notes_12

December 4, 2024

1 Previous Class Definitions

The previously defined Layer_Dense, Activation_ReLU, Activation_Softmax, Loss, and Loss_CategoricalCrossEntropy classes.

[1]:
```
# imports
import matplotlib.pyplot as plt
import numpy as np
import nnfs
from nnfs.datasets import spiral_data, vertical_data
nnfs.init()
```

[2]:
```
class Layer_Dense:
    def __init__(self, n_inputs, n_neurons):
        # Initialize the weights and biases
        self.weights = 0.01 * np.random.randn(n_inputs, n_neurons)  # Normal␣
 ↪distribution of weights
        self.biases = np.zeros((1, n_neurons))

    def forward(self, inputs):
        # Calculate the output values from inputs, weights, and biases
        self.output = np.dot(inputs, self.weights) + self.biases        #␣
 ↪Weights are already transposed

class Activation_ReLU:
    def forward(self, inputs):
        self.output = np.maximum(0, inputs)

class Activation_Softmax:
    def forward(self, inputs):
        # Get the unnormalized probabilities
        # Subtract max from the row to prevent larger numbers
        exp_values = np.exp(inputs - np.max(inputs, axis=1, keepdims=True))

        # Normalize the probabilities with element wise division
        probabilities = exp_values / np.sum(exp_values, axis=1,keepdims=True)
        self.output = probabilities
```

```python
# Base class for Loss functions
class Loss:
    '''Calculates the data and regularization losses given
    model output and ground truth values'''
    def calculate(self, output, y):
        sample_losses = self.forward(output, y)
        data_loss = np.average(sample_losses)
        return data_loss

class Loss_CategoricalCrossEntropy(Loss):
    def forward(self, y_pred, y_true):
        '''y_pred is the neural network output
        y_true is the ideal output of the neural network'''
        samples = len(y_pred)
        # Bound the predicted values
        y_pred_clipped = np.clip(y_pred, 1e-7, 1-1e-7)

        if len(y_true.shape) == 1:      # Categorically labeled
            correct_confidences = y_pred_clipped[range(samples), y_true]
        elif len(y_true.shape) == 2:    # One hot encoded
            correct_confidences = np.sum(y_pred_clipped*y_true, axis=1)

        # Calculate the losses
        negative_log_likelihoods = -np.log(correct_confidences)
        return negative_log_likelihoods
```

2 Backpropagation of a Single Neuron

Backpropagation helps us find the gradient of the neural network with respect to each of the parameters (weights and biases) of each neuron.

Imagine a layer that has 3 inputs and 1 neuron. There are 3 inputs (x0, x1, x2), three weights (w0, w1, w2), 1 bias (b0), and 1 output (z). There is a ReLU activation layer after the neuron output going into a square loss function (loss = z^2).

Loss = (ReLU(sum(mul(x0, w0), mul(x1, w1), mul(x2, w2(, b0)))))^2

$\frac{\delta Loss()}{\delta w0} = \frac{\delta Loss()}{\delta ReLU()} * \frac{\delta ReLU()}{\delta sum()} * \frac{\delta sum()}{\delta mul(x0,w0)} * \frac{\delta mul(x0,w0)}{\delta w0}$

$\frac{\delta Loss()}{\delta ReLU()} = 2 * ReLU(sum(...))$

$\frac{\delta ReLU()}{\delta sum()}$ = 0 if sum(...) is less than 0 and 1 if sum(...) is greater than 0

$\frac{\delta sum()}{\delta mul(x0,w0)} = 1$

$\frac{\delta mul(x0,w0)}{\delta w0} = x0$

This is repeated for w0, w1, w2, b0.

We then use numerical differentiation to approximate the gradient. Then, we update the parameters using small step sizes, such that $w0[i+1] = w0[i] - step * \frac{\delta Loss()}{\delta w0}$

[1]:
```
import numpy as np

# Initial parameters
weights = np.array([-3.0, -1.0, 2.0])
bias = 1.0
inputs = np.array([1.0, -2.0, 3.0])
target_output = 0.0
learning_rate = 0.001

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return np.where(x > 0, 1.0, 0.0)

for iteration in range(200):
    # Forward pass
    linear_output = np.dot(weights, inputs) + bias
    output = relu(linear_output)
    loss = (output - target_output) ** 2

    # Backward pass to calculate gradient
    dloss_doutput = 2 * (output - target_output)
    doutput_dlinear = relu_derivative(linear_output)
    dlinear_dweights = inputs
    dlinear_dbias = 1.0

    dloss_dlinear = dloss_doutput * doutput_dlinear
    dloss_dweights = dloss_dlinear * dlinear_dweights
    dloss_dbias = dloss_dlinear * dlinear_dbias

    # Update weights and bias
    weights -= learning_rate * dloss_dweights
    bias -= learning_rate * dloss_dbias

    # Print the loss for this iteration
    print(f"Iteration {iteration + 1}, Loss: {loss}")

print("Final weights:", weights)
print("Final bias:", bias)
```

```
Iteration 1, Loss: 36.0
Iteration 2, Loss: 33.872399999999985
Iteration 3, Loss: 31.870541159999995
Iteration 4, Loss: 29.98699217744401
```

```
Iteration 5, Loss: 28.21476093975706
Iteration 6, Loss: 26.54726856821742
Iteration 7, Loss: 24.978324995835766
Iteration 8, Loss: 23.502105988581878
Iteration 9, Loss: 22.113131524656684
Iteration 10, Loss: 20.80624545154949
Iteration 11, Loss: 19.576596345362915
Iteration 12, Loss: 18.419619501351963
Iteration 13, Loss: 17.331019988822064
Iteration 14, Loss: 16.306756707482677
Iteration 15, Loss: 15.343027386070442
Iteration 16, Loss: 14.43625446755368
Iteration 17, Loss: 13.583071828521266
Iteration 18, Loss: 12.780312283455652
Iteration 19, Loss: 12.024995827503426
Iteration 20, Loss: 11.314318574097976
Iteration 21, Loss: 10.645642346368787
Iteration 22, Loss: 10.016484883698395
Iteration 23, Loss: 9.424510627071816
Iteration 24, Loss: 8.867522049011871
Iteration 25, Loss: 8.34345149591527
Iteration 26, Loss: 7.850353512506679
Iteration 27, Loss: 7.386397619917536
Iteration 28, Loss: 6.949861520580408
Iteration 29, Loss: 6.539124704714106
Iteration 30, Loss: 6.152662434665503
Iteration 31, Loss: 5.789040084776769
Iteration 32, Loss: 5.446907815766464
Iteration 33, Loss: 5.124995563854669
Iteration 34, Loss: 4.822108326030859
Iteration 35, Loss: 4.537121723962434
Iteration 36, Loss: 4.268977830076255
Iteration 37, Loss: 4.016681240318748
Iteration 38, Loss: 3.7792953790159096
Iteration 39, Loss: 3.55593902211607
Iteration 40, Loss: 3.345783025909011
Iteration 41, Loss: 3.148047249077789
Iteration 42, Loss: 2.9619976566572896
Iteration 43, Loss: 2.786943595148845
Iteration 44, Loss: 2.622235228675548
Iteration 45, Loss: 2.4672611266608238
Iteration 46, Loss: 2.3214459940751673
Iteration 47, Loss: 2.1842485358253243
Iteration 48, Loss: 2.055159447358047
Iteration 49, Loss: 1.9336995240191863
Iteration 50, Loss: 1.8194178821496518
Iteration 51, Loss: 1.7118902853146072
Iteration 52, Loss: 1.6107175694525138
```

```
Iteration 53, Loss: 1.5155241610978685
Iteration 54, Loss: 1.4259566831769857
Iteration 55, Loss: 1.3416826432012259
Iteration 56, Loss: 1.2623891989880334
Iteration 57, Loss: 1.18778199732784
Iteration 58, Loss: 1.1175840812857638
Iteration 59, Loss: 1.0515348620817762
Iteration 60, Loss: 0.9893891517327436
Iteration 61, Loss: 0.930916252865338
Iteration 62, Loss: 0.8758991023209965
Iteration 63, Loss: 0.8241334653738256
Iteration 64, Loss: 0.775427177570232
Iteration 65, Loss: 0.7295994313758314
Iteration 66, Loss: 0.6864801049815188
Iteration 67, Loss: 0.6459091307771113
Iteration 68, Loss: 0.6077359011481849
Iteration 69, Loss: 0.5718187093903269
Iteration 70, Loss: 0.538024223665358
Iteration 71, Loss: 0.5062269920467352
Iteration 72, Loss: 0.4763089768167732
Iteration 73, Loss: 0.44815911628690125
Iteration 74, Loss: 0.4216729125143454
Iteration 75, Loss: 0.3967520433847474
Iteration 76, Loss: 0.3733039976207088
Iteration 77, Loss: 0.35124173136132447
Iteration 78, Loss: 0.3304833450378703
Iteration 79, Loss: 0.3109517793461324
Iteration 80, Loss: 0.29257452918677557
Iteration 81, Loss: 0.275283374511837
Iteration 82, Loss: 0.2590141270781873
Iteration 83, Loss: 0.24370639216786646
Iteration 84, Loss: 0.22930334439074573
Iteration 85, Loss: 0.21575151673725296
Iteration 86, Loss: 0.20300060209808138
Iteration 87, Loss: 0.1910032665140845
Iteration 88, Loss: 0.17971497346310233
Iteration 89, Loss: 0.16909381853143318
Iteration 90, Loss: 0.159100373856225
Iteration 91, Loss: 0.14969754176132244
Iteration 92, Loss: 0.1408504170432283
Iteration 93, Loss: 0.13252615739597354
Iteration 94, Loss: 0.1246938614938715
Iteration 95, Loss: 0.11732445427958361
Iteration 96, Loss: 0.11039057903166032
Iteration 97, Loss: 0.10386649581088914
Iteration 98, Loss: 0.09772798590846545
Iteration 99, Loss: 0.09195226194127527
Iteration 100, Loss: 0.08651788326054573
```

```
Iteration 101, Loss: 0.08140467635984756
Iteration 102, Loss: 0.07659365998698067
Iteration 103, Loss: 0.07206697468175016
Iteration 104, Loss: 0.06780781647805846
Iteration 105, Loss: 0.06380037452420505
Iteration 106, Loss: 0.060029772389824425
Iteration 107, Loss: 0.05648201284158581
Iteration 108, Loss: 0.05314392588264792
Iteration 109, Loss: 0.05000311986298341
Iteration 110, Loss: 0.04704793547908098
Iteration 111, Loss: 0.044267402492267266
Iteration 112, Loss: 0.04165119900497416
Iteration 113, Loss: 0.03918961314378044
Iteration 114, Loss: 0.03687350700698295
Iteration 115, Loss: 0.03469428274287037
Iteration 116, Loss: 0.032643850632766785
Iteration 117, Loss: 0.030714599060370343
Iteration 118, Loss: 0.028899366255902458
Iteration 119, Loss: 0.027191413710178605
Iteration 120, Loss: 0.025584401159906987
Iteration 121, Loss: 0.02407236305135653
Iteration 122, Loss: 0.02264968639502141
Iteration 123, Loss: 0.02131108992907558
Iteration 124, Loss: 0.020051604514267202
Iteration 125, Loss: 0.018866554687474092
Iteration 126, Loss: 0.01775154130544445
Iteration 127, Loss: 0.01670242521429262
Iteration 128, Loss: 0.015715311884128023
Iteration 129, Loss: 0.014786536951776045
Iteration 130, Loss: 0.01391265261792606
Iteration 131, Loss: 0.013090414848206555
Iteration 132, Loss: 0.01231677133067759
Iteration 133, Loss: 0.011588850145034609
Iteration 134, Loss: 0.01090394910146302
Iteration 135, Loss: 0.010259525709566512
Iteration 136, Loss: 0.00965318774013127
Iteration 137, Loss: 0.009082684344689475
Iteration 138, Loss: 0.008545897699918257
Iteration 139, Loss: 0.008040835145853137
Iteration 140, Loss: 0.00756562178873318
Iteration 141, Loss: 0.007118493541019131 4
Iteration 142, Loss: 0.006697790572744897
Iteration 143, Loss: 0.0063019511498957235
Iteration 144, Loss: 0.005929505836936862 5
Iteration 145, Loss: 0.005579072041973895
Iteration 146, Loss: 0.005249348884293221
Iteration 147, Loss: 0.004939112365231496
Iteration 148, Loss: 0.0046472108244463226
```

```
Iteration 149, Loss: 0.004372560664721515
Iteration 150, Loss: 0.004114142329436494
Iteration 151, Loss: 0.0038709965177668067
Iteration 152, Loss: 0.003642220623566796
Iteration 153, Loss: 0.003426965384714043
Iteration 154, Loss: 0.0032244317304774253
Iteration 155, Loss: 0.003033867815206219
Iteration 156, Loss: 0.0028545662273275238
Iteration 157, Loss: 0.002685861363292454
Iteration 158, Loss: 0.002527126956721865
Iteration 159, Loss: 0.0023777737535795648
Iteration 160, Loss: 0.002237247324743051
Iteration 161, Loss: 0.0021050260078507234
Iteration 162, Loss: 0.001980618970786757
Iteration 163, Loss: 0.001863564389613244
Iteration 164, Loss: 0.0017534277341871227
Iteration 165, Loss: 0.001649800155096659
Iteration 166, Loss: 0.0015522969659304577
Iteration 167, Loss: 0.0014605562152439574
Iteration 168, Loss: 0.001374237342923055
Iteration 169, Loss: 0.0012930199159562866
Iteration 170, Loss: 0.0012166024389232565
Iteration 171, Loss: 0.0011447012347829103
Iteration 172, Loss: 0.0010770493918072343
Iteration 173, Loss: 0.0010133957727514104
Iteration 174, Loss: 0.0009535040825818146
Iteration 175, Loss: 0.0008971519913012098
Iteration 176, Loss: 0.0008441303086153165
Iteration 177, Loss: 0.0007942422073761319
Iteration 178, Loss: 0.0007473024929202092
Iteration 179, Loss: 0.0007031369155886454
Iteration 180, Loss: 0.0006615815238773228
Iteration 181, Loss: 0.0006224820558161947
Iteration 182, Loss: 0.0005856933663174615
Iteration 183, Loss: 0.0005510788883681067
Iteration 184, Loss: 0.0005185101260655349
Iteration 185, Loss: 0.0004878661776150635
Iteration 186, Loss: 0.00045903328651800607
Iteration 187, Loss: 0.0004319044192847727
Iteration 188, Loss: 0.00040637886810505474
Iteration 189, Loss: 0.0003823618770000461
Iteration 190, Loss: 0.00035976429006934636
Iteration 191, Loss: 0.00033850222052625716
Iteration 192, Loss: 0.0003184967392931672
Iteration 193, Loss: 0.0002996735820009388
Iteration 194, Loss: 0.000281962873304691
Iteration 195, Loss: 0.0002652988674923804
Iteration 196, Loss: 0.0002496197044235683
```

```
Iteration 197, Loss: 0.00023486717989213552
Iteration 198, Loss: 0.00022098652956051033
Iteration 199, Loss: 0.0002079262256634926
Iteration 200, Loss: 0.00019563778572677975
Final weights: [-3.3990955  -0.20180899  0.80271349]
Final bias: 0.6009044964039992
```