nne_from_scratch

January 9, 2022

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

[1]:
```
'1.21.4'
```

[2]:
```
class SGDOptimizer:
    def __init__(self,lr :float):
        self.lr = lr
    def apply(self,vector,dvector):
        return vector - dvector * self.lr

class DenseLayer():
    def __init__(self,input_nodes : int,output_nodes : int):
        self.learnable = True
        self.input_nodes = input_nodes
        self.output_nodes = output_nodes
        self.weight = np.random.randn(self.output_nodes * self.input_nodes).
 ↪reshape((self.output_nodes,self.input_nodes))
        self.bias = np.random.randn(self.output_nodes)
        self.gradient_weight = None
        self.gradient_bias = None
        self.input = None
    def forward(self,X : np.ndarray) -> np.ndarray:
        self.input = X
        return self.weight @ X + self.bias
    def backward(self,delta):
        self.gradient_weight = delta.reshape((self.output_nodes,1))  @ self.
 ↪input.reshape((1,self.input_nodes))
        self.gradient_bias = delta
        dX = self.weight.T @ delta
        return dX
    def apply(self,opt):
        self.weight = opt.apply(self.weight,self.gradient_weight)
        self.bias = opt.apply(self.bias,self.gradient_bias)
```

[3]:
```
sample_input = np.array([0.5,0.5])
dense_layer = DenseLayer(2,4)
```

[4]:
```
dense_layer.forward(sample_input)
```

[4]:
```
array([ 0.08896882, -0.13611622, -1.72671291, -0.65042457])
```

[5]:
```
dense_layer.backward(np.array([1,1,0,2]))
```

[5]:
```
array([-2.25026662,  0.37713448])
```

[40]:
```
class ReluActivation:
    def __init__(self):
        self.learnable = False
        self.input = None
        self.gradient = None
    def forward(self,X):
        self.input = X
        return np.maximum(X,0)
    def backward(self,delta):
        drelu = 1. * (self.input > 0)
        return drelu * delta

class SigmoidActivation:
    def __init__(self):
        # self.input = None
        self.learnable = False
        self.gradient = None
        self.output = None
    def forward(self,X):
        # self.input = X
        self.output =  1 / (1 + np.exp(-X))
        return self.output
    def backward(self,delta):
        dsigmoid = self.output * (1 - self.output)
        return dsigmoid * delta
```

[41]:
```
sigmoid_activation = SigmoidActivation()
sigmoid_activation.forward(np.array([-0.5,2,0]))
sigmoid_activation.backward(np.array([0.3,0.2,0.1]))
```

[41]:
```
array([0.07050111, 0.02099872, 0.025     ])
```

[42]:
```
relu_activation = ReluActivation()
relu_activation.forward(np.array([-25,0,125]))
```

[42]:
```
array([  0,   0, 125])
```

[43]:
```
class MSELoss:
    def forward(self,y_true,y_pred):
        return 0.5 * np.mean((y_true - y_pred)**2)
    def backward(self,y_true,y_pred):
        return y_pred - y_true
```

[44]:
```
mse = MSELoss()
mse.backward(np.array([1,2]),np.array([2,0]))
```

[44]:
```
array([ 1, -2])
```

[45]:
```
class NeuralNetwork:
    def __init__(self,layers,loss,opt):
        self.layers = layers
        self.loss = loss
        self.opt = opt

    def forward(self,X):
        current_input = X
        for layer in layers:
            current_input = layer.forward(current_input)
        return current_input

    def backward(self,y_true,y_pred):
        dloss = self.loss.backward(y_true,y_pred)
        current_delta = dloss
        for layer in reversed(layers):
            current_delta = layer.backward(current_delta)
    def apply(self):
        for layer in layers:
            if layer.learnable:
                layer.apply(self.opt)
    def fit(self,X,y_true):
        y_pred = self.forward(X)
        loss = self.loss.forward(y_true,y_pred)
        self.backward(y_true,y_pred)
        self.apply()
        return loss
```

[66]:
```
Xs = np.array([[0,1],[1,1],[1,0],[1,1]])
Ys = np.array([[1],[0],[1],[0]])
layers = [
    DenseLayer(2,2),
    SigmoidActivation(),
    DenseLayer(2,1),
    SigmoidActivation()
]
```

```
nn = NeuralNetwork(layers,MSELoss(),SGDOptimizer(0.3))
```

[67]:
```
nn.fit(Xs[0],Ys[0])
```

[67]:
```
0.09145182576909387
```

[68]:
```
EPOCHS = 10000
for i in range(EPOCHS):
    loss = 0
    random_inds = np.arange(4)
    np.random.shuffle(random_inds)
    for x, y in zip(Xs[random_inds],Ys[random_inds]):
        loss += nn.fit(x,y)
    loss /= Xs.shape[0]
    if i % 1000 == 0:
        print(f"EPOCH {i}: {loss:.4f}")
```

```
EPOCH 0: 0.1418
EPOCH 1000: 0.0038
EPOCH 2000: 0.0010
EPOCH 3000: 0.0005
EPOCH 4000: 0.0004
EPOCH 5000: 0.0003
EPOCH 6000: 0.0002
EPOCH 7000: 0.0002
EPOCH 8000: 0.0002
EPOCH 9000: 0.0001
```

[65]:
```
for x,y in zip(Xs,Ys):
    y_pred = nn.forward(x)
    print(f"Target={y},Predicted={y_pred}")
```

```
Target=[1],Predicted=[1.]
Target=[0],Predicted=[0.]
Target=[1],Predicted=[1.]
Target=[0],Predicted=[0.]
```

[ ]:
```

```