DEEP LEARNING PRÁCTICA 2 - 2025

Redes Neuronales

1. En papel, utilizando lo visto sobre Grafos Computationales calcular las siguientes funciones de activación y todos sus gradiente para una red feedforward con dos entradas + umbral (bias):

   (a) Sigmoide
   $$f(x) = \frac{1}{1 + \exp(-x)} \tag{1}$$

   (b)
   $$f(x) = \tanh(x) \tag{2}$$

   (c) ELU:
   $$f(x) = x \;\; (x < 0) \quad \alpha(\exp(x) - 1) \;\; (x > 0) \tag{3}$$

   (d) Leaky Relu:
   $$f(x) = max(x/10, x) \tag{4}$$

   donde las entradas y los pesos son los siguientes, $x = (2, 8, -1, 8), w = (1.45, -0.35), b = -4$. Hacer las cuentas usando tres decimales.

2. De forma similar al punto anterior calcular la activación, función de costo y todos los gradientes involucrados en una red neuronal cuya arquitectura es la de un perceptrón compuesto por dos neuronas + bias con funciones de activación son sigmoide y la función de costo es MSE. El tamaño de la entrada es de 2 y tiene la siguiente configuración: $x = (-3.1, 1.5)$
   $$W = \begin{pmatrix} -0.2 & 2 \\ -0.5 & -0.3 \end{pmatrix} \tag{5}$$
   $b_1 = -4$, $b_2 = -1$.

3. Implementar una red neuronal (sin POO) de dos capas totalmente conectadas para resolver el problema de CIFAR-10. Las 100 neuronas de la primer capa tienen como función de activación a la función sigmoidal. La segunda capa es la de salida y esta formada por 10 neuronas con una activación lineal. Como función de costo utilizar MSE y

agregar un término de regularización L2. Por simplicidad considere la entrada como un vector 1D de 3072 componentes y la clasificación como un vector de categorías de la siguiente forma: la clase 1 se representa por [0 1 0 0 0 0 0 0 0 0]. Utilizando lo visto sobre Grafos Computationales armar el grafo y calcular los gradientes. Implementar la función accuracy para ir analizando la precisión del método. Graficar con matplotlib la evolución de la función de costo y la precisión (accuracy) del método a lo largo de las épocas.

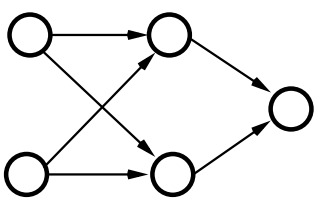
Opcionalmente analizar los resultados respecto a los clasificadores lineales SVM y LogisticRegression de scikit-learn. Armar un gráfico aparte para mostrar la precisión de estos métodos todos juntos (sólo para los datos de testing).

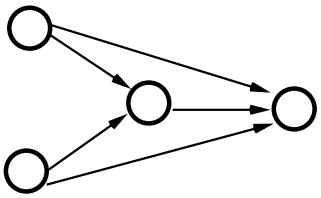
4. Idem al problema anterior pero ahora que la función de costo sea Categorical Cross Entropy. Esta función de costo puede verse, como aplicar la función softmax seguido de la loglikelihood. Modificar el código anterior para que se pueda cambiar la métrica a utilizar (loss mse o loss softmax) y lo mismo para el cálculo del gradiente (grad mse o grad softmax).

5. Idem al problema anterior pero esta vez que las funciones de activación de la capa oculta sean ReLU y las de la capa de salida sean sigmoidal. Al igual que en los problemas anteriores armar el grafo, calcular los gradientes, graficar la evolución de la precisión (accuracy) y las funciones de costo (mse y categorical cross-entropy) para las distintas épocas tanto para los datos de validación como entrenamiento. Discuta que métrica es mejor y porque. Además analizar que pasa si usamos una ReLU + activación lineal en la última capa. Explicar y fundamentar si usar una activación lineal en la última capa es mejor o no.

6. La regla XOR tiene dos entradas (1 o -1) y la salida es -1 si ambas son diferentes y 1 si ambas son iguales. Utilizar los conceptos de POO, paquetes/módulos de python, Grafo Computational y el algoritmo de retropropagación de errores para aprender el XOR en las siguientes arquitecturas (incluir unidades de entrada adicional para simular los umbrales):

   Utilizar como función de activación de las neuronas la función tanh y como función de costo MSE. Comparar la evolución de la precisión y





la función de costo a lo largo de las épocas entre ambas arquitecturas. Para resolver el problema se recomienda implementar el código en los siguientes módulos (utilice su criterio para ver cuando implementar clases o funciones):

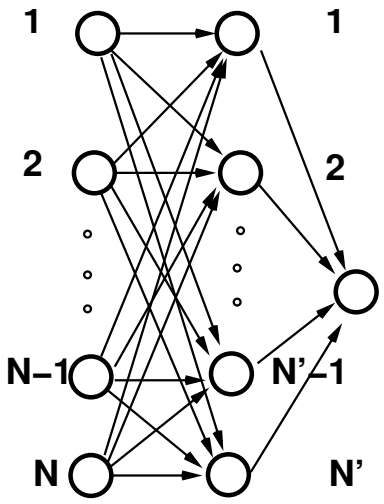
- metric.py
  - accuracy
  - MSE
- losses.py
  - Loss: Interfaz de las funciones de costo donde se define el uso del metodo
    `__call__`
    y gradient.
  - MSE: Clase donde se implementa la función de costo mse.
- activations.py
  - ReLU
  - Tanh
  - Sigmoid
- models.py
  - Network: Clase que implementa una red neuronal feedfoward.
- layers.py
  - BaseLayer: Clase genérica de cualquier tipo de capa.
  - Input: Representa la capa de entrada de la red neuronal que hereda las funcionalidades básicas de la clase BaseLayer.
  - Layer: Clase genérica de cualquier tipo de capa con pesos.

- Dense: Representa una capa densa que hereda las funcionalidades de la clase Layer.

- optimizers.py
  - Optimizador: Interfaz para los optimizadores.
  - SGD: clase que implementa el optimizador stochastic gradient descendent.

Nota: Generar todos los datos y utilizar un batch size del total de los ejemplos.

7. El problema de paridad es una generalización del XOR para N entradas. La salida es 1 si el producto de las N entradas es 1, y -1 si el producto de las entradas es -1. Implementar la red neuronal descripta en la gráfica para aprender el problema de paridad utilizando los conceptos y el código desarrollado en el ítem anterior. Estudiar el efecto de los tamaños de N y N'.



Generar todos los datos y utilizar un batch size del tamaño de la cantidad de ejemplos. Por otro lado, para los que quieren probar su destreza con python/numpy, intentar generear los datos utilizando la menor cantidad de líneas de código. ¿Fueron menos de 3?

8. Utilizando la implementación del punto 6 resolver el problema 3 agregando una capa oculta de 100 neuronas con la misma función de activación de la capa oculta existente. Graficar con matplotlib la evolución

de la función de costo y la precisión (accuracy) del método a lo largo de las épocas. Comparar estos resultados con los obtenidos en el punto 3.