Instituto Tecnológico y de Estudios Superiores de Monterrey

Diseño de redes neuronales y aprendizaje profundo (Gpo 302)

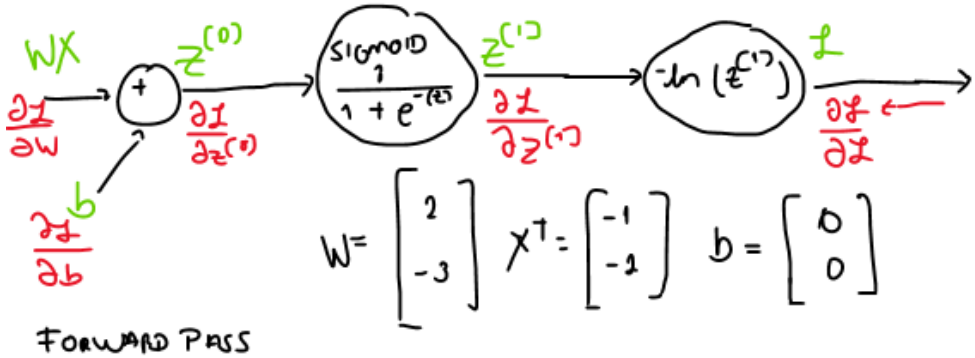
Actividad 2. Backpropagation

Equipo:
Enrique García Varela A01705747
Michelle Yareni Morales Ramón A01552627

Introducción

A través de esta actividad se realizaron distintas formas de llegar al resultado del proceso de forward pass y retropropagación en el primer paso. En la parte 2.1 se muestra el grafo computacional y el cálculo de los gradientes. Mientras que la sección 2.2 contiene el diseño de la red, la solución su descripción y validación.

2.1. Backpropagation en grafos (cálculo manual)



FORWARD PASS

$z^{(0)} = (2)(-1) + 0 + (-3)(-2) + 0 = 4$

$z^{(1)} = \frac{1}{1+e^{-(4)}} = 0.98201$

$\mathcal{L} = -\ln(0.98201) = -0.01814$

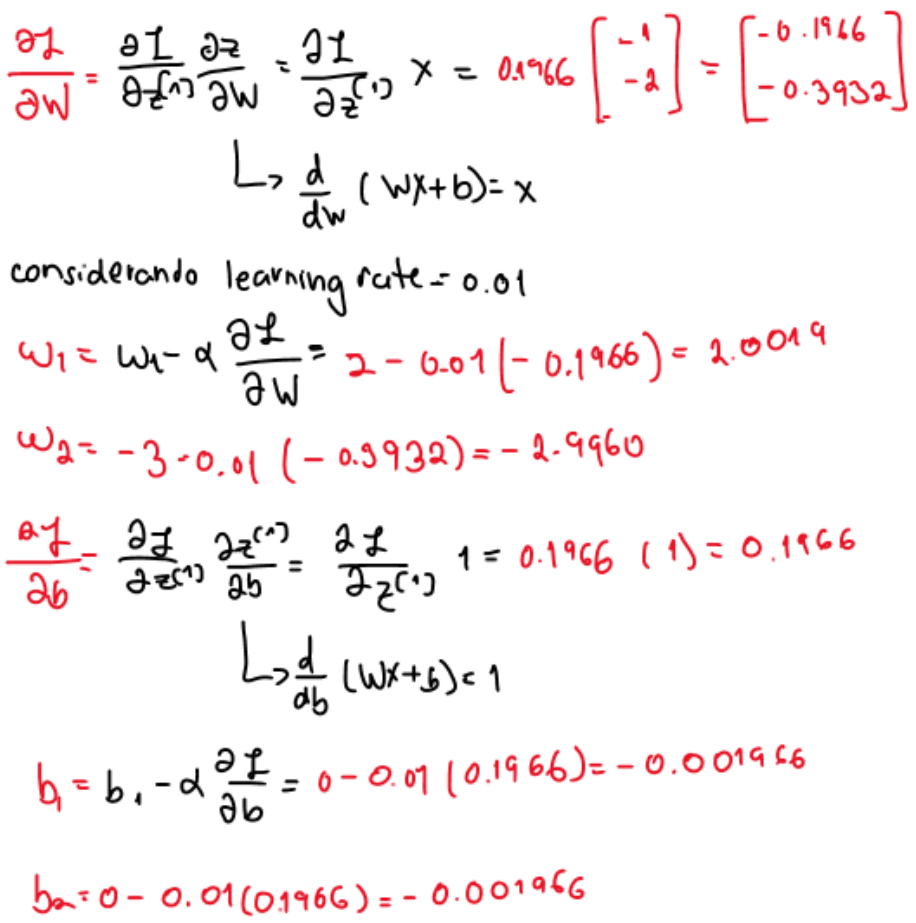
BACKPROPAGATION

$\frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$

$\frac{\partial \mathcal{L}}{\partial z^{(1)}} = \frac{d}{dz}(-\ln(z)) = -\frac{1}{z} = \frac{-1}{1} = -1$

$\frac{\partial \mathcal{L}}{\partial z^{(0)}} = \frac{d}{dz}((1+e^{-z})^{-1}) = \frac{-1}{(1+e^{-z})^2}\frac{d}{dz}(1+e^{-z})$

$= \frac{-1}{(1+e^{-z})^2}(-e^{-z}) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{e^{-(-1)}}{(1+e^{-(-1)})^2} = 0.1966$



2.2. Implementación de backpropagation y validación:

La red neuronal diseñada está dada por una capa, a la cual se introducen 2 variables con clasificación binaria. Esta cuenta con dos perceptrones como se muestra a continuación, para el desarrollo de toda la red neuronal estará dado por la función de calc_backpropagation.

```
#variables
x = np.array([[-1,-2]]).T
y = np.array([[0,1]]).T

def calc_backpropagation(x,y,act_func = None, floss = None):
  #parámetros 2 neuronas y 2 datos de la variable dependiente
  W = np.array([[2,-3]])
  #En caso de querer pesos sinápticos random: W = np.random.randn(3,4) * 0.01
  b = np.zeros((2,1))
  # de los siguientes resultados el indice del parametro representa el indice respectivo a la neurona
  print("PARÁMETROS ORIGINALES\npesos sinápticos:\n{} \nsesgo:\n{}".format(W,b))

  z = W@x + b
```

Además, hay diferentes formas de poder realizar el forward pass. Estado estas establecidas en la siguiente función de python llamada activation_function. De predeterminado tiene la función sigmoidal, pero se podría poner softmax o tanh. Esto se realizó principalmente con el objetivo de que en un futuro nos pudiéramos acoplar a las funciones que se encuentran dentro de las librerías de micrograd.

```
def activation_function(x,f = None):
  if f == "softmax":
    exp_x = np.exp(x)
    sum_exp = np.sum(exp_x)
    return exp_x / sum_exp
  elif f == "tanh":
    exp_x = np.exp(x)
    neg_exp_x = np.exp(-x)
    return (exp_x - neg_exp_x) / (exp_x + neg_exp_x)
  else:
    exp_x = np.exp(-x)
    return 1/(1+exp_x)
```

A continuación se realiza la función de pérdida, para el cual se podría realizar la aplicación de la función Cross-entropy o de Mean Square Error. Tiene el mismo propósito que el caso anterior sobre poder acercarnos a lo establecido por la librería de Microgad, ya que las funciones que se encuentran se ven limitadas.

```
def loss_function(y0,ypred, f = None):
  if f == "mse":
    return sum((y0-ypred)[0])**2
  else:
    return -np.log(y_hat[np.where(y==1)])

loss_i = loss_function(y, y_hat)
print('loss function inicial: {}'.format(loss_i))
```

Después, se hace la aplicación el proceso de backpropagation, que también estaría dependiendo de la misma función que se había establecido de un inicio para la activación de los perceptrones.

```
#Gradiente dloss_i = 1
def backprop(f = None):
  if f == "softmax": #solo se puede ocupar para función de perdida xentropy
    dz = y_hat - y
  elif f == "tanh":
    dz = 1 - y_hat ** 2
  else:
    dz = (np.exp(1))/((1+np.exp(1))**2) #Gradiente Sigmoid

  dW = dz * x.T #Gradiente pesos sinápticos
  db = dz #Gradiente sesgo
  return dW, db

dW, db = backprop(act_func)
```

Con ello, se hace la actualización de los pesos sinápticos y del sesgo, con respecto al gradiente calculado. Así como se muestran los resultados obtenidos.

```
learning_rate = 1e-2
W = W - learning_rate * dW
b = b - learning_rate * db

print("\nPARÁMETROS FINALES\n pesos sinápticos:\n{} \nsesgo:\n{} ".format(W,b))


z = (W@x + b)
y_hat = activation_function(z,act_func)
print('loss function inicial: {}'.format(loss_function(y, y_hat)))
```

Finalmente, se realizó la implementación de la red con Micrograd, a través de la función de multicapa perceptrón con la que se establece dos parámetros iniciales y las dos variables de inicio, esto para la primera capa. A continuación se declara que va a requerir dos neuronas y se crea la función de pérdida (MSE). También, cabe destacar que la función de activación se estaría ocupando tanh debido a que es la que permite Micrograd.

```
from micrograd.nn import MLP

xs = [-1.0, -2.0]
ys = [0.0, 1.0]

n = MLP(2, [2, 2]) #La función de activación que ocupa la API es tanh

for k in range(2):
  #-------------------------Forward pass--------------------------
  ypred = n(x)

  # al no poderse aplicar una funcion log en micrograd se obtiene mse
  loss = sum((yout - ygt) ** 2 for ygt, yout in zip(ys, ypred))

  #------------------------Backward pass-------------------------
  for p in n.parameters():
    p.grad = 0.0
  loss.backward()

  # update
  for p in n.parameters():
    p.data += 0.1 * p.grad

  print('iteración #{}'.format(k), '\nloss function: {}\n'.format(loss.data), '\nparametros: {}'.format(n.parameters()))
```

La implementación y validación se pueden visualizar en la siguiente liga:
https://colab.research.google.com/drive/1ziVlO3LL_DFJAS_BirRSHj36LQoj01VI