Machine learning

Elaborado por: Deep Squad

2019

¿Que es Machine Learning?

Machine Learning es el estudio de los algoritmos de computadora que permitirán a los programas de computadora mejorar automáticamente con la experiencia.

Se dice que un programa de computadora aprende de la experiencia E con respecto a una tarea T y una medida de rendimiento P, si su rendimiento en T, medido por P, incrementa con la experiencia E.

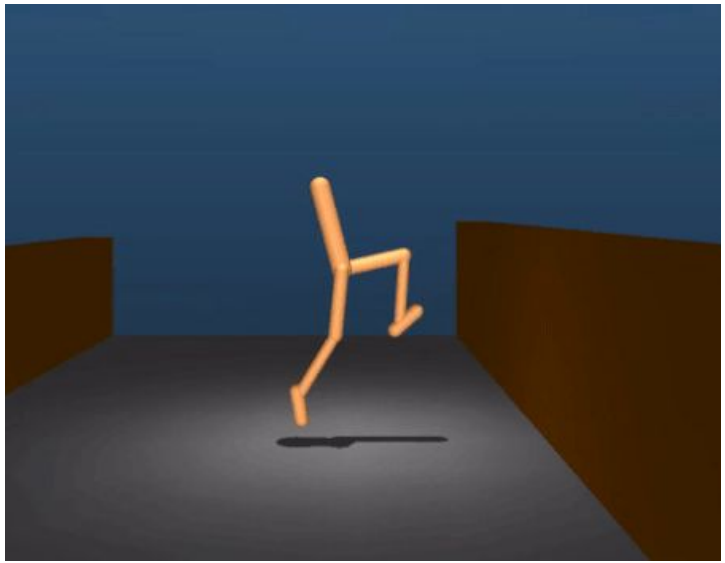


Figura 1. Ejemplo de machine learning

Supervised learning

- Le enseñaremos a la computadora a realizar cierta tarea.
- Un algoritmo de Supervised learning infiere una función a partir de datos de entrenamiento.
- Esta función inferida podrá ser utilizada para nuevos datos.

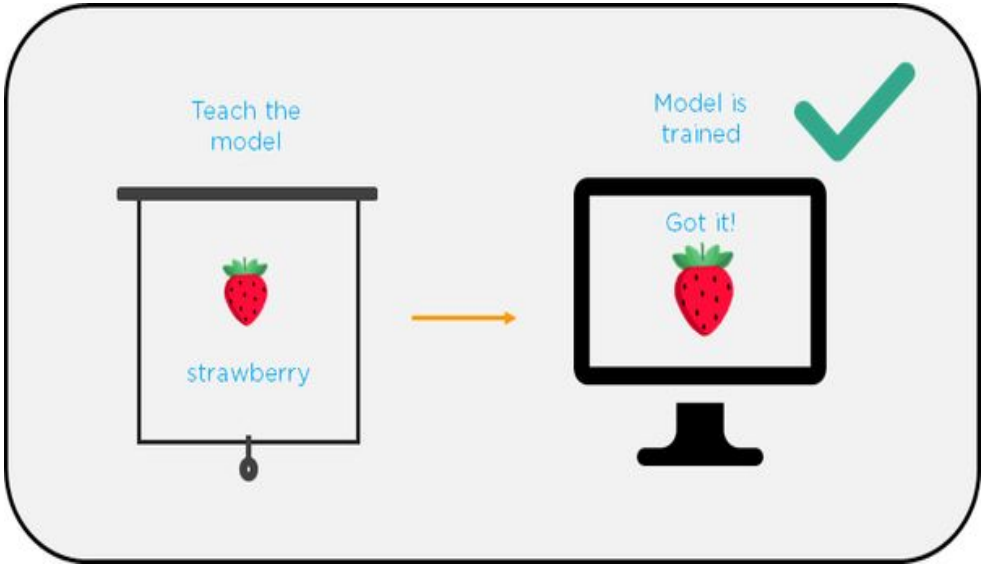


Figura 2. Entrenamiento de un modelo de machine learning

Supervised learning

Classification

Este tipo de algoritmo tiene el objetivo de **predecir** a qué clase pertenece un conjunto de datos. En otras palabras, predice un **valor discreto**.

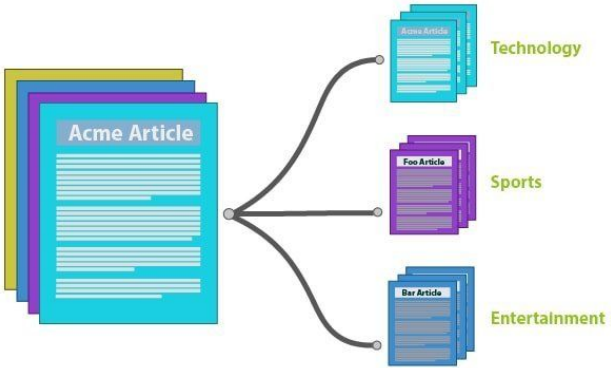


Figura 3. Clasificación discreta de datos

Regression

Este tipo de algoritmo tiene como objetivo **predecir** un **valor continuo**(2, 3.1416, 4.5, etc).

Figura 4. Estimación continua de datos

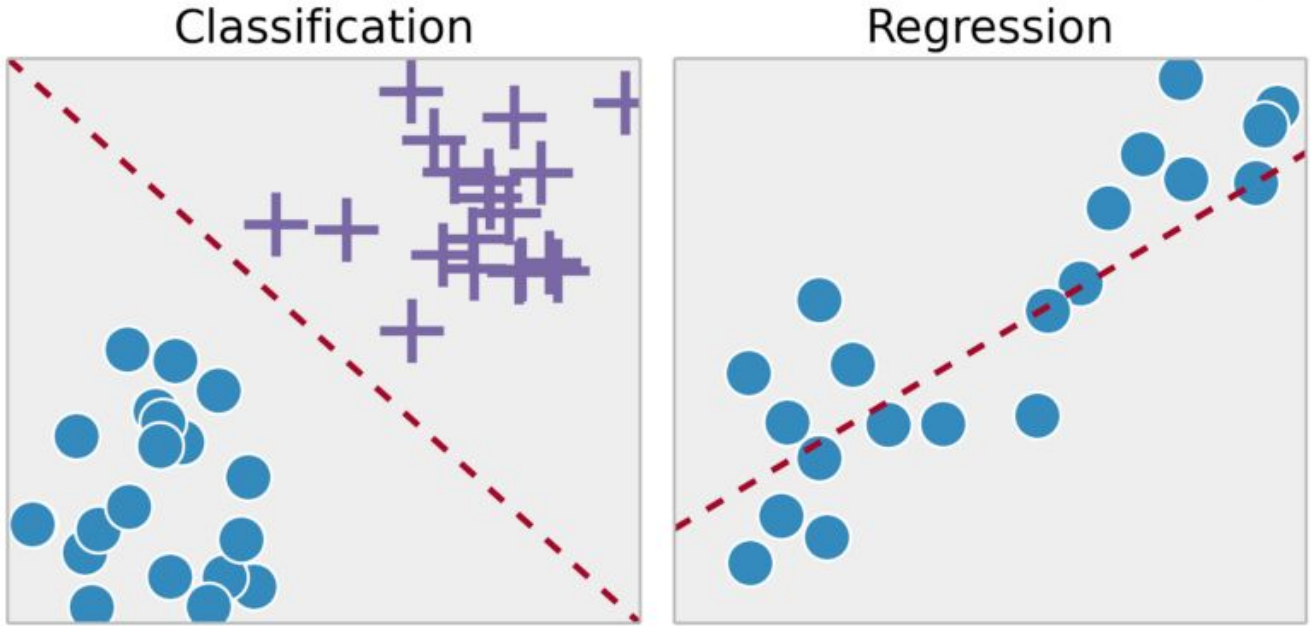


Figura 5. Representación de la clasificación y regresión.

Unsupervised learning

- En Supervised learning nosotros entrenamos al algoritmo dándole las “respuestas correctas”.
- En Unsupervised learning tenemos datos que no tienen etiquetas.
- La computadora va a aprender por sí sola.
- En Unsupervised learning se trata de medir como los datos pueden estar relacionados para agruparlos en “clusters”.

Unsupervised learning

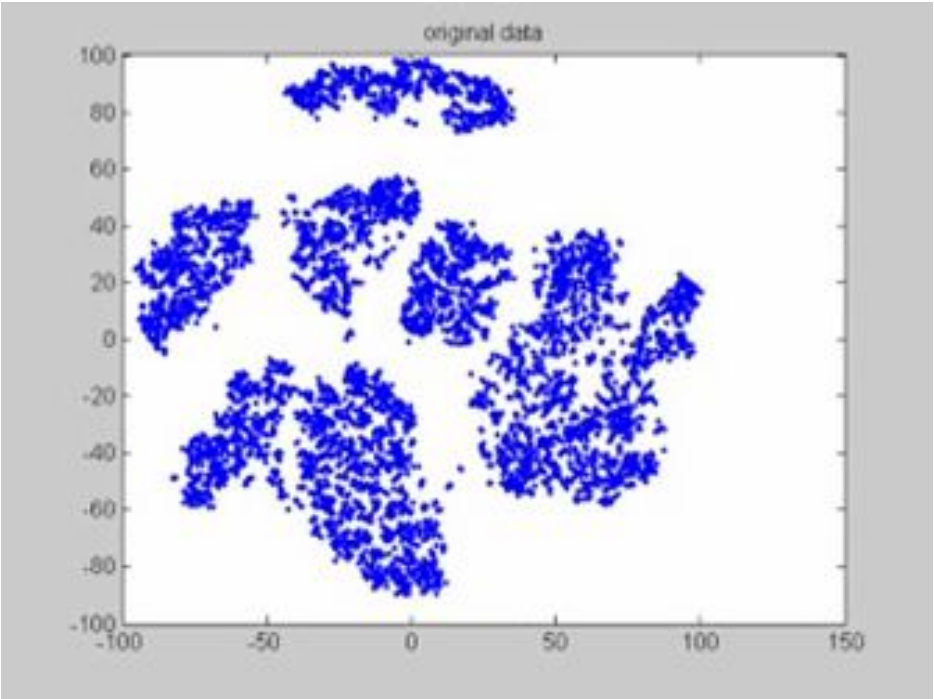


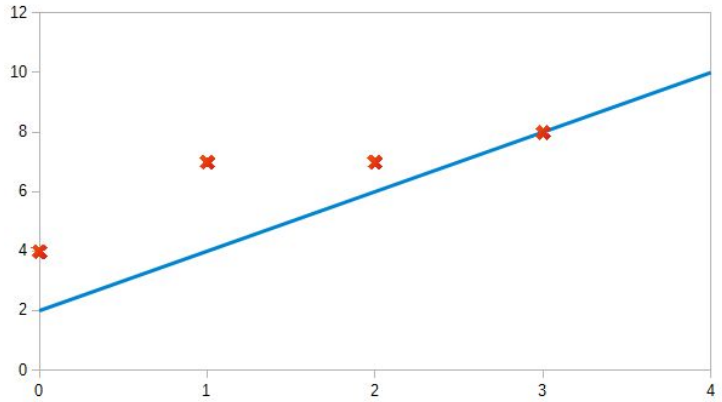
Figura 6. Representación de un modelo no supervisado.

Linear Regression

Cost Function

Si tenemos la siguiente hipótesis

$$h_\theta(x) = \theta_0 + \theta_1 x$$

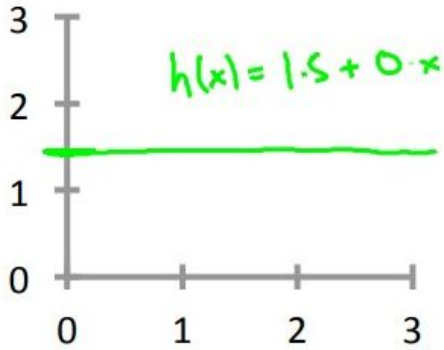


¿Cómo encontramos los parámetros de ella?

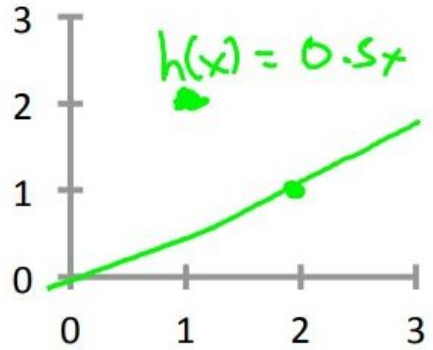
Objetivo: Elegir a los parámetros tal que nuestra función hipótesis se ajuste lo mejor a nuestros datos de entrenamiento.

Cost Function

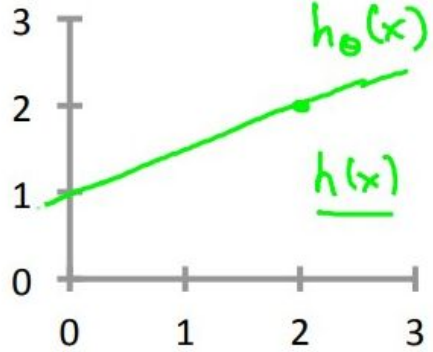
$h_\theta(x) = \theta_0 + \theta_1 x$



→ $\theta_0 = 1.5$
→ $\theta_1 = 0$



→ $\theta_0 = 0$
→ $\theta_1 = 0.5$



→ $\theta_0 = 1$
→ $\theta_1 = 0.5$

Cost Function

Tenemos a la **función de costo**, que nos permite ver cual es el error promedio entre nuestra predicción y el valor real que debería tener

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y(i)]^2$$

Nuestro objetivo es minimizar el Error promedio, osea minimizar la Función de costo

$$\underset{\theta_0, \theta_1}{minimize}\, J(\theta_0, \theta_1)$$

Gradient Descent

El **gradiente descendiente** es un algoritmo utilizado en diversas áreas del machine learning. En este caso se utilizará para minimizar el error promedio J de la función de costo mencionado previamente.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x^{(i)}) - y(i)]^2$$

Gradient Descent

Asumamos la siguiente representación matemática como una colina y estamos descendiendo por ella desde un punto en la zona roja. Se debe hacer una inspección en 360° y hallar la dirección por la cual se podría llegar más rápido a la parte más baja.

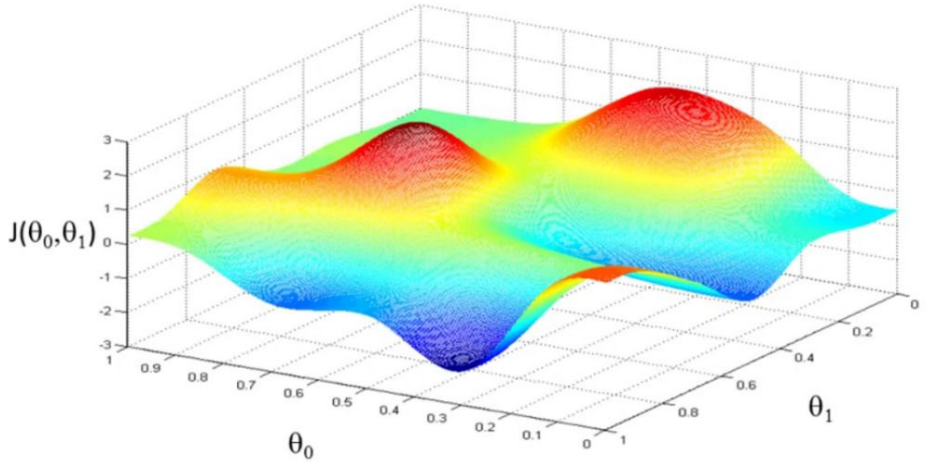


Figura 7. Representación matemática de la función de error promedio. Tomada del curso Machine Learning por la Universidad de Stanford.

Gradient Descent

Esto se debería repetir cada cierto tramo y volver a evaluar cuál es la nueva dirección por donde se tendría el descenso más rápido. Al final se obtendrá una trayectoria como se indica en la figura.

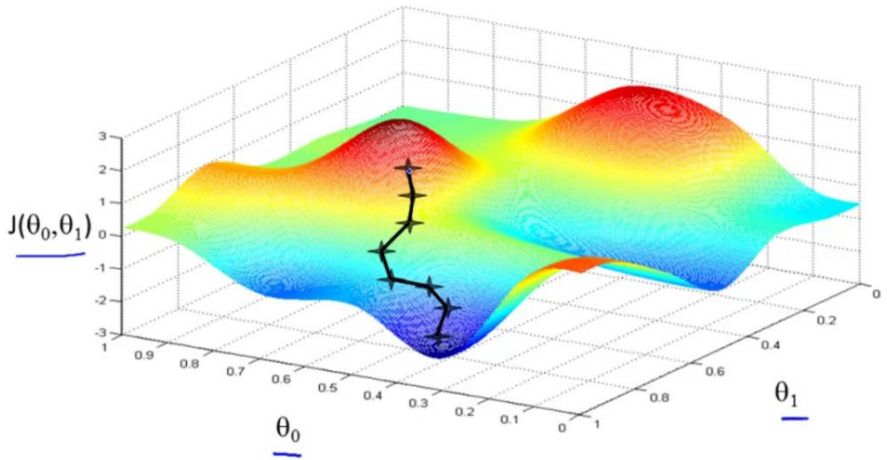


Figura 7. Trayectoria utilizada en representación del gradiente descendiente. Tomada del curso Machine Learning por la Universidad de Stanford.

Gradient Descent

El gradiente descendiente se denota por la siguiente ecuación, donde “α” es la tasa de aprendizaje. Si esta es pequeña el gradiente descendiente podría ser muy lento, por el contrario, podría tener problemas para converger o incluso no hacerlo.

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(simultaneously update } j = 0 \text{ and } j = 1)$$
$$\}$$

Gradient Descent for Linear Regression

Cuando se aplica el gradiente descendiente al modelo de regresión lineal se obtiene la siguiente ecuación.

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

Gradient Descent for Linear Regression

Si evaluamos la función de costo J para un modelo lineal en función de sus dos parámetros θo y θ1. Mientras más se llegue al centro de las curvas de J, más fidedigno será el modelo lineal h(x).

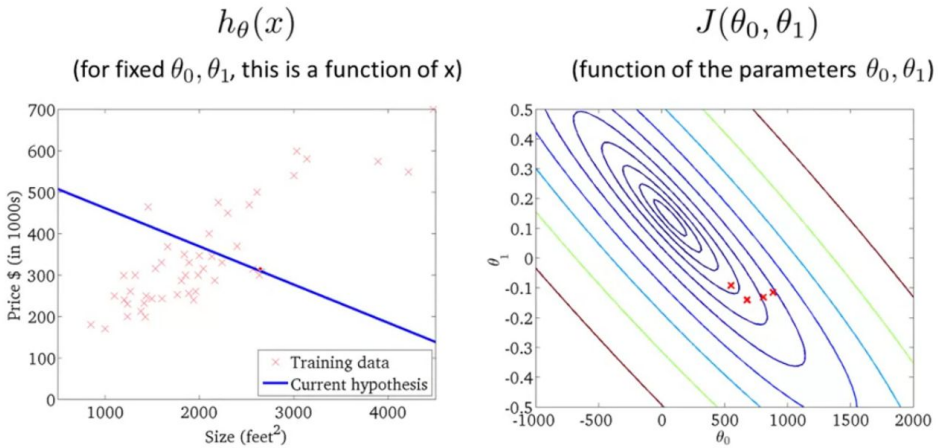


Figura 8. Representación de un modelo lineal con respecto a valores de θo y θ1 y su relación con la función de costo J

Gradient Descent for Multiple Variables

En este caso utilizaremos un modelo no lineal h(x) como se puede apreciar en la siguiente ecuación.

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient Descent for Multiple Variables

Con esto, el gradiente descendiente aplicado a un modelo en varias variables será denotado de la siguiente manera.

New algorithm $(n \geq 1)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

Resolviendo el problema de overfitting

¿Que es overfitting?

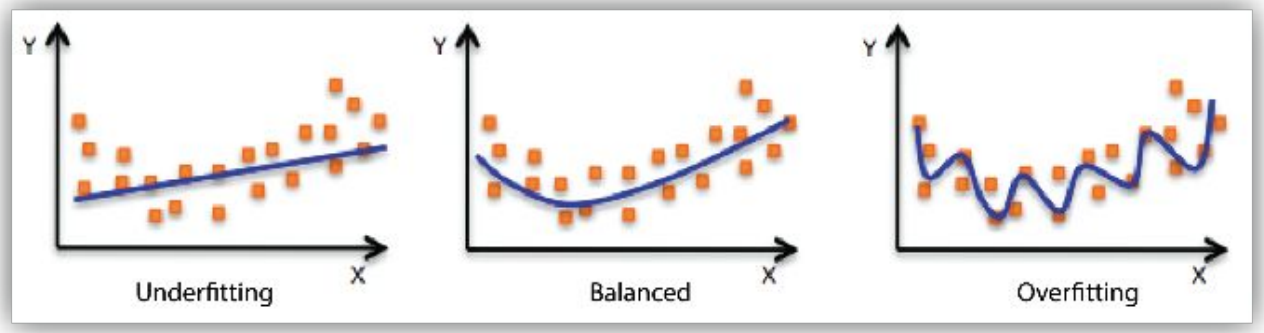


Figura 9. Diferencias de un modelo balanceado con respecto a modelos con problemas de underfitting y overfitting.

En el caso de overfitting, el modelo se adecua muy bien a los datos de entrenamiento, pero podría fallar al incluir nuevos datos o al predecir nuevos ejemplos. Se podría decir que es un “sobreentrenamiento”

¿Como evitamos el overfitting?

Con una tecnica llamada **Regularization**. Esta técnica busca disminuir o regular los coeficientes del modelo, transformándolo en un modelo más simple como se observa en la imagen.

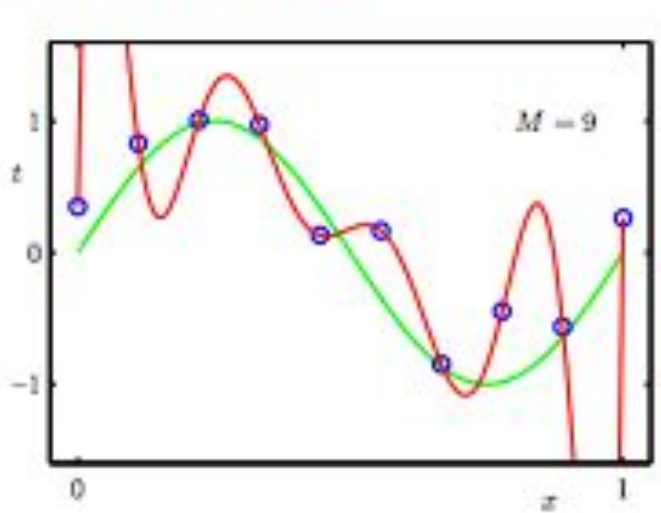


Figura 10. Modelo con overfitting(rojo) y modelo utilizando regularization (verde)

¿Como funciona Regularization ?

Linear regression

Se agrega una expresión a la función de costo.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

Para minimizar la nueva función de costo, los valores de theta tienen que reducirse.El parámetro de regularización λ ajusta el intercambio entre tener un modelo que se ajuste bien a los datos y mantener los parámetros theta pequeños.

Se actualiza la función de gradient descent

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

El efecto de regularización es que en cada actualización de los parámetros theta, antes de realizar la actualización original, se está reduciendo el valor inicial de theta dependiendo de los valores de λ y α.

Logistic regression

Se agrega una expresión a la función de costo.

$$J(\theta) = \left| -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_\theta(x^{(i)}) + (1 - y^{(i)}) \log 1 - h_\theta(x^{(i)})\right| \right| + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Se actualiza la función de gradient descent

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Neural Networks

¿Cómo funcionan las neuronas en el cerebro?

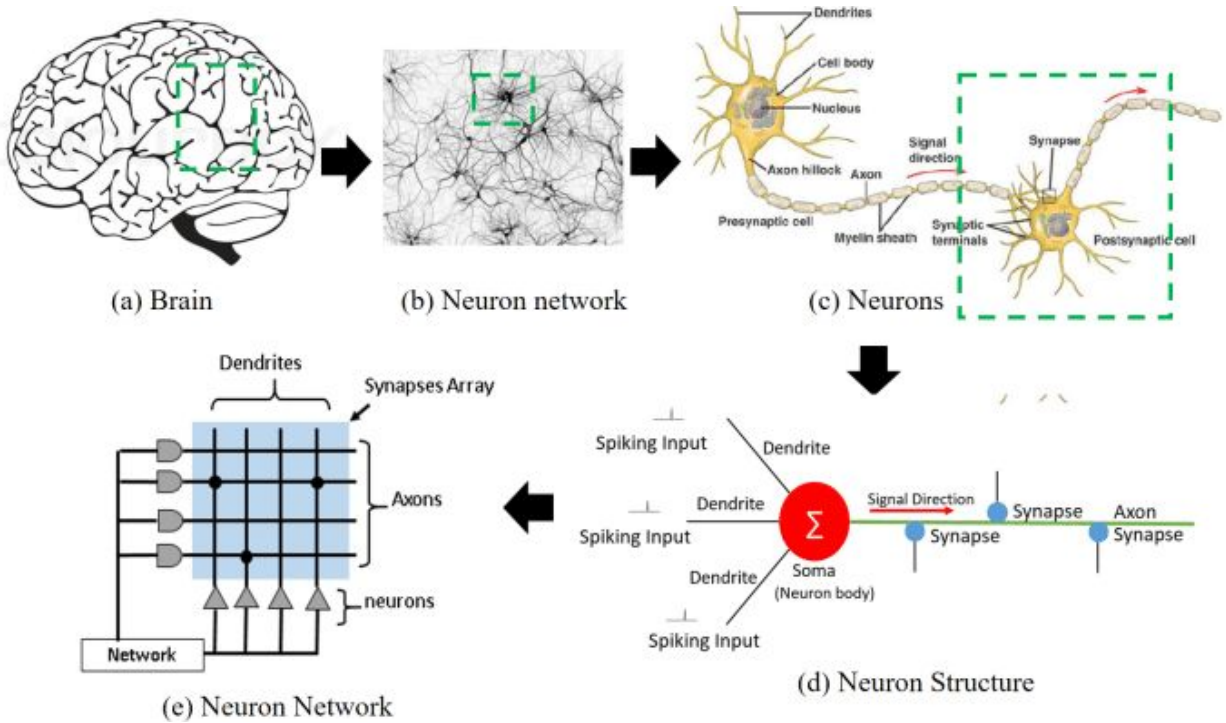


Figura 11. Funcionamiento de una neurona

Modelo de una neurona artificial

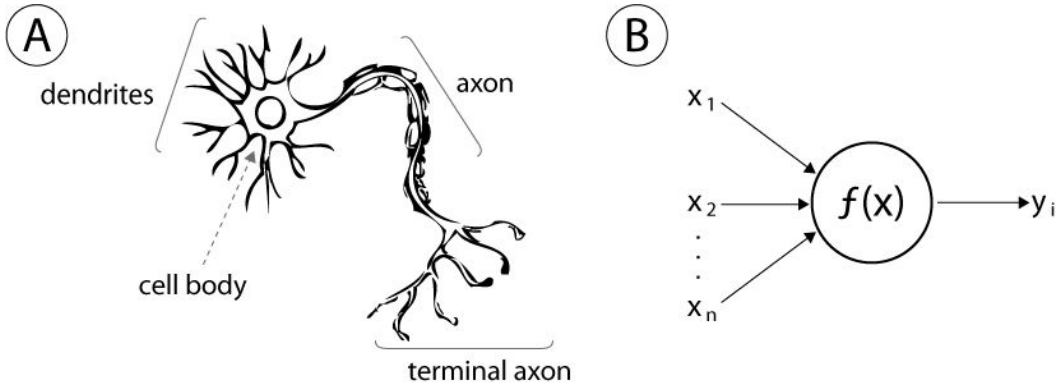


Figura 12. Comparación entre una neurona cerebral y una neurona artificial

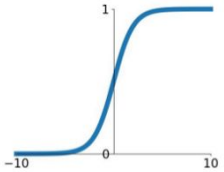
Para representar una neurona de manera artificial utilizamos un modelo logístico simple como se observa en el inciso B de la figura 12. El vector de entrada X representa a las dendritas y la salida Y representa al axon.

¿Que funcion representa F(x)?

A F(x) se le denomina función de activación. La figura muestra las funciones de activación más usadas.

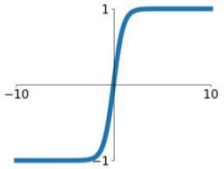
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



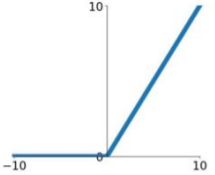
tanh

$$\tanh(x)$$

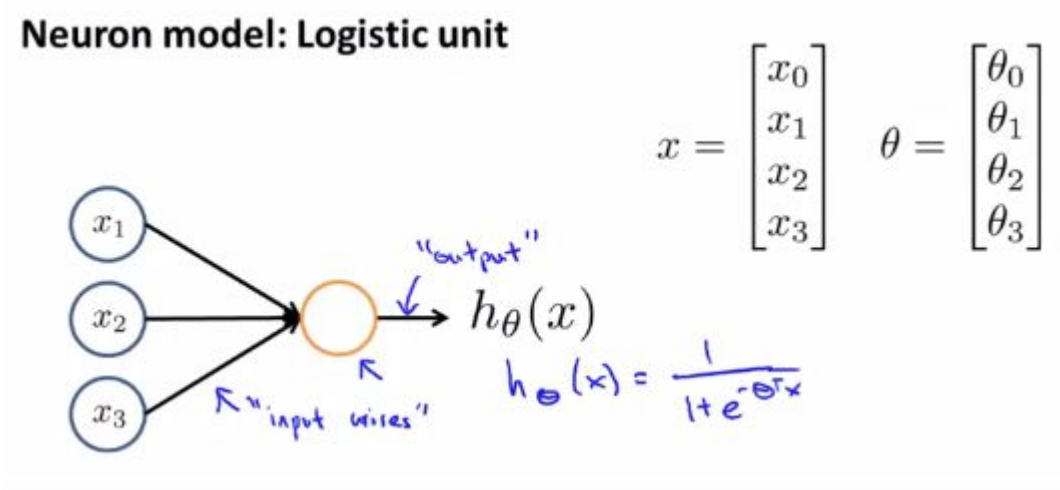


ReLU

$$\max(0, x)$$

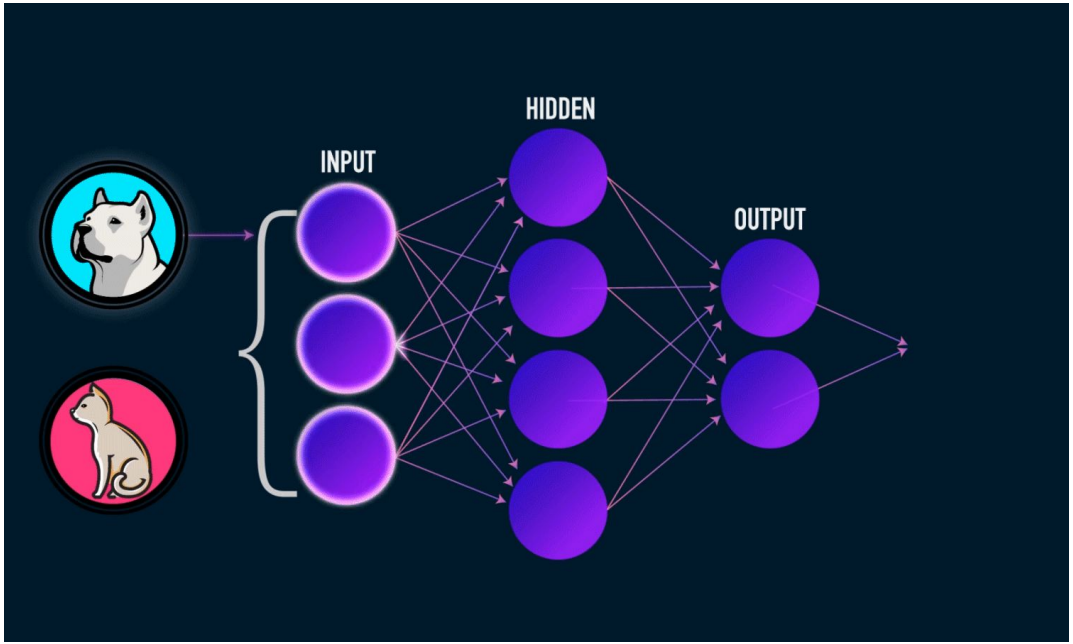


¿Como es la salida de la neurona?



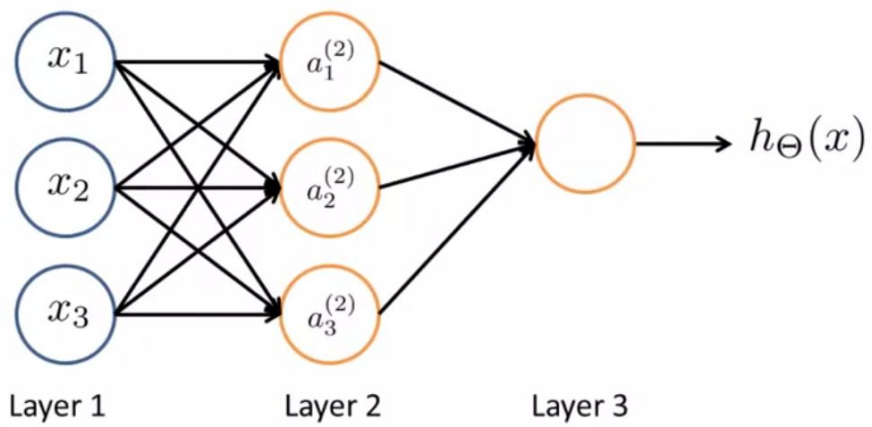
Como se puede observar, se ha utilizado una función de activación sigmoide. **θ** representa los parámetros del modelo y se le denomina “weights”.

Modelo de la Neural Network



Representación computacional

Para explicar mejor los cálculos matemáticos es mejor tener una notación.



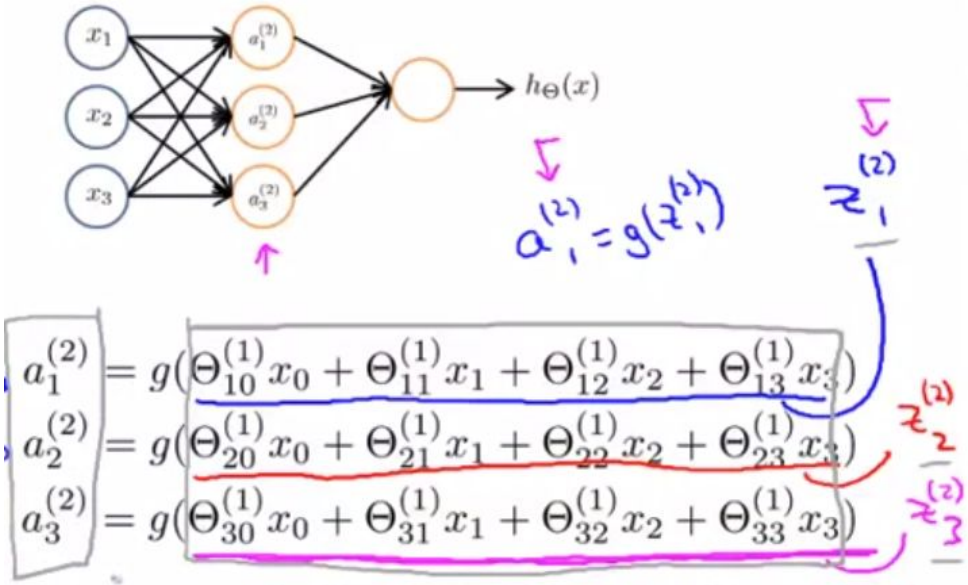
$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

Realizando los cálculos por cada neurona:

$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})
\end{aligned}
$$

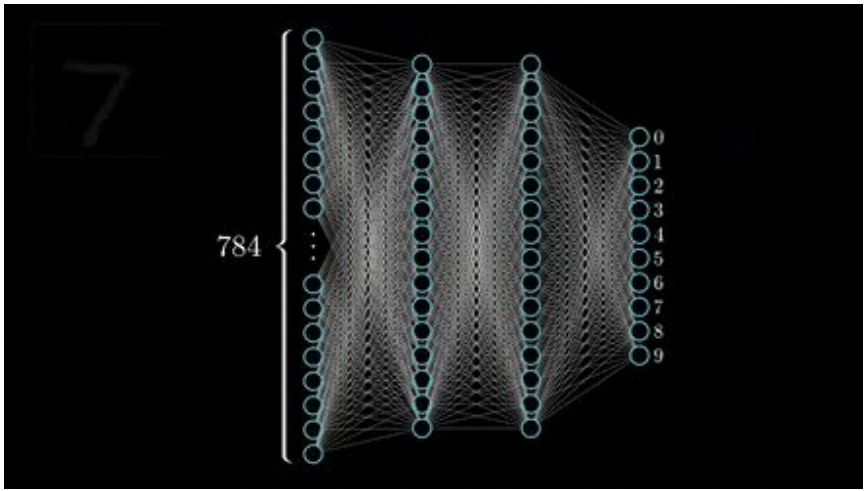
Con los resultados mostrados podemos hacer las siguientes generalizaciones:

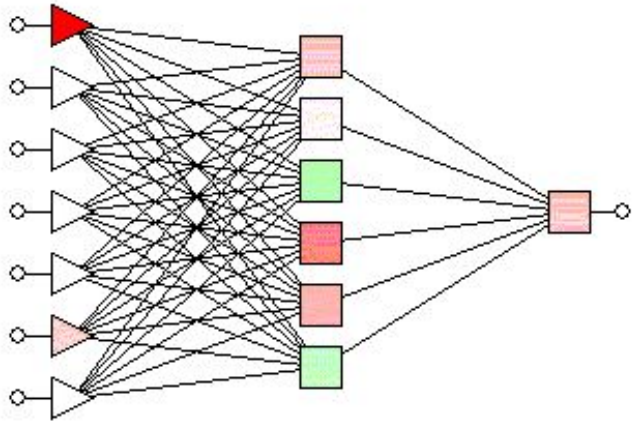


$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

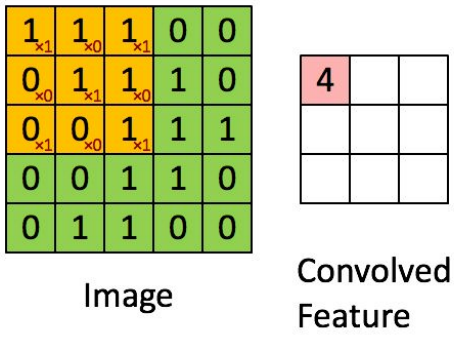
Otras arquitecturas de NN





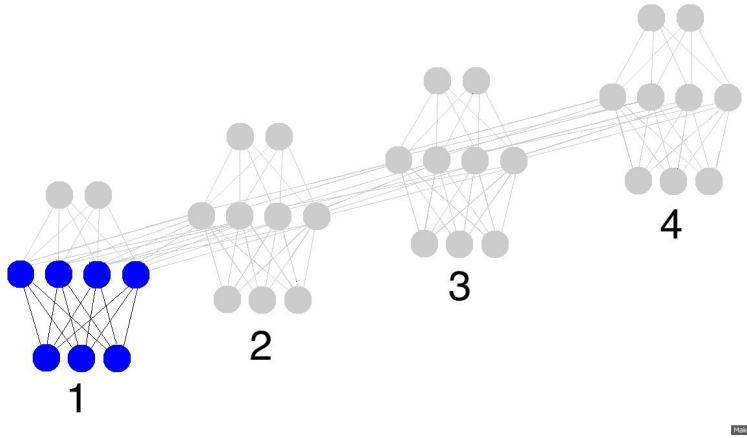
Convolutional Neural Networks

En una CNN, la primera capa siempre es convolucional. Están definidas usando tres dimensiones espaciales: largo, ancho y profundidad.Estas capas no están totalmente conectadas, en otras palabras, las neuronas de una capa no están conectadas a cada neurona de la siguiente capa.



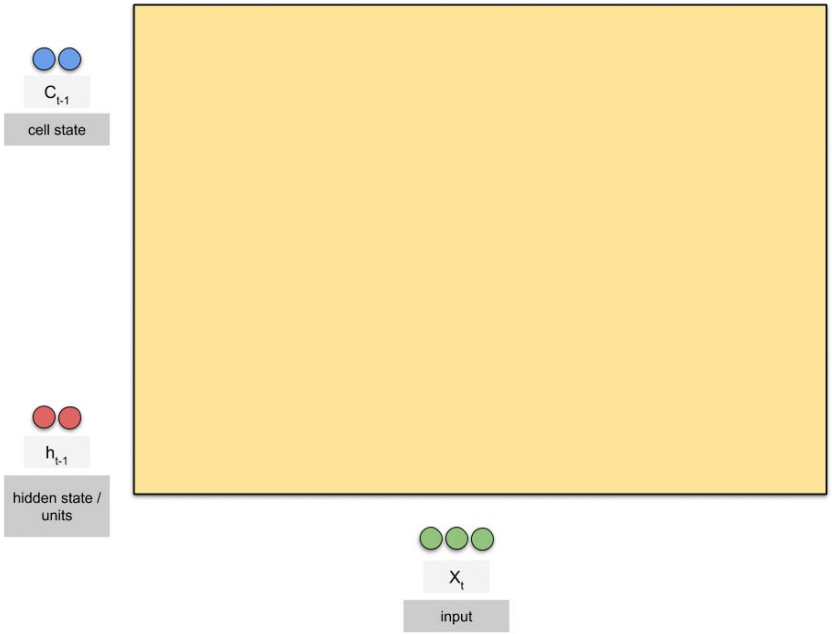
Recurrent Neural Networks

La RNN utiliza una arquitectura que no es diferente a la tradicional NN. La diferencia es que introduce el concepto de memoria, y existe en la forma de un diferente tipo de enlace. Las salidas de algunas capas son introducidas a las entradas de la capa anterior. Esta adición permite el análisis de data secuencial que las NN no pueden hacer.



Long Short Term Memory

Es un tipo especial de RNN que es capaz de aprender dependencias de largo término.



Investigación:
American Sign Language Recognition using Deep Learning and Computer Vision

Introducción

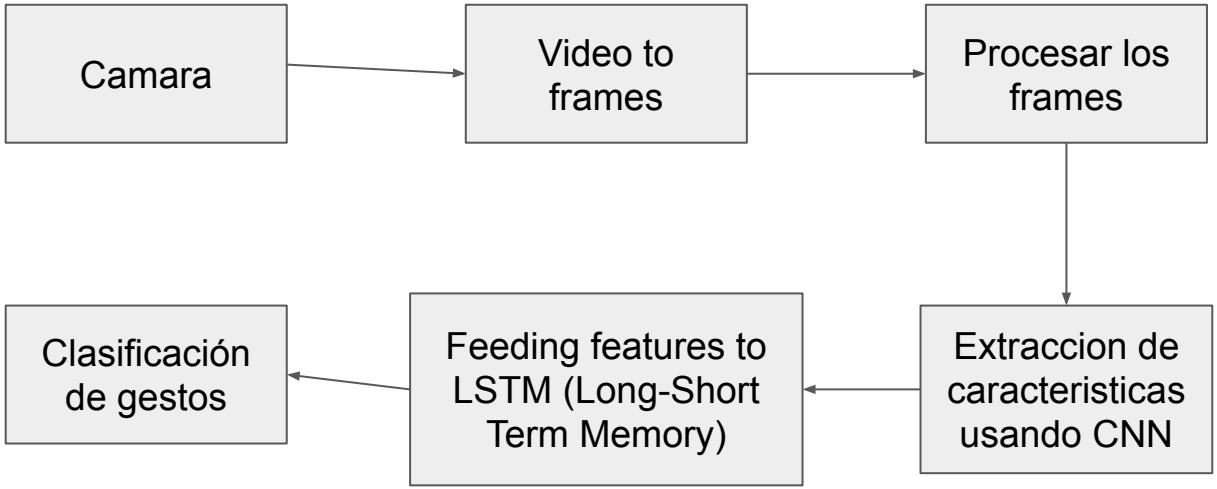
Este estudio fue publicado en 2018 en la conferencia de big data por IEEE.

El lenguaje de señas es utilizado por personas con discapacidad para hablar y/o escuchar. A pesar de que se ha utilizado por muchos años, hasta ahora es muy complicado su comprensión para personas que desconocen dicho lenguaje. Por ello, este estudio plantea una solución a dicho problema aplicando algoritmos de visión por ordenador para poder obtener la información del mensaje por medio de imágenes. Para esto se utilizó una base de datos llamada American Sign Language Dataset.

Acondicionamiento de base de datos

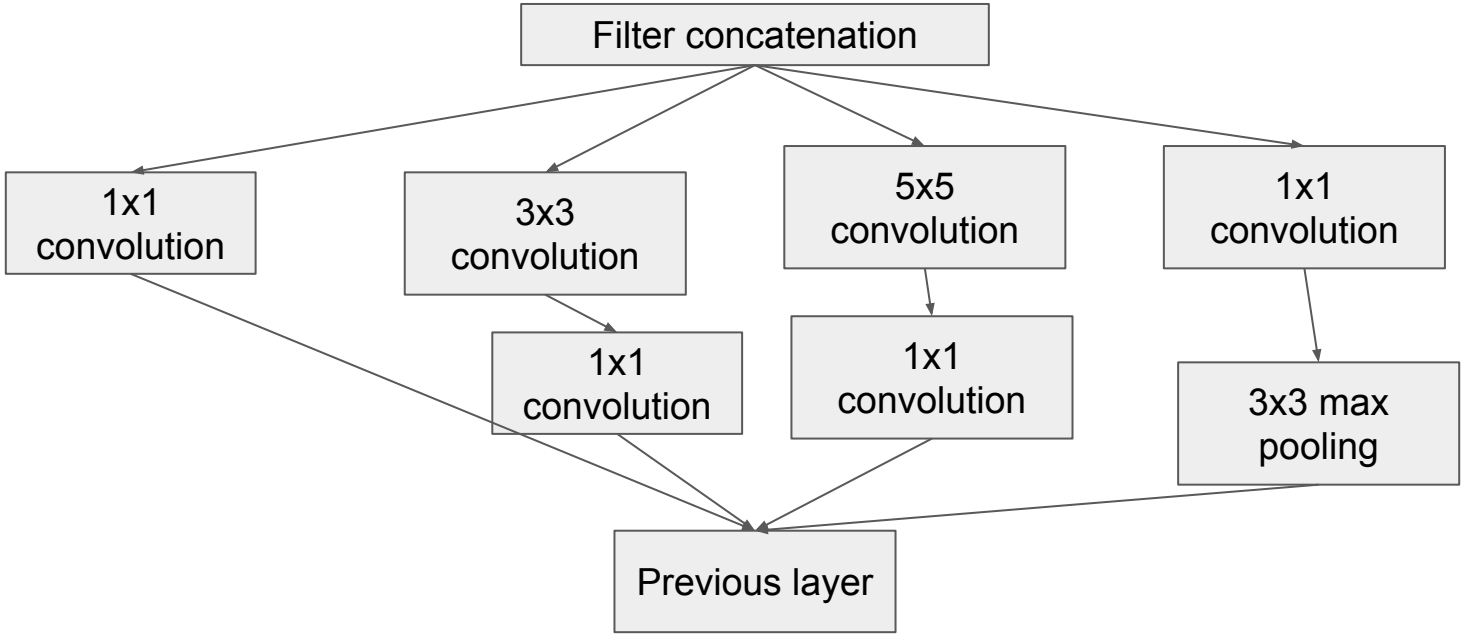
La base de datos original contenía cientos de señas distintas las cuales fueron realizadas cinco veces cada una con diferentes condiciones luminosas y la velocidad de ejecución de la seña. Los videos fueron grabados a 60fps con una resolución de 720p. De cada video se extrajo 300 cuadros. Luego se utilizó técnicas de incremento de base de datos como re escalamiento y rotación para obtener 2400 cuadros por cada seña dentro de la base de datos.

High level system architecture



Gesture Detection

- Modelo CNN: Inception (hecho por Google).
- Inception hace todas las convoluciones en paralelo y luego concatena el resultado antes de pasar a la siguiente capa.



Gesture Classification

- Se tomó como entradas las salidas de la capa Softmax y Max Pooling.
- Ambas salidas fueron ingresadas a la RNN (Recurrent Neural Network)

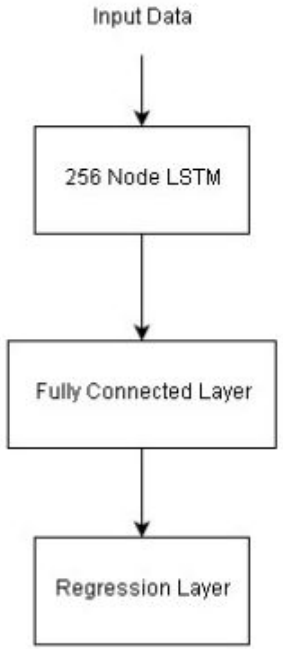


Figura 11. Arquitectura de RNN

Los gestos identificados de la red convolucional anterior son clasificados en la etapa LSTM dentro de alguna de las categorías establecidas.

Resultados

En la siguiente tabla podemos apreciar los resultados obtenidos para diferentes números de señas utilizados. También como la comparación al utilizar softmax layer y pooling layer.

# of Signs	Accuracy with Softmax Layer	Accuracy with Pool Layer
10	90%	55%
50	92%	58%
100	93%	58%
150	91%	55%

Figura 12. Tabla de resultados utilizando Softmax y Pooling layer.