Red neuronal desde cero como grafo

Esta texto acompaña al código que crea una red neuronal desde cero que se encuentra aquí

https://colab.research.google.com/drive/1OEgZiRgJUi5fbu2oT--k-iDqpW19eg3z?usp=sharing

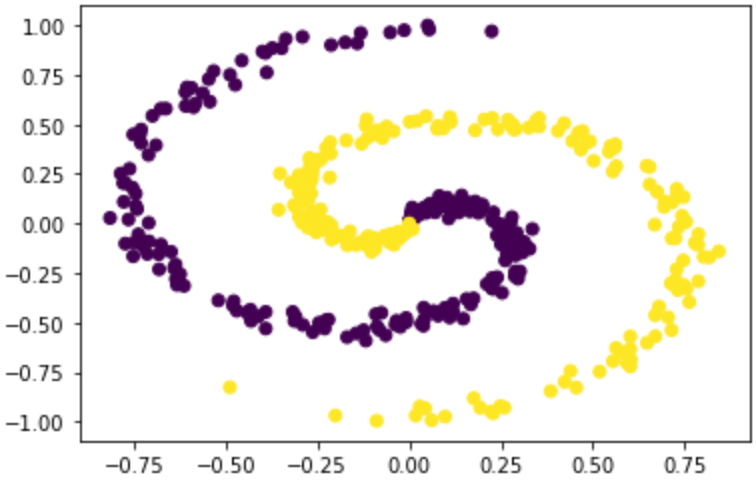
Aspectos prácticos

- La red que se tiene hasta ahora no es un grafo general sino una secuencia de capas fully connected
- Hasta el momento sólo ha sido posible entrenar una red con dos clases y diez muestras por que la red entrena muy lento
- Hasta el momento sólo usa backpropagation con los gradientes
- La red no incluye bias
- La última capa sigue unos aspectos técnicos que se explican más adelante

Generación de datos

- Usamos el mismo procedimiento que se usa en el curso de Deep Learning para generar los datos que se usa acá: https://github.com/Atcold/pytorch-Deep-Learning/blob/master/04-spiral_classification.ipynb
- El procedimiento genera hélices de datos para un número de clases establecidos, los datos no están desorganizados, es decir están en orden por "pétalo" generado

- Aquí vemos 200 datos con dos clases



Estructuras de datos

Por el momento esta versión de la red no es como un grafo general sino como una lista, cada elemento de la lista representa una capa conformada por: transformación afin (multiplicación matricial) y una no linealidad. La red está conformada por dos listas

1. Lista de pesos transformación afin (pesos)
2. Lista de no linealidades y su derivada

Generación de lista de pesos

La lista de pesos se genera con otra lista que determina la dimensión de las capas intermedias, también se utiliza la información de la dimensionalidad de entrada y la dimensionalidad de salida. Por ejemplo, con dos capas intermedias, salen 3 matrices:

1. De dimensionalidad entrada a dimensionalidad capa 1
2. De dimensionalidad capa 1 a dimensionalidad capa 2
3. De dimensionalidad capa 2 a dimensionalidad salida

Las matrcies quedan de la siguiente forma [M X N] donde N es la dimensión de la que se parte y la M la dimensión a la que se va; el código que genera esto es:

```
pesos = []
d_ = X.shape[1]
for capa in capas:
  W=np.random.randn(capa,d_)*math.sqrt(1/(capa+d_))
  pesos.append(W)
  d_=capa
W=np.random.rand(C,d_)
pesos.append(W)
```

La configuración que funciona hasta el momento es sólo una capa de dimensionalidad 100

Lista de linealidades

La lista de linealidades consiste en una lista de tuplas, conde la función lineal es asociada con su derivada. La configuración que funciona es la primera linealidad es con ReLu y la segunda con softmax.

El paso forward

El paso *forward* es muy directo, toma la entrada y cicla sobre los pesos para aplicar las operaciones

1. a=Wz
2. z=f(a)

El único truco es que hacer con el dato X de entrada ya que para hacer todas las operaciones en un ciclo, por lo que la z se inicializa con X y de esa forma se asocia al ciclo.

El modo collect

El modo collect tiene el objetivo de guardar cálculos hechos para ser utilizados en el paso backward

El paso backward

Función de costo

Se elijó la crossentropy para calcular la perdida, y para calcular su derivada estamos usando la versión descrita aquí:

- https://sefiks.com/2017/12/17/a-gentle-introduction-to-cross-entropy-loss-function/

Aquí hay que notar, que mientras la función e costo es escalar la derivada de esta es vectorial

Hacía atrás

Al igual que el paso hacia adelante, ahora hay que recorrer las listas de pesos y funciones no lineales en reversa por cada una de ellas hay que hacer lo siguiente:

1. dCda=dCdz * f'(a)
2. dCdz = w.T dCda y dCdw = dCda x.T

El componente dCdz es la derivada que tenemos hasta ese ciclo relacionada con el costo, y la idea es pasar por la no linealidad; la segunda derivadas sirven para pasar por la transformación afin dCdz ; sin embargo existen dos porque ahí podemos calcular dCdw que son los gradientes que se utilizarán para cambiar los pesos.

Notar que dCdz es la derivada a la entrada de la red neuronal, y nos servirá para calcular la siguiente dCda del siguiente ciclo. Tambień notar que se necesita una inicial, esa se convierte en la derivada de mi función costo es decir la derivada de crossentropy.

Truco con gradiente de softmax

El gradiente de softmax no es tan directo, sin embargo el gradiente de softmax y cross entropy es muy directo; entonces para que funcione se hizo lo siguiente:

1. El gradiente de crossentropy en realidad es el gradiente de crossentropy y softmax
2. El gradiente de softmax es el de la identidad para que tome los gradeitnes directos del crossentropy

Por supuesto lo anterior está muy limitado a que cuando usemos la función de perdida de crossentropy sólo podemos usar la capa de softmax.

Ciclo de entrenamiento

Para el entrenamiento se sigue el siguiente ciclo:

1. Se itera n *epochs*
2. Por cada *epoch* se itera todo el dataset
3. Por cada elemento del dataset se hace lo siguiente
    1. Se hace un forward con modo collect
    2. Se hace un backward para calcular los gradiente
    3. Se actualizan los gradientes por capa

Problemas identificados

1. Es muy lento
2. No ha alcanzado 100% de sobre entrenamiento
3. No es general en las operaciones
4. No incluye sesgos