $f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ | $f'(x) = 1 - f(x)^2$ |
| Rectified Linear Unit (relu) |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Leaky Rectified Linear Unit (Leaky relu) |  | $f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |

De antemano sabemos que probaremos con relu y leaky relu para las capas escondidas. Hemos decidido no probar con sigmoid y tanh en estas capas dado que durante la investigación encontramos que generalmente no suelen ser muy efectivas para estas capas porque si “z” es muy pequeño o muy grande, la pendiente suele ser muy chica lo cual alenta el proceso de descenso por gradiente.

Por otro lado, para la capa de salida probaremos sólo con sigmoid, ya que en nuestro dataset la salida del valor real es de 0 (no está sonriendo) o 1 (está sonriendo), así que particularmente la función sigmoid nos parece la más viable dado que la predicción también fluctuará en un valor entre 0 y 1.

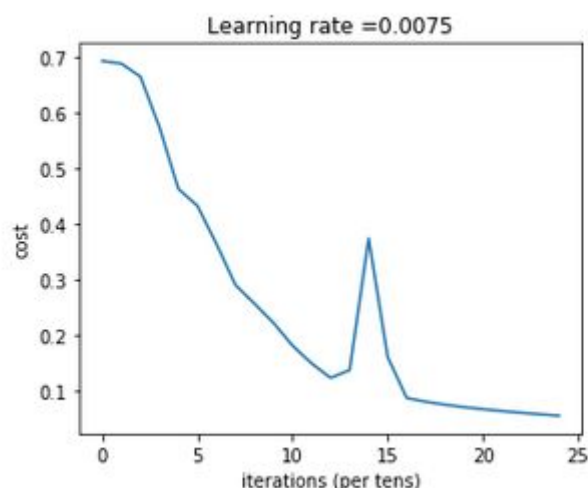
Resultados de la metodología

Primeramente mostraremos los resultados preliminares conseguidos con las redes (2-layer model y 4-layer model sin modificación alguna en los parámetros, tal cual las trabajamos en la tarea 9).

Resultados con la red de 2 capas original:

El resultado que obtuvimos fue sorprendentemente bueno, sabíamos que dada la naturaleza de nuestro dataset, la clasificación inicial de imagenes iba a ser buena, pero los resultados superaron nuestras expectativas. Aquí los resultados del de las train set y el test set:

Cost after iteration 2400: 0.05515685144734092



```
In [38]: predictions_train = predict(train_x, train_y, parameters)
```

```
Accuracy: 0.9883333333333334
```

Expected Output:

```
**Accuracy** 1.0
```

```
In [39]: predictions_test = predict(test_x, test_y, parameters)
```

```
Accuracy: 0.94
```

Expected Output:

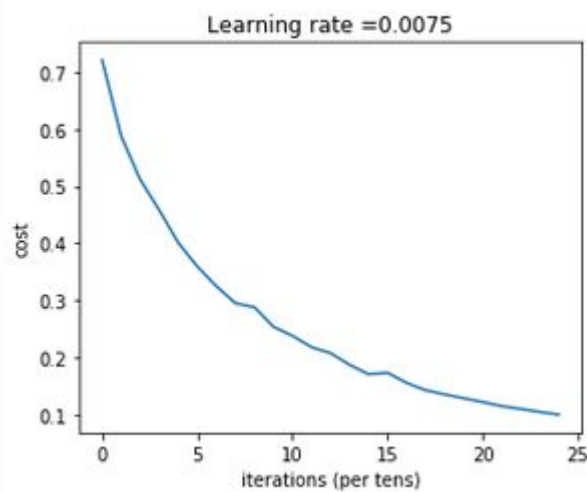
```
**Accuracy** 0.72
```

Como podemos observar en las imágenes, con los parámetros iniciales conseguimos muy buenos resultados. El costo bajó hasta un 0.05515 (aunque por algún motivo hay un crecimiento exponencial en determinado punto) lo cual es bastante razonable y el accuracy se quedó en 0.98 lo cual tampoco es malo, pero lo que más nos sorprendió fue el resultado del test set con una exactitud del 0.94. Sin duda alguna es un excelente resultado, sin embargo más adelante modificaremos los parámetros para ver si podemos mejorar aún más la exactitud de la clasificación.

Resultado de la red con 4 capas original:

Con la red de 4 capas obtuvimos un resultado impresionantemente bueno. Ambas precisiones aumentaron (un 1% y 2% respectivamente) lo cual a esos niveles en el dataset es bastante bueno. El trainset quedo con una exactitud de un 0.9933 y el test set con una impresionante exactitud del 0.96. Aquí las imágenes de comprobación.

Cost after iteration 2400: 0.099769



```
In [44]: pred_train = predict(train_x, train_y, parameters)
```

Accuracy: 0.9933333333333334

****Train Accuracy**** 0.985645933014

```
In [45]: pred_test = predict(test_x, test_y, parameters)
```

Accuracy: 0.96

Expected Output:

****Test Accuracy**** 0.8

Sin lugar a duda son excelentes resultados, sin embargo vamos a modificar los hiper parámetros para ver si logramos conseguir una exactitud aun mejor.

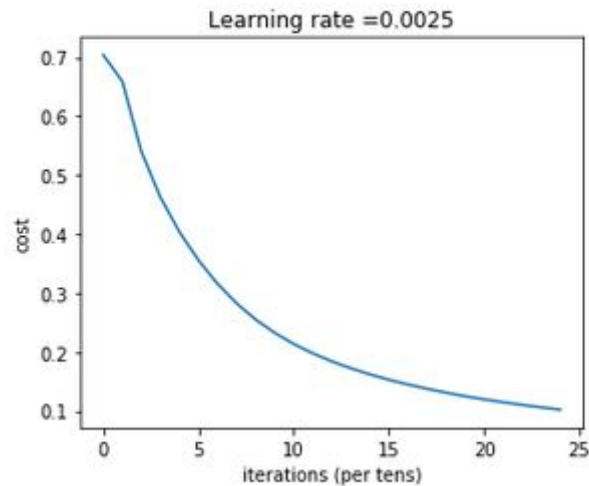
Modificando los hiper-parámetros:

Dados los números de las redes originales, decidimos trabajar con la red de 4 capas, porque evidentemente obtuvo un mejor resultado en general.

Modificando el learning rate:

Comenzamos probando con learning rate bastante bajo (0.0025) y estos fueron los resultados:

Cost after iteration 2400: 0.102048



```
In [71]: pred_train = predict(train_x, train_y, parameters)
```

Accuracy: 0.9833333333333334

****Train Accuracy**** 0.985645933014

```
In [72]: pred_test = predict(test_x, test_y, parameters)
```

Accuracy: 0.9533333333333334

Expected Output:

****Test Accuracy**** 0.8

los resultados son bastante buenos pero no superan los iniciales y el tiempo de ejecución fue un poco más lento. Sin embargo, son excelentes valores de Accuracy con los cuales comenzar.

Después de probar con los siguientes valores:

- 0.0025 (Train Accuracy: 0.98, Test Accuracy: 0.9533)
- 0.0055 (Train Accuracy: 0.9866, Test Accuracy: 0.9534)
- 0.0075 (Train Accuracy: 0.9933, Test Accuracy: 0.96)
- 0.009 (Train Accuracy: 0.9833, Test Accuracy: 0.9333)
- 0.09 (Train Accuracy: 0.5, Test Accuracy: 0.43)

Encontramos que el que mejor resultado nos daba era el de 0.0075 con una excelente efectividad. Decidimos dejar ese valor (el original) ya que era el mejor también para nuestro dataset ya que con valores cercanos por debajo y superiores se perdía un poco de exactitud en los resultados.

Modificando el número de iteraciones:

Probamos con las siguientes iteraciones y estos fueron sus respectivos resultados: (Todas probadas con el learning rate conseguido (0.0075))

- 1500 (Train Accuracy: 0.9566, Test Accuracy: 0.9266)
- 2000 (Train Accuracy: 0.98, Test Accuracy: 0.94)
- 2500 (Train Accuracy: 0.9933, Test Accuracy: 0.96)
- 2700 (Train Accuracy: 0.9933, Test Accuracy: 0.9733)
- 3000 (Train Accuracy: 0.9850, Test Accuracy: 0.9333)
- 3500 (Train Accuracy: 0.9850, Test Accuracy: 0.9333)

Como podemos observar en los resultados, después de analizar con diferentes cantidades de interacciones llegamos a la conclusión de que el mejor rango se encontraba entre las 2600 y 2700 iteraciones. (con 2800 comenzaba a bajar y en las 3000 iteraciones ya estábamos en un 0.9333)

Con este cambio logramos aumentar incluso un poco más el porcentaje de accuracy en el test set, llegando hasta el 0.9733, lo cual consideramos está bastante bien.

Modificando el número de capas:

Resultado con 5 capas:

Agregando una capa más al modelo original y agregando más neuronas no se obtuvo buen resultado, bajo la precisión del resultado. Este sería el extremo de mayor cantidad de capas y neuronas.

```
In [29]: pred_train = predict(train_x, train_y, parameters)
        Accuracy: 0.9233333333333333

                                                **Train Accuracy** 0.985645933014

In [30]: pred_test = predict(test_x, test_y, parameters)
        Accuracy: 0.8600000000000001
```

Resultado con 3 capas:

Utilizando el modelo más chico de L-Capas (3 en este caso), tenemos un mejor resultado que con 5 capas obteniendo un test accuracy de 93%, aunque no superamos el modelo original de 96% ni del 97% que hasta ahora tenemos con el modelo de 4 capas.

```
In [34]: pred_train = predict(train_x, train_y, parameters)
        Accuracy: 0.9933333333333334

                                                **Train Accuracy** 0.985645933014

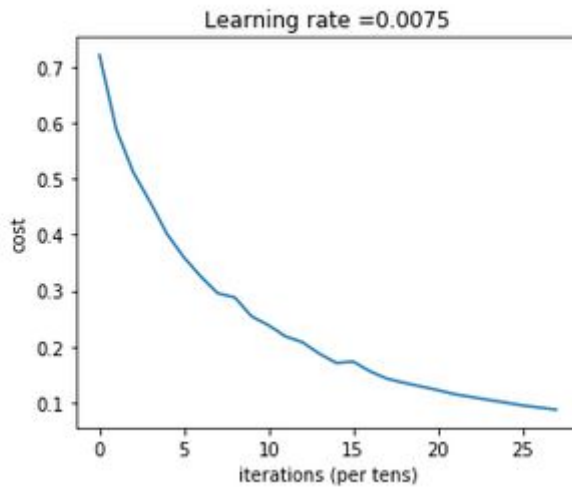
In [35]: pred_test = predict(test_x, test_y, parameters)
        Accuracy: 0.9333333333333333
```

Modificando la función de activación por capa:

En cuanto a las funciones de activación, inicialmente teníamos pensado intentar usar leaky relu para ver si podíamos mejorar el modelo, sin embargo para este punto la red funciona muy bien y cumple casi a la perfección con la tarea de clasificación (97% en su mejor configuración). Por este motivo es que hemos decidido continuar usando las función de relu para la capas escondidas, ya que cumple de manera muy eficiente para nuestro dataset.

Resultados del Test Set:

Cost after iteration 2700: 0.087034



```
In [77]: pred_train = predict(train_x, train_y, parameters)
```

Accuracy: 0.9933333333333334

****Train Accuracy**** 0.985645933014

```
In [78]: pred_test = predict(test_x, test_y, parameters)
```

Accuracy: 0.9733333333333334

Expected Output:

****Test Accuracy**** 0.8

Después de estar probando con muchos diferentes parámetros llegamos a la conclusión de que la red que mejor funciona para nuestro test set es la red con:

- Learning rate : 0.0075
- Número de iteraciones: 2700
- Número de neuronas: [12288, 20, 7,5, 1]
- Red de 4 capas
- Funciones de activación: ReLu (hidden layers) y Sigmoid (output)

Como podemos observar en las capturas de pantalla el resultado con la configuración arriba descrita funciona de manera muy correcta, llegando a un accuracy de un 0.9733 lo cual para términos el proyectos consideramos que esta muy bien. Está muy por encima del 0.8 se se consiguió en la tarea 9, con el dataset de clasificación de gatos.

Análisis de resultados erróneos:



Algunos de los motivos por los cuales se obtuvieron clasificaciones erróneas son los siguientes:

- La persona sonríe de manera inusual (boca muy abierta)
- El ángulo de la foto no es el correcto (de frente)
- La foto esta muy oscura
- La postura no es muy natural

Sin embargo en general se obtuvo un excelente resultado, ya que fueron muy pocas las imágenes que se clasificaron de manera incorrecta

Conclusiones:

En conclusión consideramos que el trabajo de entrenar una red neuronal es toda una ciencia y un arte ya que cada parámetro debe ser seleccionado de una manera muy precisa y así poder incrementar la efectividad de la red. Sin lugar en el desarrollo de este proyecto aprendimos bastante sobre como adaptar una red a una necesidad específica. En nuestro caso en particular nos ayudó bastante que la red que teníamos anteriormente estaba ya muy bien configurada para nuestro dataset, ya que realmente es muy parecido el train y test set que usamos con los que originalmente venían.

Consideramos de igual manera muy interesante el hecho de tener que “encajar las piezas” para que al final se clasifiquen las instancias de la mejor manera.

Consideramos realmente interesante todo ese proceso de saber que funciones de activaciones usar, como ir aumentando el learning rate, ir probando con diferentes iteraciones, capas, etc. En general es un proceso muy entretenido pero creemos que puede llegar también a ser algo tedioso para redes muy complejas.

Referencias:

<https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0>

<https://www.kaggle.com/iarunava/happy-house-dataset/>

<https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>

<https://engmrk.com/activation-function-for-dnn/>