Deep Learning Assignment 1

Josip Grguric

November 14, 2021

1 Part 1

1.1 Question 1

In order to be able to back propagate we need to calculate the derivatives for each of the formulae used in our forward propagation. Most of the results are already given to us in the presentations with two exceptions. Our loss function and the soft-max calculation. For our loss function the derivative was simple, while for our softmax function we had to make two separate derivatives based on whether $i == j$

$$\frac{dl}{dy_c} = -\frac{1}{y_c}$$

Next we have the derivative of softmax in which $i! = j$

$$\frac{dy}{do_j} = \frac{d}{do_j}\frac{e^{o_i}}{\sum_j e^{o_j}}$$
$$\frac{0 * \sum_j e^{o_j} - e^{o_i}e^{o_i}}{(\sum_j e^{o_j})^2}$$
$$\frac{-e^{o_i}}{\sum_j e^{o_j}}\frac{e^{o_j}}{\sum_j e^{o_j}}$$
$$-softmax(i)\,softmax(j)$$

Lastly we have the derivative of softmax in which $i == j$

$$\frac{dy}{do_j} = \frac{d}{do_j}\frac{e^{o_i}}{\sum_j e^{o_j}}$$
$$\frac{e_{o_i}\sum_j e^{o_j} - e^{o_i}e^{o_i}}{(\sum_j e^{o_j})^2}$$
$$\frac{e^{o_i}}{\sum_j e^{o_j}}(\frac{\sum_j e^{o_j}}{\sum_j e^{o_j}} - \frac{e^{o_j}}{\sum_j e^{o_j}})$$
$$softmax(i)(1 - softmax(j))$$

1.2 Question 2

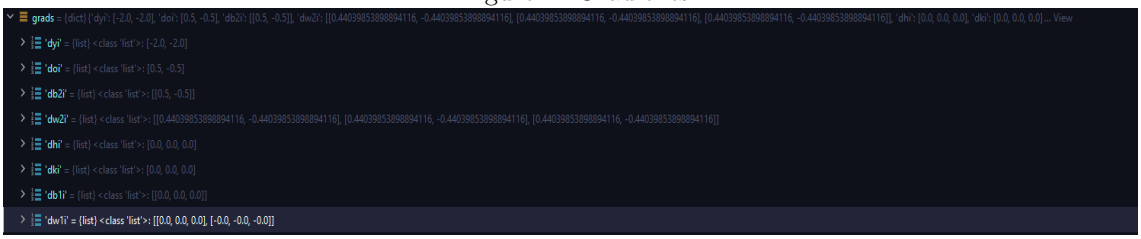
To calculate the derivative $\frac{dl}{do_i}$ we first need to apply the chain rule and calculate the derivatives of $\frac{dl}{dy}$ and $\frac{dy}{do_j}$. Given the fact that we have already calculated said derivatives, we therefore do not need to calculate the derivative of $\frac{dl}{do_i}$ separately.

1.3 Question 3

The result of a single forward and backward pass can be seen in table 1

The implementations:

Figure 1: Gradients



Name	Value
dyi	[-2.0, -2.0]
doi	[0.5, -0.5]
db2i	[0.5, -0.5]
dw2i	[[0.440, -0.440], [0.440, -0.440], [0.440, -0.440]]
dhi	[0, 0, 0]
dki	[0, 0, 0]
db1i	[0, 0, 0]
dw1i	[0, 0, 0], [0, 0, 0]

Table 1: An example table.

Figure 2: Forward propagation

```
def scalar_forward(parameters):
    #Calculating up to h
    params = parameters
    for i in range(len(params["k"])):
        params["k"][i] = 0
        for j in range(len(params["inputs"])):
            params["k"][i] += params["inputs"][j]*params['w1'][j][i]
        params["k"][i] += params["b1"][i]
        params["h"][i] = sigmoid(params["k"][i])
    #Calculating output
    for i in range(len(params["pre_soft"])):
        params["pre_soft"][i] = 0
        for j in range(len(params["h"])):
            params["pre_soft"][i] += params["h"][j]*params['w2'][j][i]
        params["pre_soft"][i] += params["b2"][i]
    # Lastly, calculating softmax activation
    params["loss"] = 0
    for i in range(len(params["pre_soft"])):
        params["softmax"][i] = softmax(params["pre_soft"][i], params["pre_soft"])
        params["li"][i] = loss(params["softmax"][i], params["true_c"], i)
        params["loss"] += params["li"][i]
    return params
```

1.4 Question 4

The resulting loss over time can be seen in figure 5 in which the average loss is decreased with each epoch. The reason this graph is not smoother is due to the high learning rate of 0.9. With a lower learning rate there is less epochs needed to make this work.

1.5 Question 5 & 6

The implementations of both can be found in the code given, however they are flawed and do not work as intended.

For part 5 the derivatives are for the most part the same the same, with the exception of the softmax derivative which was supposed to be a Jacobian.

For part 6 I've managed to use matmul to create a proper batched forward propagation, which can

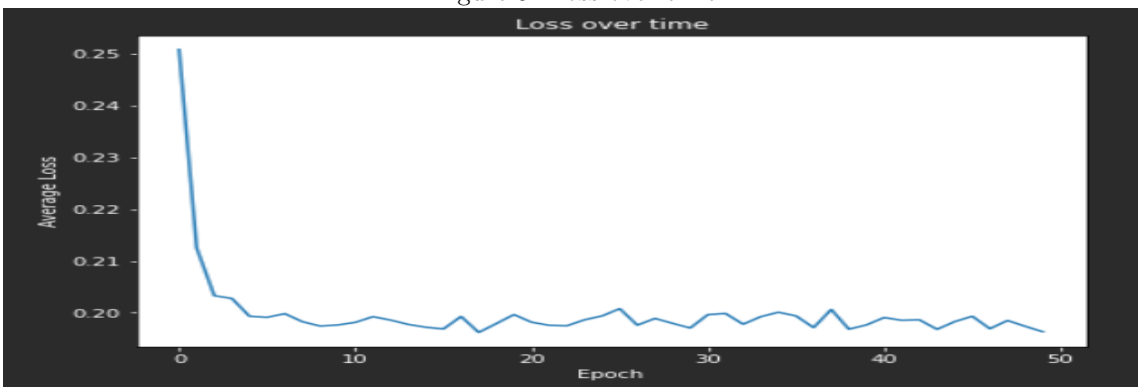
Figure 3: Backward propagation

```
def scalar_backward(params, y=None):
    grads = {"dyi": [],"doi": [],"db2i": [], "dw2i": [],"dhi": [],"dki": [],"db1i": [],"dw1i": []}
    #dl/dyi = dl/dli * dli/dyi
    for i in range(len(params["softmax"])):
        grads["dyi"].append(-1/(params["softmax"][i]))
    #dl/doi = dl/dyi * dyi/doi
    if y:
        grads["doi"].append(grads["dyi"][0]*((params["softmax"][0])*(1-(params["softmax"][0]))))
        grads["doi"].append(grads["dyi"][0]*((params["softmax"][0])*(0-(params["softmax"][1]))))
    else:
        grads["doi"].append(grads["dyi"][1]*((params["softmax"][1])*(0-(params["softmax"][0]))))
        grads["doi"].append(grads["dyi"][1]*((params["softmax"][1])*(1-(params["softmax"][1]))))
    #dl/db2 = dl/doi * doi/db2
    grads["db2i"].append([grads["doi"][0],
                          grads["doi"][1]])
    #dl/dw2 = dl/doi * doi/dw2i
    for i in range(len(params["h"])):
        grads["dw2i"].append([grads["doi"][0] * params["h"][i],
                              grads["doi"][1] * params["h"][i]])

    #dl/dh = dl/doi * doi/dhi
    for i in range(len(params["h"])):
        grads["dhi"].append(grads["doi"][0]*params["w2"][i][0] +
                              grads["doi"][1]*params["w2"][i][1])
    #dl/dk = dl/dh * dhi/dki
    for i in range(len(params["k"])):
        grads["dki"].append(grads["dhi"][i]*(params["h"][i]*(1-params["h"][i])))
    #dl/dw1
    for i in range(len(params["inputs"])):
        grads["dw1i"].append([grads["dki"][0] * params["inputs"][i],
                              grads["dki"][1] * params["inputs"][i],
                              grads["dki"][2] * params["inputs"][i]])
```

Figure 4: Gradient Descent

```
def gradient_descent(parameters, grads, learning_rate = 0.9):
    params = parameters
    for i in range(len(parameters["w1"])):
        for j in range(len(parameters["w1"][i])):
            params["w1"][i][j] = params["w1"][i][j] - learning_rate*grads["dw1i"][i][j]
            if i == 0:
                params["b1"][j] = params["b1"][j] - learning_rate*grads["db1i"][i][j]
    for i in range(len(parameters["w2"])):
        for j in range(len(parameters["w2"][i])):
            params["w2"][i][j] = params["w2"][i][j] - learning_rate*grads["dw2i"][i][j]
            if i == 0:
                params["b2"][j] = params["b2"][j] - learning_rate*grads["db2i"][i][j]

    return params
```

Figure 5: Loss over time



be seen under the batched_tensor_forward function in python.

After extensive debugging I've noticed the problem was caused in my implementation of tensor back propagation. Had I paid attention to this last week when I originally implemented the majority of this code, this would not have been a problem.

1.6 Question 7

This question was a failure due to issues found in implementation of tensor propagation.