BackPropagation

Tinoco Sergio.
Instituto Politécnico Nacional, Escuela Superior de Cómputo.
Redes neuronales y aprendizaje profundo

23 de marzo de 2023

En el presente documento se va a presentar el desarrollo de una red neuronal optimizada por medio de back propagation en un paso y en 300 épocas.

Para ello se va a generar una red neuronal multicapa con 2 neuronas en la capa oculta.

Implementación BackPropagation Un Paso.

Primeramente se importa el paquete de numpy para trabajar con matrices.

```
#importamos paquetes
import numpy as np
```

Posteriormente se definen las funciones de activación y diferenciación.

```
def sigmoid_prime(x):
    return sigmoid(x)*(1-sigmoid(x))

def sigmoid(x):
    """
    Calculate sigmoid
    """
    return 1 / (1 + np.exp(-x))
```

Se definen los hiperparámetros de la red neuronal, valores de entrada, valor objetivo, taza de aprendizaje, y los pesos de la red.

```
x = np.array([0.5, 0.1, -0.2])
target = 0.6
learnrate = 0.5

weights_input_hidden = np.array([[0.5, -0.6],
                                 [0.1, -0.2],
                                 [0.1, 0.7]])

weights_hidden_output = np.array([0.1, -0.3])
```

Se definen los valores de entrada y salida de cada capa (Forward pass).

```
hidden_layer_input = np.dot(x, weights_input_hidden)
hidden_layer_output = sigmoid(hidden_layer_input)

output_layer_in = np.dot(hidden_layer_output, weights_hidden_output)
output = sigmoid(output_layer_in)
```

Se calculan los valores del error residual y los términos de error de cada capa (Backward pass) basado en fórmula 1.

$$\delta_o = (y - \hat{y}) f'(h)$$

$$\delta_h = \delta_o w_{h,o} f'(h_h)$$

Fórmula 1.

Términos de error.

```
# TODO: Calcula el término de error para la capa de salida
del_err_output = error*sigmoid_prime(output_layer_in)

# TODO: Calcula el término de error para la capa oculta
del_err_hidden = del_err_output* np.multiply(weights_hidden_output , sigmoid_prime(hidden_layer_input))
```

Posteriormente se calculan los incrementos de los pesos por medio de la fórmula 2.

$$\Delta w_{h,o} = \Delta w_{h,o} + \delta_o \hat{y}_h$$
$$\Delta w_{i,h} = \Delta w_{i,h} + \delta_h x_i$$

Fórmula 2.

Incremento de los pesos.

Se genera una matriz con los valores del término de error de la capa oculta de tal forma que cada fila contenga un término de error diferente para posteriormente realizar la multiplicación matricial.

```
# TODO: Calcular el incremento en cada peso de la capa oculta a la salida
delta_w_h_o = del_err_output*learnrate*hidden_layer_output

# TODO: Calcular el incremento en la capa de entrada a la oculta

delta_w_i_h = learnrate*(del_err_hidden*x[:,None])  #Transpuesta

print('Incremento de los pesos oculta a salida:')
print(delta_w_h_o)

print('Incremento de los pes}os de entrada a oculta:')
print(delta_w_i_h)
```

Resultados

```
Incremento de los pesos oculta a salida:
[0.00804047 0.00555918]
Incremento de los pes}os de entrada a oculta:
[[ 1.77005547e-04 -5.11178506e-04]
 [ 3.54011093e-05 -1.02235701e-04]
 [-7.08022187e-05  2.04471402e-04]]
```

Como se pudo observar en los resultados, cada capa va a adquirir un incremento a manera de vector o matriz de acuerdo al número de parámetros entre cada capa.

Dichos valores se van a sumar a nuestro pesos iniciales en una etapa posterior.

Implementación de BackPropagation completo.

Nuevamente se importan los paquetes y archivos necesarios para la implementación.

```
import numpy as np
from data_prep import features, targets, features_test, targets_test
```

Se definen las funciones de activación necesarias para el algoritmo.

```
def sigmoid(x):
    """
    Calculate sigmoid
    """
    return 1 / (1 + np.exp(-x))


def sigmoid_prime(x):
    return sigmoid(x)*(1-sigmoid(x))
```

Se definen los hiperparámetros de la red.

```
# Hyperparámetros
n_hidden = 2  # number of hidden units
epochs = 300
learnrate = 0.005
```

Se obtiene el número de características y de ejemplos.

```
n_records, n_features = features.shape
last_loss = None
```

Se generan los pesos iniciales por medio de una distribución normal.

```
weights_input_hidden = np.random.normal(scale=1 / n_features ** .5,
                                        size=(n_features, n_hidden))
weights_hidden_output = np.random.normal(scale=1 / n_features ** .5,
                                         size=n_hidden)
```

Se comienza la iteración de cada época para el algoritmo de BackPropagation.

```
for e in range(epochs):
    del_w_input_hidden = np.zeros(weights_input_hidden.shape)
    del_w_hidden_output = np.zeros(weights_hidden_output.shape)
```

En cada época se va a realizar un forward pass.

```
for x, y in zip(features.values, targets):
    ## Forward pass ##
    #Calculate the output

    hidden_input = np.dot(x,weights_input_hidden)
    hidden_output = sigmoid(hidden_input)

    output_in=np.dot(hidden_output,weights_hidden_output)
    output = sigmoid(output_in)
```

Posteriormente se define el Backward pass.

Se obtiene el error residual de cada iteración y el término de error de cada capa.

```
## Backward pass ##
# TODO: Calculate the error
error = y-output

# TODO: Calculate error gradient in output unit
output_error = error*sigmoid_prime(output_in)

# TODO: propagate errors to hidden layer
hidden_error = output_error*np.multiply(weights_hidden_output,sigmoid_p
```

Se obtienen los valores de incremento de los pesos siguiendo la fórmula 2.

```
# TODO: Update the change in weights
del_w_hidden_output += learnrate*output_error*hidden_output
del_w_input_hidden += learnrate*(hidden_error*x[:,None])
```

Se actualizan los pesos por cada época.

```
# TODO: Update weights
weights_input_hidden += del_w_input_hidden
weights_hidden_output += del_w_hidden_output
```

Se obtienen 10 muestras del error cuadrático medio utilizando los nuevos pesos.

```
if e % (epochs / 10) == 0:
    hidden_output = sigmoid(np.dot(x, weights_input_hidden))
    out = sigmoid(np.dot(hidden_output,
                         weights_hidden_output))
    loss = np.mean((out - targets) ** 2)

    if last_loss and last_loss < loss:
        print("Train loss: ", loss, "  WARNING - Loss Increasing")
    else:
        print("Train loss: ", loss)
    last_loss = loss
```

Se calcula la exactitud del modelo utilizando los nuevos pesos.

```
# Calculate accuracy on test data
hidden = sigmoid(np.dot(features_test, weights_input_hidden))
out = sigmoid(np.dot(hidden, weights_hidden_output))
predictions = out > 0.5
accuracy = np.mean(predictions == targets_test)
print("Prediction accuracy: {:.4f}".format(accuracy))
```

Resultados

Entrenamiento con 10 épocas.

```
Train loss:  0.2706673334333724
Train loss:  0.2655878330193409
Train loss:  0.26103220094684143
Train loss:  0.2569544926650341
Train loss:  0.25330971172229466
Train loss:  0.2500548662464056
Train loss:  0.24714964773251663
Train loss:  0.24455681257107317
Train loss:  0.24224234034449074
Train loss:  0.2401754322361078
Prediction accuracy: 0.7000
PS C:\Users\Sergio Tinoco\Desktop\Backpropagation>
```

Entrenamiento con 400 épocas.

```
Train loss:  0.2706673334333724
Train loss:  0.2500548662464056
Train loss:  0.23832840089879223
Train loss:  0.23161504645355424
Train loss:  0.2276661782149478
Train loss:  0.2252653953183287
Train loss:  0.22375745422325288
Train loss:  0.22278197812540176
Train loss:  0.22213491848656094
Train loss:  0.2216971420461159
Prediction accuracy: 0.7500
PS C:\Users\Sergio Tinoco\Desktop\Backpropagation>
```

Entrenamiento con 100 épocas.

```
Train loss:  0.2706673334333724
Train loss:  0.23832840089879223
Train loss:  0.2276661782149478
Train loss:  0.22375745422325288
Train loss:  0.22213491848656094
Train loss:  0.22139698958170445
Train loss:  0.22104840544502874
Train loss:  0.22089320622926106
Train loss:  0.2208458939917315
Train loss:  0.22086600060737027   WARNING - Loss Increasing
Prediction accuracy: 0.7500
PS C:\Users\Sergio Tinoco\Desktop\Backpropagation>
```

Entrenamiento con 1000 épocas.

```
Train loss:  0.2706673334333724
Train loss:  0.2276661782149478
Train loss:  0.22213491848656094
Train loss:  0.22104840544502874
Train loss:  0.2208458939917315
Train loss:  0.22093311822228542   WARNING - Loss Increasing
Train loss:  0.22117037575274942   WARNING - Loss Increasing
Train loss:  0.2215186963328865   WARNING - Loss Increasing
Train loss:  0.22196741385666646   WARNING - Loss Increasing
Train loss:  0.22251391945696458   WARNING - Loss Increasing
Prediction accuracy: 0.7250
PS C:\Users\Sergio Tinoco\Desktop\Backpropagation>
```

Conclusiones

Como se pudo observar en los resultados, la pérdida en cada época fue disminuyendo al menos para las primeras 100, épocas, ya que al momento de utilizar 200 épocas, el modelo sufrió un incremento de los errores causando una menor exactitud del modelo. Claro que esto también depende de la taza de aprendizaje y los pesos obtenidos. Por lo que el número de épocas también es una variable a considerar y posiblemente con algún método matemático se pueda estimar el número óptimo de épocas a utilizar en un modelo de red neuronal.