Proyecto

December 16, 2020

1 Proyecto final

Proyecto final de **Metodos Numericos** del curso impartido en *Ago - Dic 2020*.

2 Redes Neuronales "feedforward"

Benjamin Rivera

2.1 Historia de las redes neuronales

1958 - Perceptron Frank Rosenblatt

- Concepto de *pesos*

1965 - Perceptron multicapa

- Umbrales fijos
- Valores binarios

1980 - Aprendizaje automatico

1986 – Backpropagation

Modernidad

- 1989 Redes Neuronales Convolucionales
- 1997 Long Short Term Memory (LSTM) y Redes Neuronales Recurrentes

Era de hielo

Deep Learning

- Transformers
- Deepe Learning
- Autoencoders
- Redes Adversarias
- Restricted Boltzmann Machines ( )

2.2 Clasificacion de las redes neuronales

Por capa

- Monocapa (red de Hopfield)
- Multicapa

Por topologia

- Feedforward
- Recurrente (LSTM)
- Convolucionales

Por tipo de entrenamiento

- Por iluminacion divina (Estocastico o matematico)
- Aprendizaje supervisado (Backpropagation)
- Aprendizaje no supervisado (Redes adversarias)
- Aprendizaje por refuerzo

2.3 En este trabajo ...

Se tratara de implementar una red neuronal feedforward con una estrategia de aprendizaje de *backpropagation*; por lo que trabajaremos en un ambiente de *entrenamiento supervisado*.

Perceptrom

$$\sum_{i=1}^{i=q} w_i * p_i$$

Back propagation

1. Identificacion de la diferencia entre el estimulo y la aproximacion de la red
2. actualizacion de los pesos por el porcentaje del gradiente.

Para el calculo de diferencias y gradiente

```
for l in range(len(a) - 2, 0, -1):
    deltas.append(deltas[-1].dot(self.weights[l].T)*self.activation_prime(a[l]))
```

backpropagation

```
for i in range(len(self.weights)):
    layer = np.atleast_2d(a[i])
    delta = np.atleast_2d(deltas[i])
    self.weights[i] += learning_rate * layer.T.dot(delta)
```

2.3.1 Implementacion

Para la implementacion de este proyecto unicmanete se usara la libreria `numpy`.

[1]:
```
import numpy as np
```

Mejores alternativas

- Keras
- PyTorch
- TensorFlow
- scikit-learn

[2]:
```python
# Funciones de Activacion
def relu(x):
    return 0 if x <= 0 else x

def relu_derivada(x):
    # Simplificacion para evitar errores.
    return 0 if x <= 0 else 1

def step(x):
    return 0 if x < 0 else 1

def step_derivada(x):
    # Simplificacion para evitar errores.
    return 0

def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))

def sigmoid_derivada(x):
    return sigmoid(x)*(1.0-sigmoid(x))

def tanh(x):
    return np.tanh(x)

def tanh_derivada(x):
    return 1.0 - x**2
```

[3]:
```python
class NeuralNetwork:
    """ Clase para generalizar redes neuronales feedforward.

    Esta clase tiene como objetivo la generalizacion de las redes neurona_
    les feedforward. Esta generalizacion no permite hacer modificaciones
    a la estructura estandar donde todos los nodos (a exepcion de los de
    salida) se comportan como perceptrones conectados a las siguientes
    capas.

    Se tiene un unico constructor que inicializa todo como se pide
    """

    def __init__(self, layers, activation='tanh', w_range = (-1,1)):
        # Verificacion de datos de entrada
```

```
        if activation not in ['sigmoid', 'tanh', 'relu', 'step']:
            raise Exception("Funcion de activacion no reconocida. ")
        if w_range[0] >= w_range[1]:
            raise Exception("Error en el rango de los pesos. ")

        # Seleccion de funcion de activacion
        if activation == 'relu':
            self.activation = relu
            self.activation_prime = relu_derivada
        elif activation == 'step':
            self.activaition = step
            self.activation_prime = step_derivada
        elif activation == 'sigmoid':
            self.activation = sigmoid
            self.activation_prime = sigmoid_derivada
        elif activation == 'tanh':
            self.activation = tanh
            self.activation_prime = tanh_derivada

        # Inicializacion de pesos
        self.weights = []
        self.deltas = []
        # valores aleatorios a pesos de entrada y capa oculta
        for i in range(1, len(layers) - 1):
            r = np.random.uniform(w_range[0], w_range[1], (layers[i-1] + 1,␣
↪layers[i] + 1))
            self.weights.append(r)
        # valores aleatorios a capa de salida
        r = np.random.uniform(w_range[0], w_range[1], (layers[i] + 1,␣
↪layers[i+1]))
        self.weights.append(r)


    def fit(self, X, y, learning_rate=0.1, epochs=100000):
        """ Funcion para realizar la aproximacion del modelo a los datos dados.

        Esta funcion trata de ajustar los pesos de la red (backpropagation) para
        que se ajuste a los datos dados. Permite que se ajuste el "ratio de
        aprendizaje" y la cantidad de iteraciones para el entrenamiento.

        Input:
            X := Datos de la capa de entrada
            y := Datos esperados en la capa de salida
            learning_rate := Ratio de aprendizaje del modelo
            epochs := Cantidad de iteraciones para el modelo
        """
```

```
        # Agregacion de bias a capa de entrada
        ones = np.atleast_2d(np.ones(X.shape[0]))
        X = np.concatenate((ones.T, X), axis=1)

        for k in range(epochs):
            i = np.random.randint(X.shape[0])
            a = [X[i]]

            for l in range(len(self.weights)):
                    dot_value = np.dot(a[l], self.weights[l])
                    activation = self.activation(dot_value)
                    a.append(activation)

            # Calculo del error en el modelo
            error = y[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]

            # Calculamos los deltas
            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.weights[l].T)*self.
↪activation_prime(a[l]))
            self.deltas.append(deltas)
            deltas.reverse()

            # backpropagation
            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate * layer.T.dot(delta)

            if k % 5000 == 0: print('epochs:', k)

    def evaluate(self, x):
        """ Funcion para evaluar una entrada en la red. """
        ones = np.atleast_2d(np.ones(x.shape[0]))
        a = np.concatenate((np.ones(1).T, np.array(x)), axis=0)
        for l in range(0, len(self.weights)):
            a = self.activation(np.dot(a, self.weights[l]))
        return a

    def set_weights(self, w):
        """ Funcion establecer ciertos pesos. """
        try:
            if w.shape == self.weights.shape:
                self.weights = w
        except AttributeError as e:
            print("Los pesos se mantienen. ")
```

```
            print(e)

    # Getters
    def get_weights(self):
        return self.weights

    def get_deltas(self):
        return self.deltas
```

[22]:
```
def fast_training(X, y, hiden_layers=[3], epochs=15000, learning_rate=0.02,␣
↪activation='tanh'):
    """ Implementacion simplificada. """
    layers = [len(X[0]), len(y[0])]
    for lay in hiden_layers:
        layers.insert(-1, lay)

    nn = NeuralNetwork(layers, activation=activation)
    nn.fit(X, y, learning_rate=learning_rate, epochs=epochs)

    for i in range(len(X)):
        print("X:", X[i], "y:", y[i], "\n\tAproximacion: ", nn.evaluate(X[i]))
```

2.3.2 Ejemplos

- Compuerta logica NOT
- Compuerta logica AND
- Compuerta logica OR
- Compuerta logica XOR - Compuerta logica AND

Not

[23]:
```
# Not
X = np.array([[0],
              [1]])

y = np.array([[1],
              [0]])
```

[24]:
```
fast_training(X, y)
```

```
epochs: 0
epochs: 5000
epochs: 10000
X: [0] y: [1]
        Aproximacion:  [0.97880185]
X: [1] y: [0]
        Aproximacion:  [0.00051898]
```

[7]:
```
fast_training(X, y, epochs=5001)
```

```
epochs: 0
epochs: 5000
X: [0] y: [1]
        Aproximacion:  [0.95406978]
X: [1] y: [0]
        Aproximacion:  [0.00098776]
```

[8]:
```
fast_training(X, y, epochs=25001)
```

```
epochs: 0
epochs: 5000
epochs: 10000
epochs: 15000
epochs: 20000
epochs: 25000
X: [0] y: [1]
        Aproximacion:  [0.98385756]
X: [1] y: [0]
        Aproximacion:  [0.00014997]
```

And

[9]:
```
# And
X = np.array([[0, 1],
              [0, 0],
              [1, 0],
              [1, 1]])

y = np.array([[0],
              [0],
              [0],
              [1]])
```

[10]:
```
fast_training(X, y)
```

```
epochs: 0
epochs: 5000
epochs: 10000
X: [0 1] y: [0]
        Aproximacion:  [0.0011221]
X: [0 0] y: [0]
        Aproximacion:  [-0.00148757]
X: [1 0] y: [0]
        Aproximacion:  [0.00260917]
X: [1 1] y: [1]
        Aproximacion:  [0.96141139]
```

[11]:
```
fast_training(X, y, epochs=5001)
```

```
epochs: 0
epochs: 5000
X: [0 1] y: [0]
        Aproximacion:  [0.07204406]
X: [0 0] y: [0]
        Aproximacion:  [-0.07873938]
X: [1 0] y: [0]
        Aproximacion:  [0.04653255]
X: [1 1] y: [1]
        Aproximacion:  [0.78034912]
```

[12]:
```
fast_training(X, y, epochs=50001)
```

```
epochs: 0
epochs: 5000
epochs: 10000
epochs: 15000
epochs: 20000
epochs: 25000
epochs: 30000
epochs: 35000
epochs: 40000
epochs: 45000
epochs: 50000
X: [0 1] y: [0]
        Aproximacion:  [0.00034363]
X: [0 0] y: [0]
        Aproximacion:  [-0.00028827]
X: [1 0] y: [0]
        Aproximacion:  [0.00036937]
X: [1 1] y: [1]
        Aproximacion:  [0.98325664]
```

OR

[13]:
```
# And
X = np.array([[0, 1],
              [0, 0],
              [1, 0],
              [1, 1]])

y = np.array([[1],
              [0],
              [1],
              [1]])
```

[14]:
```
fast_training(X, y)
```

```
epochs: 0
epochs: 5000
epochs: 10000
X: [0 1] y: [1]
        Aproximacion:  [0.97629469]
X: [0 0] y: [0]
        Aproximacion:  [0.00113128]
X: [1 0] y: [1]
        Aproximacion:  [0.97815186]
X: [1 1] y: [1]
        Aproximacion:  [0.99436356]
```

[15]:
```
fast_training(X, y, epochs=5001)
```

```
epochs: 0
epochs: 5000
X: [0 1] y: [1]
        Aproximacion:  [0.96028761]
X: [0 0] y: [0]
        Aproximacion:  [0.00419791]
X: [1 0] y: [1]
        Aproximacion:  [0.95119924]
X: [1 1] y: [1]
        Aproximacion:  [0.9903503]
```

[16]:
```
fast_training(X, y, epochs=50001)
```

```
epochs: 0
epochs: 5000
epochs: 10000
epochs: 15000
epochs: 20000
epochs: 25000
epochs: 30000
epochs: 35000
epochs: 40000
epochs: 45000
epochs: 50000
X: [0 1] y: [1]
        Aproximacion:  [0.98773139]
X: [0 0] y: [0]
        Aproximacion:  [0.00029107]
X: [1 0] y: [1]
        Aproximacion:  [0.98918482]
X: [1 1] y: [1]
        Aproximacion:  [0.99850818]
```

XOR

[17]:
```
# And
X = np.array([[0, 1],
              [0, 0],
              [1, 0],
              [1, 1]])

y = np.array([[1],
              [0],
              [1],
              [0]])
```

[18]:
```
fast_training(X, y)
```

```
epochs: 0
epochs: 5000
epochs: 10000
X: [0 1] y: [1]
        Aproximacion:  [0.95353668]
X: [0 0] y: [0]
        Aproximacion:  [0.00173207]
X: [1 0] y: [1]
        Aproximacion:  [0.95132908]
X: [1 1] y: [0]
        Aproximacion:  [0.00415304]
```

[19]:
```
fast_training(X, y, epochs=5001)
```

```
epochs: 0
epochs: 5000
X: [0 1] y: [1]
        Aproximacion:  [0.64351828]
X: [0 0] y: [0]
        Aproximacion:  [0.18061695]
X: [1 0] y: [1]
        Aproximacion:  [0.58315306]
X: [1 1] y: [0]
        Aproximacion:  [0.3167121]
```

[20]:
```
fast_training(X, y, epochs=50001)
```

```
epochs: 0
epochs: 5000
epochs: 10000
epochs: 15000
epochs: 20000
epochs: 25000
epochs: 30000
```

```
epochs: 35000
epochs: 40000
epochs: 45000
epochs: 50000
X: [0 1] y: [1]
        Aproximacion:  [0.97937955]
X: [0 0] y: [0]
        Aproximacion:  [0.00030344]
X: [1 0] y: [1]
        Aproximacion:  [0.9796585]
X: [1 1] y: [0]
        Aproximacion:  [0.00120609]
```

2.4 Referencias

[1]MIGALA D, TIPOS DE REDES NEURONALES ARTIFICIALES (2019) | INTELIGENCIA ARTIFICIAL. 2018.

[2]"Activation function", Wikipedia. dic. 13, 2020, Consultado: dic. 16, 2020. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=993982136.

[3]S. Torres y N. Nadia, "Aplicación de Redes Neuronales en controladores de baterías", dic. 2019, Consultado: dic. 16, 2020. [En línea]. Disponible en: http://dspace.unila.edu.br:80/handle/123456789/5826.

[4]P. J. Werbos, "Backpropagation through time: what it does and how to do it", Proceedings of the IEEE, vol. 78, núm. 10, pp. 1550–1560, oct. 1990, doi: 10.1109/5.58337.

[5]"Backpropagation-BP – Numerentur.org". http://numerentur.org/backpropagation/ (consultado dic. 16, 2020).

[6]"Multicapa Sobreaprendizaje Perceptron Neuronas". https://web.archive.org/web/20140714231842/http://www
de-neuronas/perceptron-multicapa.html (consultado dic. 16, 2020).

[ ]: