Redes Neuronales y Aprendizaje Profundo para Visión Artificial

Dr. Ariel H. Curiale

Cuatrimestre: 2do de 2019

PRÁCTICA 2: INTRODUCCIÓN A LAS REDES NEURONALES

---

1. En papel, utilizando grafos y lo visto sobre *Computational Graphs* calcular las siguientes funciones de activación y su gradiente para un perceptron con dos entradas + umbral (bias):
   (a) sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
   (b) $\tanh(x)$
   (c) ELU: $f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$
   (d) Leaky Relu: $\max(0.1x, x)$

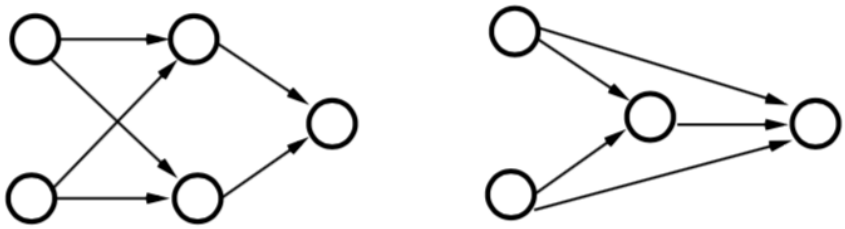
   donde las entradas y los pesos son los siguientes, $x = (2.8, -1.8)$, $w = (1.45, -0.35)$ y $b = -4$.
2. De forma similar al punto anterior calcular la activación y gradiente de un perceptrón compuesto por dos neuronas + bias cuyas funciones de activación son sigmoide. El tamaño de la entrada es de 2D y tiene la siguiente configuración: $x = (-3.1, 1.5)$, $W = \begin{pmatrix} -0.2 & 2 \\ -0.5 & -0.3 \end{pmatrix}$ y $b_1 = -4$ y $b_2 = -1$.
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 tanh. La segunda capa es la de salida y esta formada por 10 neuronas con una activación lineal. Como función de costo utilizar MSE y agregue un termino de regularización L2. Por simplicidad considere la entrada como un vector 1D de 3072 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 *Computational Graphs* armar el grafo y calcular los gradientes.

   Nota: 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.
4. Idem al problema anterior pero ahora que la función de costo sea *Categorical Cross-Entropy*.

   Nota: 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 softmax) para las distintas épocas tanto para los datos de validación como entrenamiento.

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, *Computational Graphs* y el algoritmo de retropropagación del error para aprender el XOR en las siguientes arquitecturas (incluir unidades de entrada adicional para simular los umbrales):

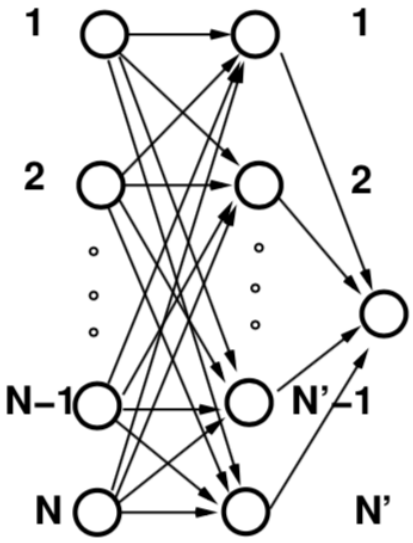


Utilizar como función de activación de las neuronas la función tanh y como función de costo MSE. Comparar la evolución de la precisión y la función de costo a lo largo de las épocas entre ambas arquitecturas. Para resolver el problema se recomienda implementar el código en los siguientes módulos (utilice su criterio para ver cuando implementar clases o funciones):

- metrics.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 (BGD).

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.



Nota: Para los que quieren probar su destreza con python/numpy, intentar genere los datos utilizando la menor cantidad de lineas de código. ¿Fueron menos de 3?

8. Utilizando la implementación del punto 6 resolver el problema 3 agregando una capa oculta de 100 neuronas. 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 6.