Machine Learning (E23) Assignment 2

Tobias Washeim: 202007525, Alexander Myrtoft Skjødt: 202004110

November 8th, 2023

Contents

Part I

Derivative

The loss function we are working with is denoted as

$$L(z) = -\sum_{i=1}^{k} y_i \ln(\text{softmax}(z)_i), \tag{1}$$

where $z \in \mathbb{R}^k$ is the input and y_i is the output that can take the values 0 and 1. We let the output be $y_i = \delta_{i,j}$ where $\delta_{i,j}$ is the delta function.

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_{a=1}^{k} e^{z_a}}. \tag{2}$$

We will be using both eq. (1) and eq. (2) to calculate the derivative of the loss function with respect to z_i. It is determined below.

$$\begin{aligned}
\frac{\partial L}{\partial z_i} &= \frac{\partial}{\partial z_i}\left[-\sum_{i=1}^{k} y_i \ln(\text{softmax}(z)_i\right] \\
&= \frac{\partial}{\partial z_i}\left[-\delta_{i,j} \ln(\text{softmax}(z)_i)\right] \\
&= -\delta_{i,j} \frac{\partial \ln(\text{softmax}(z)_i)}{\partial \text{softmax}(z)_i} \frac{\partial \text{softmax}(z)_i}{\partial z_i} \\
&= -\delta_{i,j} \frac{1}{\text{softmax}(z)_i} \frac{\partial}{\partial z_i}\left[\frac{e^{z_i}}{\sum_{a=1}^{k} e^{z_a}}\right] \\
&= -\delta_{i,j} \frac{1}{\text{softmax}(z)_i}\left[\frac{e^{z_i}}{\sum_{a=1}^{k} e^{z_a}} - \left(\frac{e^{z_i}}{\sum_{a=1}^{k} e^{z_a}}\right)^2\right] \\
&= -\delta_{i,j} + \text{softmax}(z)_i.
\end{aligned}$$

We have now shown the derivative of the loss function with respect to z_i above.

Part II

Implementation and Code

When we run our implementation of the neural net with one hidden layer we obtain an in-sample accuracy of 99.7%, and get an accuracy of 98.1% using the training set.

In the code below we showcase the forward and backward pass. First, We implement the cost function expression from part I. This is done in line 6. Secondly, we add a decay term in line 5.

We also coded the backward pass. For this we calculated

$$
\begin{aligned}
d_{\mathrm{w1}} &= \frac{\partial L}{\partial w1} = d_{\mathrm{cost}} \cdot w_2 \cdot I_{[w_1 X + b_1 > 0]} \cdot X + 2\lambda w_1, \\
d_{\mathrm{b1}} &= \frac{\partial L}{\partial b1} = d_{\mathrm{cost}} \cdot w_2 \cdot I_{[w_1 X + b_1 > 0]}, \\
d_{\mathrm{w2}} &= \frac{\partial L}{\partial w2} = d_{\mathrm{cost}} \cdot \mathrm{Relu}(b_1 + w_1 X) + 2\lambda w_2, \\
d_{\mathrm{b2}} &= \frac{\partial L}{\partial b2} = d_{\mathrm{cost}},
\end{aligned}
$$

where $d_{\mathrm{cost}} = \frac{\partial L}{\partial \mathrm{cost}}$. In our code we refered d_{cost} to as the variable 'd_out'.

```
    ### forward pass ###
    hidden_layer_in = X@W1+b1
    hidden_layer_out = relu(hidden_layer_in)
    out = hidden_layer_out@W2+b2
    decay = c*(np.sum(np.sum(W1**2))+np.sum(np.sum(W2**2)))
    cost = -np.log(softmax(out))*y_in_k
    cost_out = np.sum(cost)/len(cost) + decay

    ### backward pass ###
    d_out = softmax(out)-y_in_k

    # indicator function, derivative of relu()
    relu_indicator = np.array(hidden_layer_out,copy=True,dtype=bool)
    relu_indicator = np.array(relu_indicator,dtype=int)

    # all are normalized over n (# datapoints)
    # because the dot product sums over datapoints
    d_w1 = (np.multiply((d_out@W2.T),relu_indicator).T@X).T/len(X) + c*2*W1
    d_b1 = (np.multiply((d_out@W2.T),relu_indicator).T
            @np.ones((len(X),1))).T / len(X)
    d_w2 = (d_out.T@hidden_layer_out).T/len(X) + c*2*W2
    d_b2 = (d_out.T@np.ones((len(X),1))).T / len(X)
```

Result of test

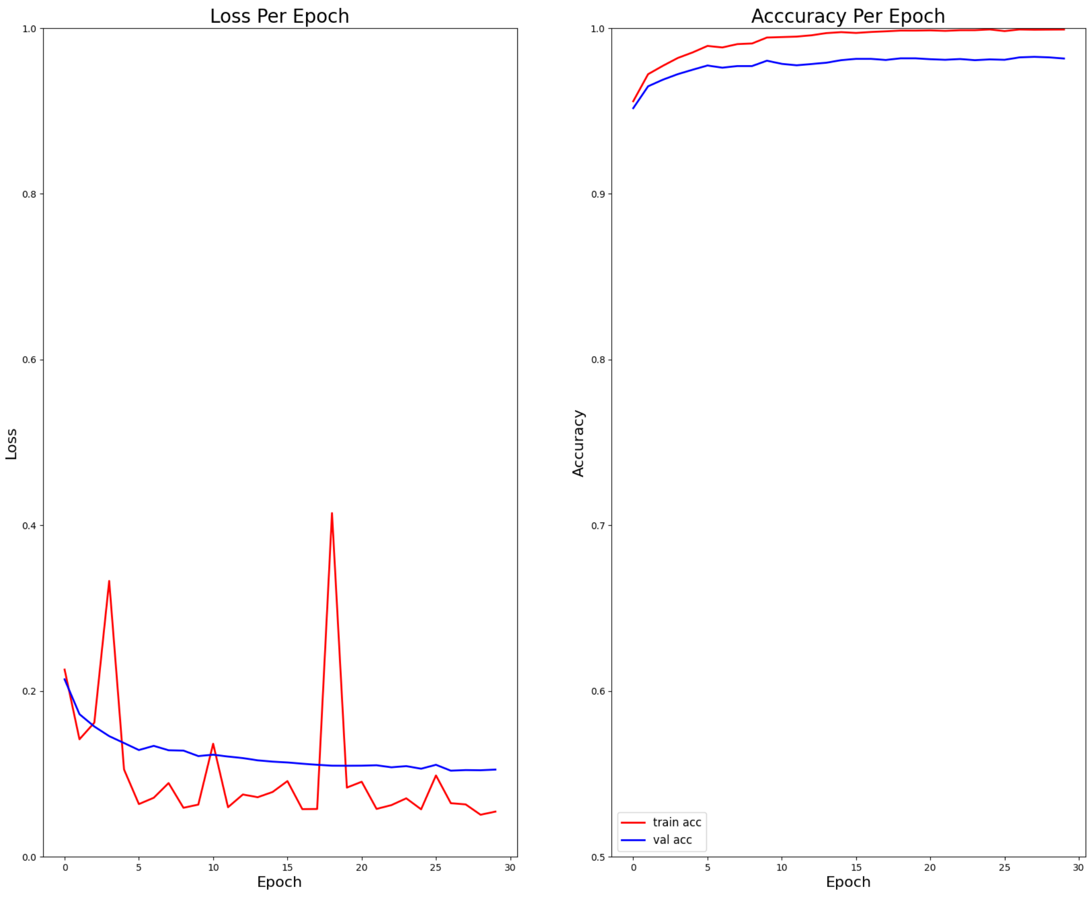

Figure 1: The loss/cost of the training data spikes more than expected, it does however seem to settle toward the end. The cost of the validation data is converging very nicely, and the same goes for both the in-sample and validation accuracy. The final validation accuracy was 98.1% and the in-sample accuracy was 99.7%