Course: Deep Learning **(14/11/2021)**

Submission Assignment 1

Coordinator: Jakub Tomczak *Name:* Bing Tan, *Netid:* 2650219/btn630

1 Question answers

Question 1 *Work out the local derivatives of the cross-entropy loss and the softmax activation. Also, show their derivations.*
The target class is given as an integer value. Therefore, the true class is noted as class $c \in C$.

$$\frac{\partial l}{\partial y_i} = \begin{cases} -\frac{1}{y_i} & \text{if i = c} \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

To slim down the equations, we define $\Sigma_C = \sum_{j=1}^{C} e^{o_j}$.

$$\text{if } i = j : \frac{\partial y_i}{\partial o_i} = \frac{\partial \frac{e^{o_i}}{\Sigma_C}}{\partial o_i} = \frac{e^{o_i}\Sigma_C - e^{o_i}e^{o_i}}{\Sigma_C^2} = \frac{e^{o_i}}{\Sigma_C}\frac{\Sigma_C - e^{o_i}}{\Sigma_C} = \frac{e^{o_i}}{\Sigma_C}(1 - \frac{e^{o_i}}{\Sigma_C}) = y_i(1 - y_i) \tag{1.2}$$

$$\text{if } i \neq j : \frac{\partial y_i}{\partial o_j} = \frac{\partial \frac{e^{o_i}}{\Sigma_C}}{\partial o_j} = \frac{0 - e^{o_i}e^{o_j}}{\Sigma_C^2} = -\frac{e^{o_i}}{\Sigma_C}\frac{e^{o_j}}{\Sigma_C} = -y_i y_j \tag{1.3}$$

Next, equations 1.2 and 1.3 can be combined to create the Jacobian:

$$\frac{\partial y}{\partial o} = \begin{bmatrix} y_0(1-y_0) & -y_1 y_0 & \cdots & -y_C y_0 \\ -y_0 y_1 & y_1(1-y_1) & & \\ \vdots & & \ddots & \\ -y_0 y_C & & & y_C(1-y_C) \end{bmatrix} \tag{1.4}$$

Question 2 *Why is equation 1.2 unnecessary provided 1.1-1.3?*
Combining the cross entropy loss with the softmax activation gives us:

$$\begin{aligned} \frac{\partial l}{\partial o_i} &= -\sum_{j=1}^{C} \frac{\partial t_j \log(y_j)}{\partial o_i} = -\sum_{j=1}^{C} t_j \frac{\partial \log(y_j)}{\partial o_i} \\ &= -\sum_{j=1}^{C} t_j \frac{1}{y_j}\frac{\partial y_j}{\partial o_i} = -\frac{t_i}{y_i}\frac{\partial y_i}{\partial o_i} - \sum_{j\neq i}^{C} \frac{t_j}{y_j}\frac{\partial y_j}{\partial o_i} \\ = -\frac{t_i}{y_i} y_i(1-y_i) - \sum_{j\neq i}^{C} \frac{t_j}{y_j}(-y_j y_i) &= -t_i + t_i y_i + \sum_{j\neq i}^{C} t_j y_i = -t_i + \sum_{j=1}^{C} t_j y_i = -t_i + y_i \sum_{j=1}^{C} t_j \\ &= y_i - t_i \end{aligned} \tag{1.5}$$

Therefore, both cases don't need to be considered as the derivative is the softmax activation itself minus t_i, where $t_i = 1$ iff i = c, otherwise 0.

Question 3 *Report all the derivatives used in the neural network*
The neural network used is a two layer model with the following notation:

The input layer consists of:

- input x
- weights W
- bias b
- linear forward z = $W^T x + b$
- activation h = $\sigma(z)$

The hidden layer consists of:

- weights V
- bias c
- linear forward o = $V^T h + c$
- activation y = softmax(o)

The important derivates are those of the loss with respect to V, c, W and b respectively. Below, the mathematical formulation of gradients is presented:

$$\frac{\partial l}{\partial o_j} = y_i - t_i \tag{1.6}$$

$$\frac{\partial l}{\partial c} = \frac{\partial l}{\partial o_j} \tag{1.7}$$

$$\frac{\partial l}{\partial V} = \frac{\partial l}{\partial o_j} h^T \tag{1.8}$$

$$\frac{\partial l}{\partial h} = V^T \frac{\partial l}{\partial o_j} \tag{1.9}$$

$$\frac{\partial l}{\partial z} = \frac{\partial l}{\partial h} * h * (1 - h) \tag{1.10}$$

$$\frac{\partial l}{\partial W} = \frac{\partial l}{\partial z} * x^T \tag{1.11}$$

$$\frac{\partial l}{\partial b} = \frac{\partial l}{\partial z} \tag{1.12}$$

The equations in code form for the Stochastic Gradient Descent can be found in chapter 6. The vectorized version for Mini Batch Gradient Descent is also done but can be found in the notebook. Important to note, the W, b, V and c parameters are initialized as column vectors, hence why the implementation in chapter 6 misses a transpose, as it is not needed.

Question 4 *Show that the training loss drops as training progresses* In this section, data of an overlapping elipse and a circle is generated. The model from question 3 is trained with this data. The code implementation can be found in chapter 6. The generated loss curve can be found at figure 1. Here, the mean loss is shown per 1000 observations. First, the loss decreases rapidly and finally it slows down as the model acquires a good fit on the training data.

2 Problem statement

The MNIST data set of handwritten digits is evaluated using a neural network consisting of 2 layers. For the input layer, a fully connected approach is used, where the entire imagine is flattened and fed into the network. The second layer consist of 300 hidden nodes using the sigmoid activation. For the output activation, the softmax is used with a cross entropy loss function. In this report, the performance of the neural network is assessed using stochastic gradient descent. In the provided notebook, the implementation for the minibatch gradient descent can also be found.

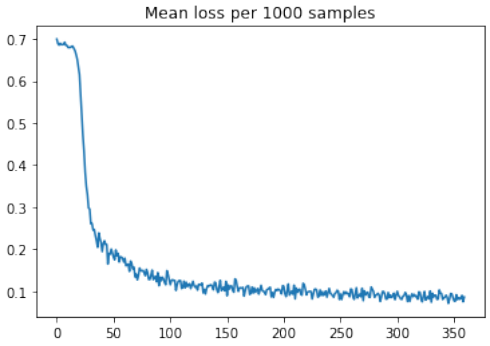

Figure 1: Loss curve Q4

3 Methodology

The neural network is built using equations 1.1 - 1.11. The gradients are then updated using the update rule below: $w_{new} = w + \alpha \nabla w$, where α represents the learning rate. In the next chapter, the different experiments are elaborated with which the model is evaluated.

4 Experiments

To make measurements fair, all experiments are done with 5 epochs.

1. For a fixed learning rate, the training and validation loss is plotted. What does this tell?
2. The average and standard deviation of the loss is plotted for SGD when run for three times. What does this tell?
3. The learning rates 0.001, 0.003, 0.01 and 0.03 are all plotted against each other. How does the learning rate influence the final performance?
4. The best model from the previous experiments is trained on all data instead of a fraction and the accuracy is reported.

5 Results and discussion

In experiment 1, the training loss is depicted against the validation loss for five epochs. Notably, the validation loss is lower than the training loss. After each epoch, both losses keep decreasing, therefore the model is acquiring a better fit for all iterations. Overfitting is indicated by a higher validation loss than the training loss, which is luckily not the case. To conclude, the model has a good fit for a learning rate of 0.01 and an accuracy of 91.2%.

Experiment 2 aims to demonstrate the rate of convergence of the Stochastic Gradient Descent method. The model is trained three times and all training losses are put together. The mean and standard deviation of the loss is plotted per epoch in figure 2. The confidence of the initial epoch is very low, hence the wide spread. As training progresses, the mean loss and the spread decrease. In other words, the model becomes more confident that the mean loss resides within a certain range.

The model is run multiple times with different learning rates. The reported validation accuracies are 89.82%, 94.8%, 92.60% and 96.14% for learning rates 0.001, 0.003, 0.01 and 0.03 respectively. The graphs of the mean training and validation loss for 5 epochs is depicted in appendix figures 3, 4, 5 and 6. Interestingly, all learning rates higher than 0.001 eventually report a bigger validation loss compared to the training loss. Due to the cross entropy loss function, bad predictions are punished more than good predictions are rewarded. In the case of 0.003 learning rate, the validation loss is slightly above the training, but the accuracy is much higher. Indicating, that for learning rate 0.001 an extra epoch could benefit the final performance.
The output of our layer has a softmax activation, meaning the output is modelled as a probability that the observation belongs to a certain class. This value lies between (0,1], hence $-log(y_c)$ goes to infinity when y_c approaches 0. In short, the model tends to overfit when the validation loss is higher than the training loss.

Thus, incorrect predictions tend to be farther off and the model does not generalize anymore.

The final experiment demonstrates the full power of the model with the best found hyperparameters based on the three previous experiments. Experiment 3 showed the trade off between learning rates 0.001 and 0.003. The latter reported a much higher accuracy for only a slight probable overfit. Therefore, 0.003 is used for the final model. Because of stochastic gradient descent and the training on the entire data set, the fit of the model will not be similar when retrained. The final model has an accuracy of 91.55%. In figure 7 is the mean validation and training loss depicted. The final model has a slightly lower training loss compared to the validation loss, but a higher accuracy compared to a model with learning rate 0.001. In practice, the model could be overfit for use in practice. However, figure 3 indicates in our opinion that there is a slight improvement to be made. It could be that learning rate 0.001 with 6 or 7 epochs could outperform the current final model. To conclude, the model works well, but if the model were to be used in practice, further evaluation respective with the use case should be mandatory.

As a final note, mini batch gradient descent works slightly faster than stochastic gradient descent, especially if the batch size is chosen relatively big. However, the model averages over the derivatives of the entire batch, therefore the learning is slower and a higher learning rate should be chosen. A reason to choose mini batch gradient descent could be to more closely approximate the optimum model, as stochastic gradient descent tends to swerve around the optimum.

6 A code snippet

- A snippet of your code that is essential to understanding your code.

```
def forward(x, t, params):
    W, b, V, c = params
    z = W@x + b
    h = sigmoid(z)
    o = V@h + c
    y = softmax(o)
    loss = -math.log((y.T@t)[0][0])
    params = W, b, V, c
    cache = z, h, o
    return params, loss, y, cache

def backward(x, t, params, y, cache):
    W, b, V, c = params
    z, h, o = cache
    gy = t * (-1/y)
    go = (-y@y.T + np.diagflat(y)) @ gy
    gc = go
    gV = go @ h.T
    gh = V.T @ go
    gz = gh * h * (1-h)
    gW = gz @ x.T
    gb = gz

    grads = gW, gb, gV, gc

    return grads

def SGD(x,t,params,alpha=1e-2):
    params, loss, y, cache = forward(x, t, params)
    grads = backward(x, t, params, y, cache)
    gW, gb, gV, gc = grads
    W, b, V, c = params
    W -= alpha * gW
    b -= alpha * gb
    V -= alpha * gV
    c -= alpha * gc
    params = W, b, V, c
    return params, loss, y
```

7 Appendix

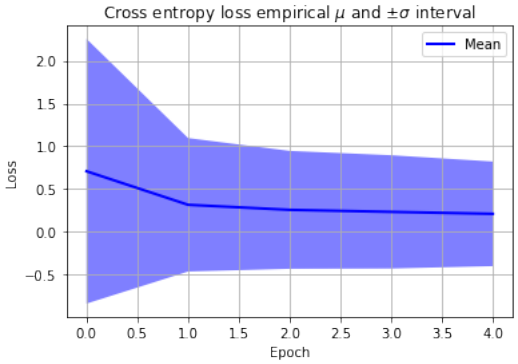

Figure 2: Loss confidence intervals

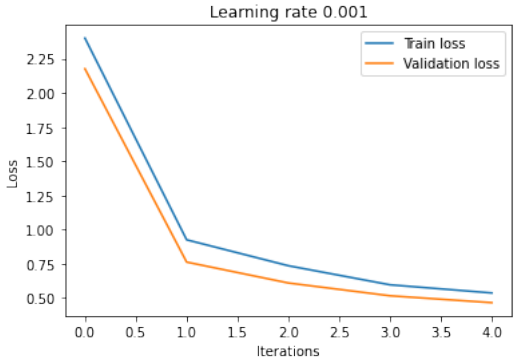

Figure 3: Learning rate 0.001

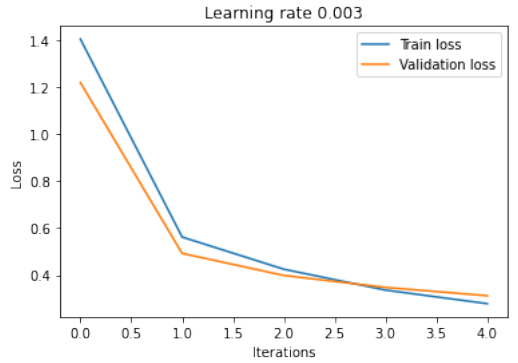

Figure 4: Learning rate 0.003

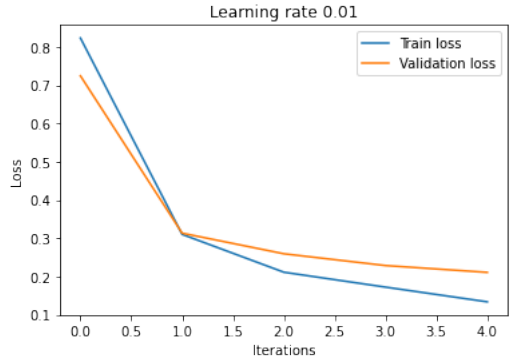

Figure 5: Learning rate 0.01

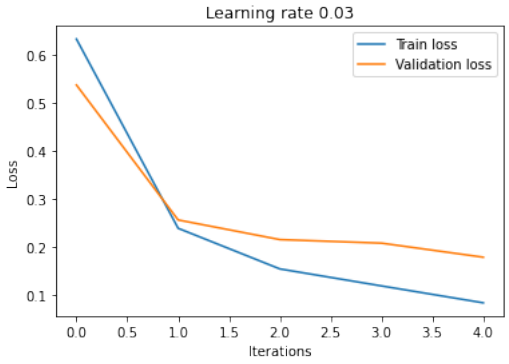

Figure 6: Learning rate 0.03

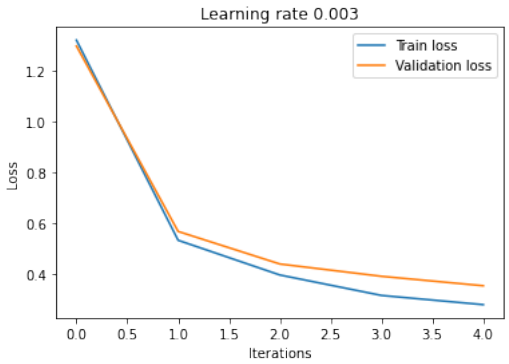

Figure 7: Final model learning rate 0.01