ARTIFICIAL NEURAL NETWORK

in Python LANGUAGE

Chapter 4: Back Propagation & Optimization

4.1. Back Propagation

- **Partial derivative:**
  - The partial derivative measures how much impact a single input has on a function's output.
  - Euler's notation:

$$f(x,y,z) \quad \rightarrow \quad \frac{\partial}{\partial x}f(x,y,z), \frac{\partial}{\partial y}f(x,y,z), \frac{\partial}{\partial z}f(x,y,z)$$

Example: $f(x,y) = x + y \quad \rightarrow$

$$\frac{\partial}{\partial x}f(x,y) = \frac{\partial}{\partial x}[x+y] = \frac{\partial}{\partial x}x + \frac{\partial}{\partial x}y = 1 + 0 = 1$$

$$\frac{\partial}{\partial y}f(x,y) = \frac{\partial}{\partial y}[x+y] = \frac{\partial}{\partial y}x + \frac{\partial}{\partial y}y = 0 + 1 = 1$$

4.1. Back Propagation

- **Partial derivative:**

Examples:

$$f(x,y) = 2x + 3y^2 \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[2x + 3y^2] = \frac{\partial}{\partial x} 2x + \frac{\partial}{\partial x} 3y^2 =$$

$$= 2 \cdot \frac{\partial}{\partial x} x + 3 \cdot \frac{\partial}{\partial x} y^2 = 2 \cdot 1 + 3 \cdot 0 = 2$$

$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[2x + 3y^2] = \frac{\partial}{\partial y} 2x + \frac{\partial}{\partial y} 3y^2 =$$

$$= 2 \cdot \frac{\partial}{\partial y} x + 3 \cdot \frac{\partial}{\partial y} y^2 = 2 \cdot 0 + 3 \cdot 2y^1 = 6y$$

$$f(x,y) = x \cdot y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = \frac{\partial}{\partial x}[x \cdot y] = y \frac{\partial}{\partial x} x = y \cdot 1 = y$$

$$\frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial y}[x \cdot y] = x \frac{\partial}{\partial y} y = x \cdot 1 = x$$

4.1. Back Propagation

- **Partial derivative:**

Partial derivative of max function:

$$f(x,y) = \max(x,y) \rightarrow \frac{\partial f(x,y)}{\partial x} = \frac{\partial \max(x,y)}{\partial x} = \begin{cases} 1 & if \quad x \geq y \\ 0 & if \quad x < y \end{cases}$$

$$f(x,0) = \max(x,0) \rightarrow \frac{\partial f(x,y)}{\partial x} = \begin{cases} 1 & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases}$$

Gradient is a vector composed of **all of the partial derivatives** of one function, calculated in function of the **input variables**.

4.1. Back Propagation

- **Gradient:**

Gradient is the vector composed by all partial derivatives of the function. Denotation: Nabla ∇

Example: $f(x, y, z) = 3x^3z - y^2 + 5z + 2yz$

$$\nabla f(x,y,z) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y,z) \\ \frac{\partial}{\partial y} f(x,y,z) \\ \frac{\partial}{\partial z} f(x,y,z) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} f(x,y,z) = \begin{bmatrix} 9x^2z \\ -2y + 2z \\ 3x^3 + 5 + 2y \end{bmatrix}$$

Gradient descent algorithm: An optimal algorithm that help finding the local minimum of a function (in the case of a NN: the loss function) by making converging the model's parameters to optimal values.

4.1. Back Propagation

- **Gradient Descent Algorithm:**

Gradient descent algorithm: Starting from a point that could be close to the solution, one will use an iterative operation to gradually approach the desired point (local minimum), i.e., when the derivative converge to 0.

In order to converge to the local minimum, one have to move in the inverse sense of the gradient vector. The formula can be as follow:

$$\text{for each } x_i \textit{ in } \mathbf{x}: \quad x_i(t+1) = x_i(t) - LR.\frac{\partial f}{\partial x_i}(\mathbf{x})$$

with LR is the learning rate

In the case of a N.N., *f* is the loss function

4.1. Back Propagation

- **The chain rule:**

When realizing a forward pass, the data is passed through layers of neurons. At each layer, the outputs are passed through the activation function before going to the next layer, … The loss function is calculated at the last layer of neurons (the output layer). Its expression can be written in function of all the parameters of the network. Example with Cross Categorical Entropy Loss:

$$L = -\sum_{k=1}^{n} y_k \log\left(\forall_{j=1}^{n_3} \frac{e^{\sum_{i=1}^{n_2}\left(\forall_{j=1}^{n_2}\max\left(0,\sum_{i=1}^{n_1}\left(\forall_{j=1}^{n_1}\max\left(0,\sum_{i=1}^{n_0} x_i\omega_{1,i,j}+b_{1,j}\right)\right)_i \omega_{2,i,j}+b_{2,j}\right)\right)_i \omega_{3,i,j}+b_{3,j}}}{\sum_{l=1}^{n_3} e^{\sum_{i=1}^{n_2}\left(\forall_{j=1}^{n_2}\max\left(0,\sum_{i=1}^{n_1}\left(\forall_{j=1}^{n_1}\max\left(0,\sum_{i=1}^{n_0} x_i\omega_{1,i,j}+b_{1,k}\right)\right)_i \omega_{2,i,j}+b_{2,k}\right)\right)_i \omega_{3,i,j}+b_{3,k}}} \right)$$

A chain of functions

4.1. Back Propagation

- **The chain rule:**

The derivative of a function chain is a product of all derivatives of all of the functions in this chain

Examples:

$$\frac{d}{dx}f(g(x)) = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx} = f'(g(x)) \cdot g'(x)$$

$$\frac{\partial}{\partial x}f(g(y,h(x,z))) = \frac{\partial f(g(y,h(x,z)))}{\partial g(y,h(x,z))} \cdot \frac{\partial g(y,h(x,z))}{\partial h(x,z)} \cdot \frac{\partial h(x,z)}{\partial x}$$

4.1. Back Propagation

- **Back Propagation:**

We want to back-propagate our gradients by calculating derivatives and partial derivatives with respect to each of our parameters and inputs. We're going to use the chain rule on our NN. We start on 1 single neuron: $ReLU(x_0w_0 + x_1w_1 + x_2w_2 + b)$

$$y = ReLU(sum(mul(x_0, w_0), mul(x_1, w_1), mul(x_2, w_2), b))$$

Calculate Gradient of the ReLU activation function:

The ReLU function is equivalent to function *max(x, 0)*. Therefore, its partial derivative is:

$$\mathbf{ReLu}(y) = \max(y, 0) \Rightarrow \frac{\partial \mathbf{ReLU}}{\partial y} = \begin{bmatrix} 1 & if & y \geq 0 \\ 0 & if & y < 0 \end{bmatrix}$$

4.1. Back Propagation

- **Back Propagation:**

drelu_dxw0: the partial derivative of the ReLU w.r.t. the first weighed input, w0x0

drelu_dxw1: the partial derivative of the ReLU w.r.t. the second weighed input, w1x1

drelu_dxw2: the partial derivative of the ReLU w.r.t. the 3rd weighed input, w2x2

drelu_db: the partial derivative of the ReLU w.r.t. the bias, w0x0

Calculate Gradient of the sum function:

The partial derivative of the sum operation is always equal to 1:

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = 1$$

$$\frac{\partial}{\partial y} f(x,y) = 1$$

4.1. Back Propagation

- **Back Propagation:**

Calculate Gradient of the multiplication function:

$$f(x,y) = x \cdot y \quad \rightarrow \quad \frac{\partial}{\partial x} f(x,y) = y$$

$$\frac{\partial}{\partial y} f(x,y) = x$$

4.1. Back Propagation

- **Back Propagation:**

Example of backpropagation on a single neuron:

```
x = [1, -2, 3] # input
w = [-3, -1, 2] # weights
b = 1 # bias
# Forward pass
xw0 = x[0] * w[0]
xw1 = x[1] * w[1]
xw2 = x[2] * w[2]

z= xw0 + xw1 + xw2 + b # the value of the chain
rule

# ReLU activation
output = max(z,0)

# The derivative of the next layer is here 1.0
dvalue = 1.0
# The derivative of the ReLU / the chain rule z
dReLU_dz = dvalue * (1. if z > 0 else 0.)
# Partial derivative of the sum, the chain rule
dsum_dxw0 = 1 # = dz_dxw0
dsum_dxw1 = 1 # = dz_dxw1
dsum_dxw2 = 1 # = dz_dxw2
dsum_db = 1

dReLU_dxw0 = dReLU_dz * dsum_dxw0
dReLU_dxw1 = dReLU_dz * dsum_dxw1
dReLU_dxw2 = dReLU_dz * dsum_dxw2
dReLu_db = dReLU_dz * dsum_db
```

4.1. Back Propagation

- **Back Propagation:**

Example of backpropagation on a single neuron:

```
# Partial derivative of the multiplication, the chain rule
dmul_dx0 = w[0]
dmul_dx1 = w[1]
dmul_dx2 = w[2]
dmul_dw0 = x[0]
dmul_dw1 = x[1]
dmul_dw2 = x[2]
dReLU_dx0 = dReLU_dxw0 * dmul_dx0
dReLU_dw0 = dReLU_dxw0 * dmul_dw0
dReLU_dx1 = dReLU_dxw1 * dmul_dx1
dReLU_dw1 = dReLU_dxw1 * dmul_dw1
dReLU_dx2 = dReLU_dxw2 * dmul_dx2
dReLU_dw2 = dReLU_dxw2 * dmul_dw2
```

```
# Determine the gradient vectors
dx = [dReLU_dx0, dReLU_dx1, dReLU_dx2] # gradient of inputs
dw = [dReLU_dw0, dReLU_dw1, dReLU_dw2] # gradient of weights
db = dReLu_db # gradient of bias
# Update the weights
w[0] += -0.001*dw[0]
w[1] += -0.001*dw[1]
w[2] += -0.001*dw[2]
b += -0.001*db
# Now, forward pass again !
xw0 = x[0] * w[0]
xw1 = x[1] * w[1]
xw2 = x[2] * w[2]

z= xw0 + xw1 + xw2 + b # the value of the chain rule
```

4.1. Back Propagation

- **Back Propagation:**

Calculate Gradient of the CCE loss: $L_i = -\sum_j y_{i,j} log(\hat{y}_{i,j})$

Li is the sample loss value, i — i-th sample in a set, j — label/output index, y — target values and y-hat — predicted values.

$$\frac{\partial L_i}{\partial \hat{y}_{i,j}} = \frac{\partial}{\partial \hat{y}_{i,j}}[-\sum_j y_{i,j} log(\hat{y}_{i,j})] = -\sum_j y_{i,j} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} log(\hat{y}_{i,j}) =$$

$$= -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot \frac{\partial}{\partial \hat{y}_{i,j}} \hat{y}_{i,j} = -\sum_j y_{i,j} \cdot \frac{1}{\hat{y}_{i,j}} \cdot 1 = -\sum_j \frac{y_{i,j}}{\hat{y}_{i,j}} = -\frac{y_{i,j}}{\hat{y}_{i,j}}$$

4.1. Back Propagation

- **Back Propagation:**

Calculate Gradient of the CCE loss:

```
def backward(self, dvalues, y_true):
        # Determine the number of samples
        samples = len(dvalues)
        # Determine the number of labels in each sample
        # We use the first sample to count
        labels = len(dvalues[0])
        # if labels are sparse, turn them into one vector
        if len(y_true.shape) == 1:
        y_true = np.eye(labels)[y_true]
        # Calculate gradient
        self.dinputs = - y_true / dvalues
        # Normalize gradient
        self.dinputs = self.dinputs / samples
```

4.1. Back Propagation

- **Back Propagation:**

Calculate Gradient of the softmax activation function:

$$S_{i,j} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}} \quad \rightarrow \quad \frac{\partial S_{i,j}}{\partial z_{i,k}} = \frac{\partial \frac{e^{z_{i,j}}}{\sum_{l=1}^{L} e^{z_{i,l}}}}{\partial z_{i,k}}$$

$$\frac{\partial S_{i,j}}{\partial z_{i,k}} = \begin{cases} S_{i,j} \cdot (1 - S_{i,k}) & j = k \\ S_{i,j} \cdot (0 - S_{i,k}) & j \neq k \end{cases} \quad \Rightarrow \quad \frac{\partial S_{i,j}}{\partial z_{i,k}} = S_{i,j} \cdot (\delta_{j,k} - S_{i,k})$$

with $$\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

4.1. Back Propagation

- **Back Propagation:**

Calculate Gradient of the softmax activation function:

```
def backward(self,dvalues):
# Create uninitialized array
self.dinputs = np.empty_like(dvalues)

# Enumerate outputs and gradients
for index, (single_output,single_dvalues) in enumerate(zip(self.output, dvalues)):
# Flatten ouput array
single_output = single_output.reshape(-1,1)
# Calculate the Jacobian Matrix of the output
jacobian_matrix = np.diagflat(single_output) - np.dot(single_output,single_output.T)

# Calculate sample-wise gradient
# and add it to the array of sample gradients
self.dinputs[index] = np.dot(jacobian_matrix,single_dvalues)
```

Artificial Neural Network

END OF CHAPTER 4.1