Entrenamiento con retropropagación

En este ejercicio implementaremos el algoritmo de retropropagación dentro del descenso por gradiente para actualizar todos los pesos de la red durante varias épocas. Para entrenar la red usaremos el conjunto de datos de calificaciones que vimos previamente.

Alumno: Trejo Arriaga Rodrigo Gerardo

```
#importamos paquetes y datos
import numpy as np
from data_prep import features, targets, features_test, targets_test

# Definiciones útiles
np.random.seed(21)

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

```
# Hyperparámetros
n_hidden = 2  # number of hidden units
epochs = 900
learnrate = 0.005

# Obtenemos el número de entradas (features) asi como el número de ejemplos (n_r
n_records, n_features = features.shape
last_loss = None

# Creamos las matrices de los pesos.
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)
```

Entrenamiento - Parámetros originales

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

```
            ## Forward pass ##
            hidden_input = np.dot(x, weights_input_hidden)
            hidden_output = sigmoid(hidden_input)
            output = sigmoid(np.dot(hidden_output, weights_hidden_output))

            ## Backward pass ##
            # Calcula el error de salida (target - actual)
            error = y - output

            # Calcula el término de error para la capa de salida
            output_error_term = error * output * (1 - output)

            # Propaga los errores a la capa oculta
            hidden_error_term = np.dot(weights_hidden_output, output_error_term) *

            # Actualiza el cambio en los pesos de la capa oculta a la capa de salic
            del_w_hidden_output += output_error_term * hidden_output

            # Actualiza el cambio en los pesos de la entrada a la capa oculta
            del_w_input_hidden += hidden_error_term * x[:, None]

        # Actualiza los pesos
        weights_input_hidden += learnrate * del_w_input_hidden / n_records
        weights_hidden_output += learnrate * del_w_hidden_output / n_records

        # Imprime el error cuadrático medio en el conjunto de entrenamiento
        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

# Calcula la precisión en los datos de prueba
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: {:.3f}".format(accuracy))
```

```
Train loss:  0.2763000206585236
Train loss:  0.27487280940102565
Train loss:  0.27348146900538234
Train loss:  0.2721253511981268
Train loss:  0.2708037972995826
Train loss:  0.2695161402601932
Train loss:  0.2682617065761968
Train loss:  0.26703981808591787
Train loss:  0.26584979364857986
Train loss:  0.26469095070807314
Prediction accuracy: 0.425
```

Entrenamiento - Parámetros modificados

```
# Hyperparámetros
n_hidden = 6  # number of hidden units
epochs = 900
learnrate = 0.009

# Obtenemos el número de entradas (features) asi como el número de ejemplos (n_
n_records, n_features = features.shape
last_loss = None

# Creamos las matrices de los pesos.
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)


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)
    for x, y in zip(features.values, targets):
        ## Forward pass ##
        hidden_input = np.dot(x, weights_input_hidden)
        hidden_output = sigmoid(hidden_input)
        output = sigmoid(np.dot(hidden_output, weights_hidden_output))

        ## Backward pass ##
        # Calcula el error de salida (target - actual)
        error = y - output

        # Calcula el término de error para la capa de salida
        output_error_term = error * output * (1 - output)

        # Propaga los errores a la capa oculta
        hidden_error_term = np.dot(weights_hidden_output, output_error_term) *

        # Actualiza el cambio en los pesos de la capa oculta a la capa de salic
        del_w_hidden_output += output_error_term * hidden_output

        # Actualiza el cambio en los pesos de la entrada a la capa oculta
        del_w_input_hidden += hidden_error_term * x[:, None]

    # Actualiza los pesos
    weights_input_hidden += learnrate * del_w_input_hidden / n_records
    weights_hidden_output += learnrate * del_w_hidden_output / n_records

    # Imprime el error cuadrático medio en el conjunto de entrenamiento
    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)
```

```
        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

# Calcula la precisión en los datos de prueba
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: {:.3f}".format(accuracy))
```

```
Train loss:  0.2698065764332278
Train loss:  0.26033829974716854
Train loss:  0.2524088524372422
Train loss:  0.2458411869677293
Train loss:  0.24044974472707353
Train loss:  0.236055574656742
Train loss:  0.23249521089471464
Train loss:  0.22962479256103419
Train loss:  0.22732098510705367
Train loss:  0.22547999549054817
Prediction accuracy: 0.750
```

Conclusión

A través de la ejecución de este ejercicio de retropropagación, he consolidado mi entendimiento del entrenamiento de redes neuronales. Pude observar cómo la red ajusta iterativamente sus pesos en respuesta a los errores de predicción, mediante un proceso que refleja una forma fundamental de aprendizaje profundo. Este algoritmo no solo es el núcleo de muchas aplicaciones modernas de IA, sino que también actúa como una piedra angular conceptual para los profesionales del campo. Al implementar manualmente la retropropagación, reafirmamos la importancia de una comprensión profunda de los principios básicos que sustentan tecnologías más avanzadas y automatizadas, permitiéndonos apreciar las abstracciones de alto nivel que estas herramientas proporcionan. Con cada peso actualizado, damos un paso más hacia la creación de modelos predictivos precisos y robustos.