Charla de grupo: Quantum Machine Learning

Marco Di Tullio

August 2018

Contents

1 Qué es Machine Learning?

Machine learning es un campo en computación que hace uso de técnicas estadísticas para hacer que la computadora "aprenda" con data, sin ser explícitamente programada. Intenta emular de cierto modo la manera en que los humanos aprenden. Pero en que sentido se distingue esto de encarar un problema de forma estadística? Dado un set discreto de data $S = \{x_i\}_{i=1}^{N}, x_i \in R$, el primer approach es encontrar regresión lineal. Queremos los parámetros que puedan fittear la data lo mejor posible. Esto puede hacerse sin Machine Learning sin ningún problema. La diferencia: en estadísitica se asumen cosas (en ajuste lineal, asumimos que la data tiene ruido gaussiano). En Machine Learning no se hace ninguna asunción.

Hay muchos métodos, pero los más difundidos son aquellos basados en Neural Networks.

Figure 1:

2 Neural Networks (redes neuronales)

¿Cómo hace un ser humano para darse cuenta que el mismo número escrito por dos personas distintas es efectivamente el mismo? Una red neuronal es un conjunto de nodos conectados por enlaces. Vamos a pensar a las neuronas como cosas que alojan un número, entre 0 y 1. A este número se lo conoce como activación. Siguiendo el ejemplo de los números escritos a mano, podemos pensar que la imagen del numero puede escribirse como una matriz de, por ejemplo, 28x28 cuyos valores son los niveles de gris de cada lugar.

La idea va a ser optimizar el valor de las conexiones de forma tal que a la salida se active el número que uno espera. Vamos a poner capas intermedias, tantas como queramos y con la cantidad de neuronas que queramos. Cada neurona de esta capa intermedia va a activarse de acuerdo a las neuronas de la capa anterior y los pesos de las conexiones. Si llamamos $\{a_1, a_2, ..., a_N\}$ a las activaciones de las N neuronas de la primer capa, y $\{w_{i1}, w_{i2}, ..., w_{iN}\}$ a los pesos de las N conexiones que llegan a la neurona i de la segunda capa, entonces consideramos la cantidad

$$S_i = \sum_j w_{ij} a_j$$

vale la pena notar que los pesos pueden ser negativos.

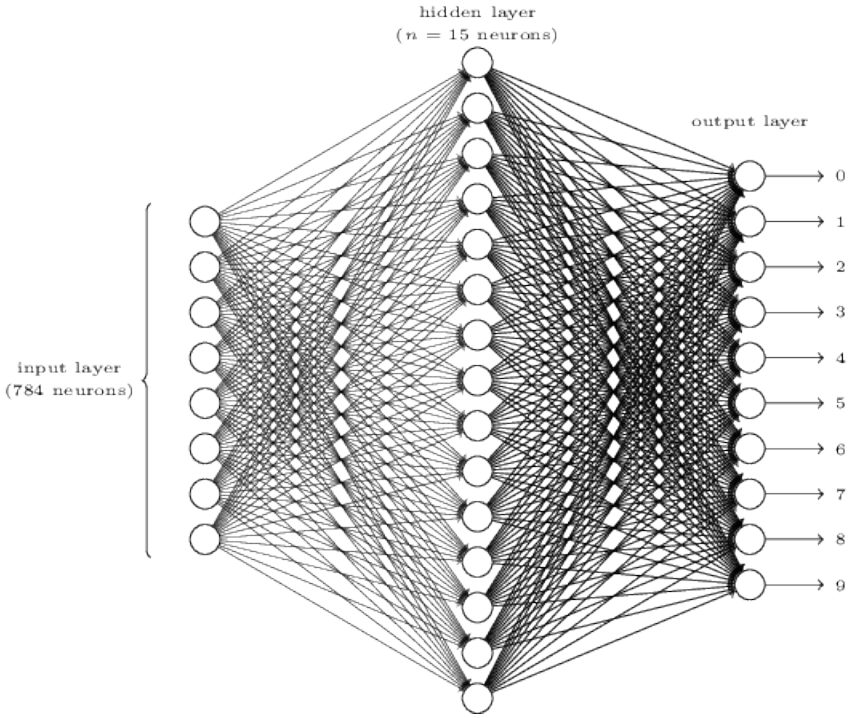

Figure 2:

La activación de la neurona i va a ser una función de S_i. Como queremos que esté entre 0 y 1, vamos a tomar la función sigmoide

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

Al aplicar esta función a S_i, pasamos a tener valores entre 0 y 1 que va a ser la activación de la neurona i en la segunda capa. Además, puede que querramos agregarle un offset, con lo cual ponemos un bias b_i.

Finalmente tenemos que la activación de la neurona i en la segunda capa va a ser

$$a_i^{(1)} = \sigma(\sum_j w_{ij} a_j^{(0)} + b_i)$$

Cada una de las neuronas de la segunda capa va a tener N conexiones con las neuronas de la primera (y por tanto N pesos) y 1 bias. Son muchimos parámetros. Y después esto se repite capa a capa. Dada una red con una capa de 784 neuronas a la entrada, dos capas intermedias de 16 y una salida de 10, hay 13002 parámetros.

Es mucho más comodo pensar esto de forma matricial. Se define un vector con las activaciones y una matriz con los pesos.

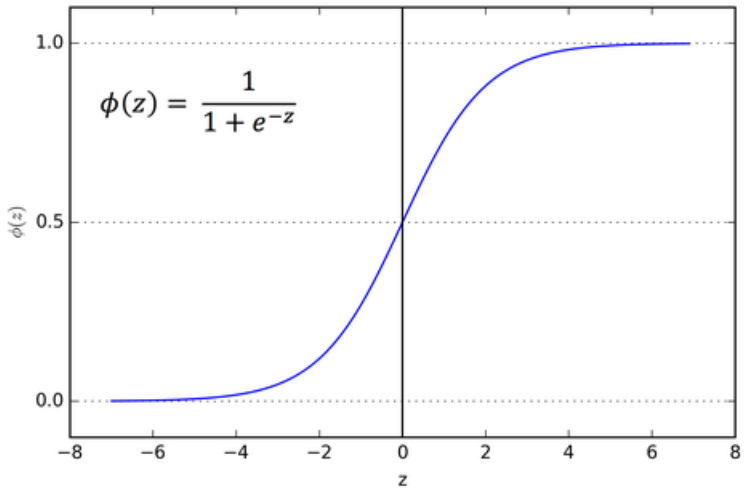

Figure 3:

$$\sigma\left[\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1N} \\ w_{21} & w_{22} & \dots & w_{2N} \\ \dots & \dots & \dots & \dots \\ w_{N1} & w_{N2} & \dots & w_{NN} \end{pmatrix} \begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \dots \\ a_N^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \dots \\ b_N^{(0)} \end{pmatrix}\right] = \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \dots \\ a_M^{(1)} \end{pmatrix}$$

Entonces una neurona no es más una cosa que contiene una activación, sino que es una función. La idea va a ser darle

Ahora, como es el entrenamiento? Hay dos grandes categorías: supervisado y no-supervisado. En el primer caso, la data se divide en categorías etiquetadas (los números escritos a mano junto con el número correspondiente). Por otra parte, los algoritmos no supervisados son aquellos donde la data no tiene ningún tipo de división y le corresponde al algoritmo hallar las clasificaciones adecuadas. Aquí vamos a ver el caso supervisado.

En todo entrenamiento supervisado, es necesario decirle a la máquina como se supone que deberían ser los resultados. Esto se hace por medio de una *cost function*, que puede ser una distancia a una cantidad deseada, o algo del estilo. Este costo va a ser lo que minimicemos en nuestro entrenamiento. Para hacer la minimización, tenemos que evaluar el costo del output correspondiente al primer input (*feed-forward*) y mover los pesos y biases en la dirección contraria al gradiente de la función costo. Es decir, tenemos que calcular un gradiente respecto a todas las variables de nuestro sistema y corregirlas con un pequeño desplazamiento. Esto es lo que se conoce como *backpropagation*, porque estamos volviendo hacia atrás en nuestra red. Luego repetimos este mecanismo con otro input, y así con todos los inputs que queramos usar para entrenar nuestra red.

Generalmente se divide la data en un grupo de entrenamiento y un grupo de testeo. La data de entrenamiento, se divide a su vez en epochs, que son subgrupos luego de los cuales se vuelve a inicializar los estados random. De esta forma se hace lo que se conoce como "stochastic gradient descent", que sirve para no trabarse tan facilmente en mínimos locales.

3 Que se puede hacer con computación cuántica

Un estado cuántico de n qubits es un vector complejo de $2n$ dimensiones. Las operaciones lógicas y las mediciones se corresponden con matrices de $2n \times 2n$. Estas transformaciones matriciales han demostrado tener speed-ups respecto de algunos problemas clásicos.

La idea de fondo es que con machine learning se busca encontrar patrones estadísticos en data. Si las computadoras cuánticas, incluso las más pequeñas, logran trabajan con patrones estadísticos difíciles de generar en computadoras clásicas, entonces quizas puedan reconocer patrones muy dificiles de encontrar clásicamente. Esta idea se basa en la eficiencia de las computadoras cuánticas y de que efectivamente existe un speed-up en este tipo de problemas.

Por ejemplo, para buscar en una base de datos no ordenada con N entradas, una computadora cuántica demora un tiempo proporcional a $\sqrt{N}$, mientras que una computadora clásica necesita un tiempo proporcional a N. De forma similar, computadoras cuánticas pueden hacer transformadas de Fourier sobre N datos, invertir matrices de $N \times N$ y hallar sus autovalores y autovectores, en un tiempo proporcional a $\log_2 N$, mientras que algoritmos clásicos necesitan tiempo del orden $N \log_2 N$.

3.1 PCA

El primer problema cercano a ML donde se consiguieron speed-ups con quantum computing fue el de PCA (principal component analysis), que es un método para encontrar correlaciones en data. Dada la data en forma de vectores $\vec{v}_j$ en un espacio vectorial d dimensional (por ejemplo el cambio de precios de las acciones del teimpo t_j al tiempo t_{j+1}). Se define la matriz de covarianza como $C = \sum_j \vec{v_j}\vec{v_j}^T$. El método de PCA diagonaliza esta matriz y estudia la distribución de los autovalores (más homogéneo implica más correlacionado). A partir de los autovectores y autovalores, es posible hacer compresiones y predecir comportamientos futuros. Las cuentas clásicas son de orden $O(d^2)$. En el caso cuántico, se elige de forma aleatoria un v_j y se lo mapea a un estado cuántico de largo $\log d$, con lo cual se tiene $\rho = (1/N) \sum_j |v_j\rangle \langle v_j|$. Usando algunos trucos, es posible diagonalizar esta matriz. El problema es ahora de $O((\log d)^2)$, siendo exponencialmente más rápido que su contraparte clásico.

3.2 Quantum Annealing

Es un método para hallar minimos globales de una dada función entre un conjunto de distintos candidatos usando fluctuaciones cuánticas. Son problemas discretos. Típicos ejemplos son hallar el estado fundamentar de un spin glass (un problema con muchos minimos locales). El sistema comienza el estado superposición de todos los posibles candidatos con el mismo peso. Luego evoluciona en el tiempo, cambiando de estados, siguiendo la ecuación de Schrodinger gobernada por algún campo externo transverso. SI el campo tiene frecuencia de campo baja, el sistema queda siempre cerca del estado fundamental. Si en cambio se tiene un cambio brusco, el sistema deja el estado fundamental temporalmente, pero produce una mejor chance de finalizar en el estado fundamental del Hamiltoniano final. Finalmente se apaga el campo y se espera que el sistema llegue

al estado fundamental de un modelo de Ising, que se corresponde con la solución del problema de optimización original.

4 Boltzmann Machines

La red deep neural network más fácil de cuantizar es la máquina de Boltzmann. La versión clásica consiste en bits con interacciones ajustables. Se entrana ajustando estas interacciones, tales que el estado térmico de los bits, descrito por distribuciones de Blotzmann reproduce la estadística de la data. Para cuantizarla, se toma la red y se la traduce a un sistema de spines interactuantes. Luego inicializando las neuronas de input a estados fijos, dejando termalizar al sistema y leyendo los qubits a la salida, se obtiene el resultado. Acá me conviene mezclar lo del paper (hoja 9 para adelatne) con lo de acá, que es la versión clásica https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-i-6df5c4918c15 https://arxiv.org/pdf/1611.09347.pdf