HLCV EXERCISE 2
DEEP NEURAL NETWORKS AND BACKPROPAGATION

Devikalyan Das
MSc. in Visual Computing
Universität des Saarlandes
Matriculation Number : 7007352
deda00002@stud.uni-saarland.de

Nobel Jacob Varghese
MSc. in Data Science and Artificial Intelligence
Universität des Saarlandes
Matriculation Number : 7002401
noja00001@stud.uni-saarland.de

Shashank Agarwal
MSc. in Embedded Systems
Universität des Saarlandes
Matriculation Number : 7009562
shag00001@stud.uni-saarland.de

1 Question 1

Part A

The scores generated for the toy inputs matches with the correct scores given by default.

```
Your scores:
[[0.3644621  0.22911264 0.40642526]
 [0.47590629 0.17217039 0.35192332]
 [0.43035767 0.26164229 0.30800004]
 [0.41583127 0.2983228  0.28584593]
 [0.36328815 0.32279939 0.31391246]]

correct scores:
[[0.3644621  0.22911264 0.40642526]
 [0.47590629 0.17217039 0.35192332]
 [0.43035767 0.26164229 0.30800004]
 [0.41583127 0.2983228  0.28584593]
 [0.36328815 0.32279939 0.31391246]]

Difference between your scores and correct scores:
2.9173411603133914e-08
```

Figure 1: Model scores comparison with correct scores

Part B

The loss function has been implemented and returns a value close to that of the correct loss value.

```
Difference between your loss and correct loss:
1.7963408538435033e-13
```

Figure 2: Model loss comparison with correct loss

Part C

All the operations have been implemented using numpy operations on matrices/vectors, without the use of any *for* loop.

2 Question 2

Part A

Please refer to the hand written solutions at the last.

Part B

Please refer to the hand written solutions at the last.

Part C

Please refer to the hand written solutions at the last.

Part D

Given below is a snapshot from the output of the code which shows that the relative error between weights and biases is very less i.e. it matches with the true values.

```
W2 max relative error: 3.440708e-09
b2 max relative error: 3.865070e-11
W1 max relative error: 3.561318e-09
b1 max relative error: 1.555470e-09
```

Figure 3: Relative error between gradients computed by our model and the numerical gradients

3 Question 3

Part A

The loss achieved on the training set is **0.017 < 0.02**.

```
Final training loss:  0.017149607938732093
```

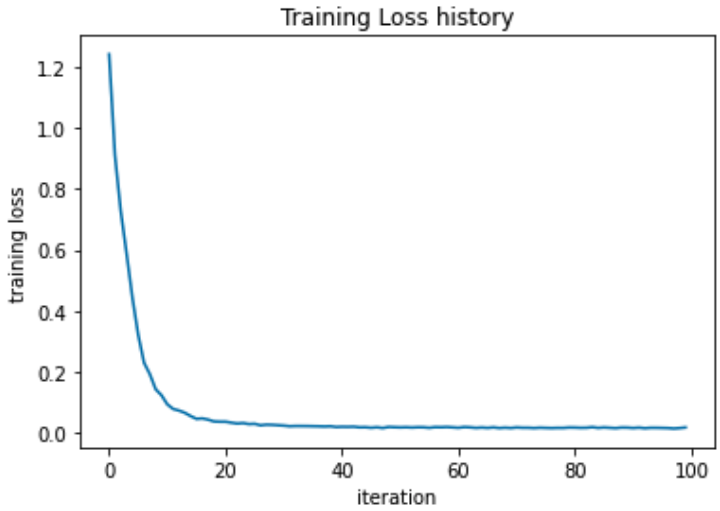


Figure 4: Loss on training set and the training curve

Part B

The obtained loss achieved is approx 28.7%.

```
iteration 0 / 1000: loss 2.302954
iteration 100 / 1000: loss 2.302550
iteration 200 / 1000: loss 2.297648
iteration 300 / 1000: loss 2.259602
iteration 400 / 1000: loss 2.204170
iteration 500 / 1000: loss 2.118565
iteration 600 / 1000: loss 2.051535
iteration 700 / 1000: loss 1.988466
iteration 800 / 1000: loss 2.006591
iteration 900 / 1000: loss 1.951473
Validation accuracy:  0.287
```

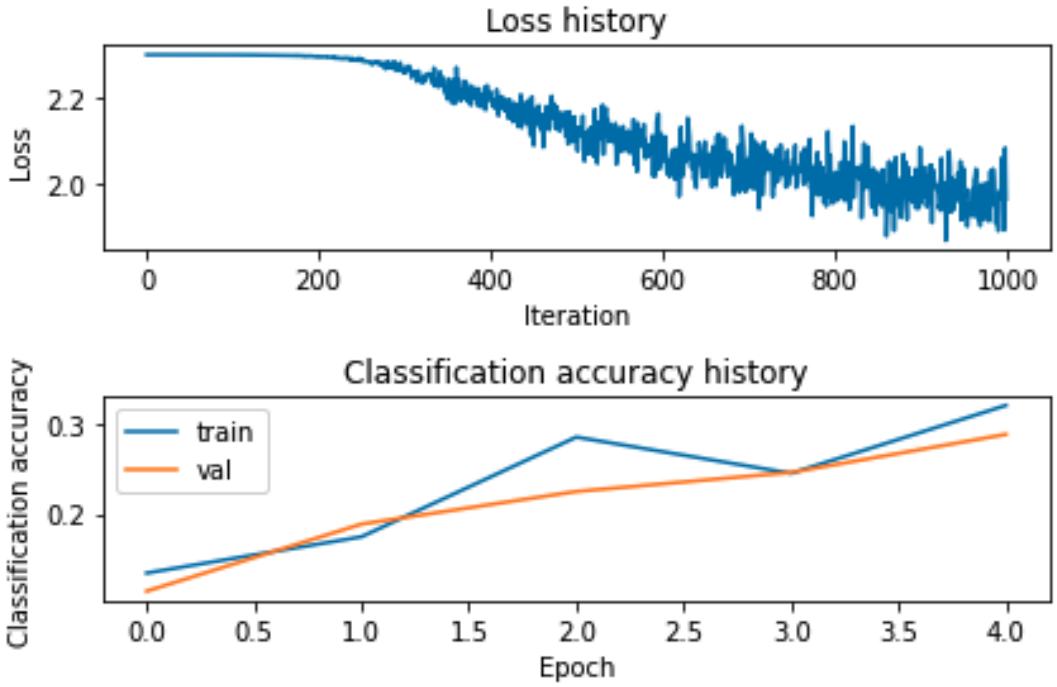


Figure 5: Validation accuracy with default parameters

We tried different experiments (as given in the table below) in order to achieve as high validation accuracy as possible.

Optimization techniques: We tried implementing Early Stopping, however it couldn't give any convincing results. Different patience levels were used - for low patience levels, the training stopped at a very early stage with a very low validation accuracy, and with a higher patience level there was not much of improvement in the validation accuracy too. The technique was tried with an initial learning rate of 0.0015, however we believe that if the learning rate would have been set to a bit lower value, it might have helped us increasing the accuracy, but still only marginally.

Dimensionality reduction: L2 regularization has been used in the implementation which helps in avoiding the to model to overfit and give a better generalisation error. Along with this, we also introduced PCA which helped us reduce the dimensionality from 3072 features to just approx. 640 features. However, we didnt see much improvement when PCA was added on top of L2 regularisation, instead we could see that the performance started to worsen up. This might be because L2 regularisation is already performing well in prioritizing the necessary features for classification and PCA doesn't adds up any good to the model, or else due to the fact that PCA doesn't keep the original structure of the images intact.

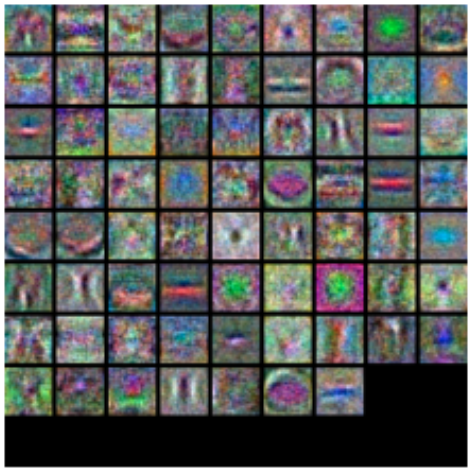
Data Augmentation: We used Data Augmentation to make the training data more diversified using random horizontal and vertical flips as well as scaling down the pixel values. Finally, we added this augmented dataset to the original train data and used this new larger dataset to train the model. This helped us in achieving a higher accuracy with only a small increase in training time.

Therefore, for the best model we have used a larger training dataset (with augmented data) with ReLU to add non linearity. L2 regularization has also been used on a 3 layer network with each hidden layer having 70 neurons. (Please refer Fig.6 for the hyperparameters used in the best model)
Best model's validation accuracy : 53%, test accuracy : 50.8%

Table 1: Top 4 models from a pool of 24 experiments performed for classification on CIFAR-10 dataset

Model	**Hidden Layer Size**	**No. of Iterations**	**Learning Rate**	**Reg. Constant**	**Validation Accuracy**
	All the models have a constant Batch Size: 1000 , Learning Rate Decay: 0.8				
Model 1	80	2000	0.0025	0.25	0.497
Model 2	70	2000	0.0025	0.25	0.487
Model 3	**80**	**4500**	**0.0015**	**0.29**	**0.53**
Model 4	80	4500	0.0025	0.25	0.495

```
Best Model
Hidden layer size:  80
Number of iterators:  4500
Batch size:  1000
Learning Rate:  0.0015
Regularization strength:  0.29
Learning Rate Decay:  0.8
New Validation accuracy:  0.53
Time consumed by best model:  185.3899884223938 seconds
```



```
Test accuracy:  0.508
```

Figure 6: Best Model Statistics

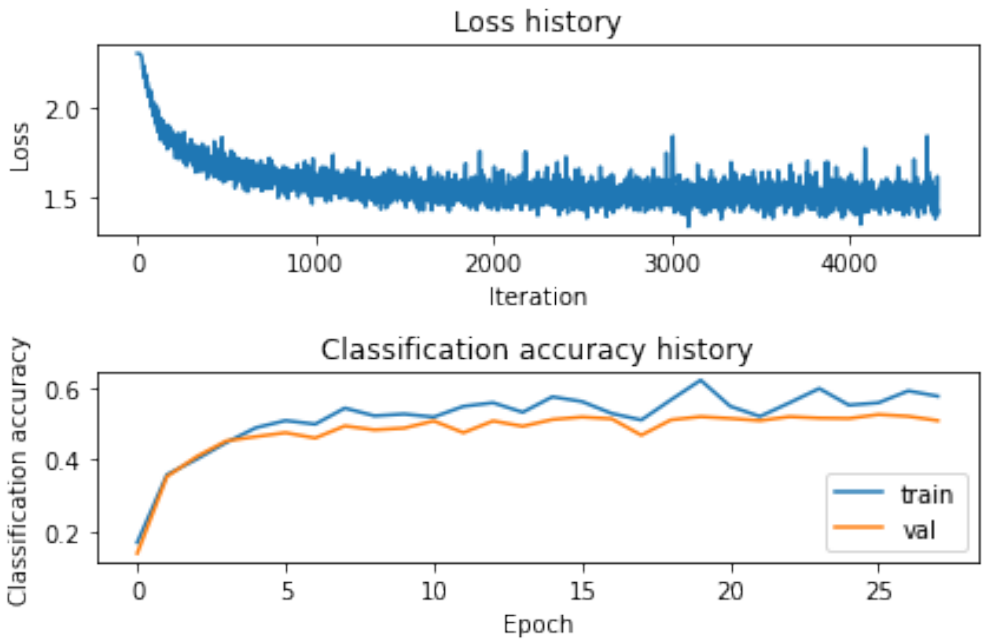


Figure 7: Best Model Statistics

4 Question 4

Part A

The code has been implemented using functions from **torch.nn**.

Part B

The code was implemented using the PyTorch built-in functions for creating a Multi Layer Perceptron model. The model consists of 2 layer with 50 neurons in the hidden layer and gave a validation accuracy of approximately **51.9%**.

Part C

We tried different experiments (as given in the table below) in order to achieve as high validation accuracy as possible. ReLu activation function was chosen to add non linearity in the models as it provides a better performance than Sigmoid and Hyperbolic tangent, when it comes to problems like vanishing gradient problems.

From the table below, it can be seen that **Model 5** would be the best choice to classify data in CIFAR-10 as it provides the highest validation accuracy. It uses 3 layers in total with 70 neurons each in the two hidden layers.
Best model's (Model 5) Validation Accuracy: 53.6%; Test Accuracy: 53.5% (Please refer Fig. 8 and 9)

Also, we can see that validation accuracy is higher than the training accuracy in general. This might be due to the fact that the validation set has less intrinsic variance than the training samples.

It can also be seen from the table that as we increased the number of layers more than 3, the training as well as validation accuracy rapidly fell to a single digit value. The performance in such scenarios points to the fact that increasing the number of layers in a network does not always increase the accuracy (not even the training accuracy). This is because for the given training data, we do not have enough features to fit the model with these many high number of layers. Hence, the learning is not happening in this case as the gradients approach to zero.

NOTE : All the experimental models have been saved in the directory as checkpoints. By default, the code runs the best model on the test dataset. In order to run any other specific model on the test dataset, please change the of the model in **ex2_pytorch.py - line number 301 and 306**. (For example, change the name to *"model_2.ckpt"* to run Model 2.)

By default, the model runs the best model for test dataset. To train the model, kindly set the flag **train = True** in line number 37 in ex2_pytorch.py

Table 2: Different models for classification on CIFAR-10 dataset using PyTorch built in functions (Accuracy has been measured after the last epoch by taking a mean of the accuracy on each sample)

Model	No. of layers	No. of neurons/hidden layer	Validation Accuracy	Train Accuracy
Model 1	2	50	51.9%	48.8%
Model 2	2	30	48.7%	46.3%
Model 3	2	70	52.3%	50.11%
Model 4	3	50,70 respectively	51.7%	46.6%
Model 5	**3**	**70,70 respectively**	**53.6%**	**49.3%**
Model 6	4	50,70,50 respectively	7.9%	9.8%
Model 7	5	50,70,70,50 respectively	7.8%	9.9%

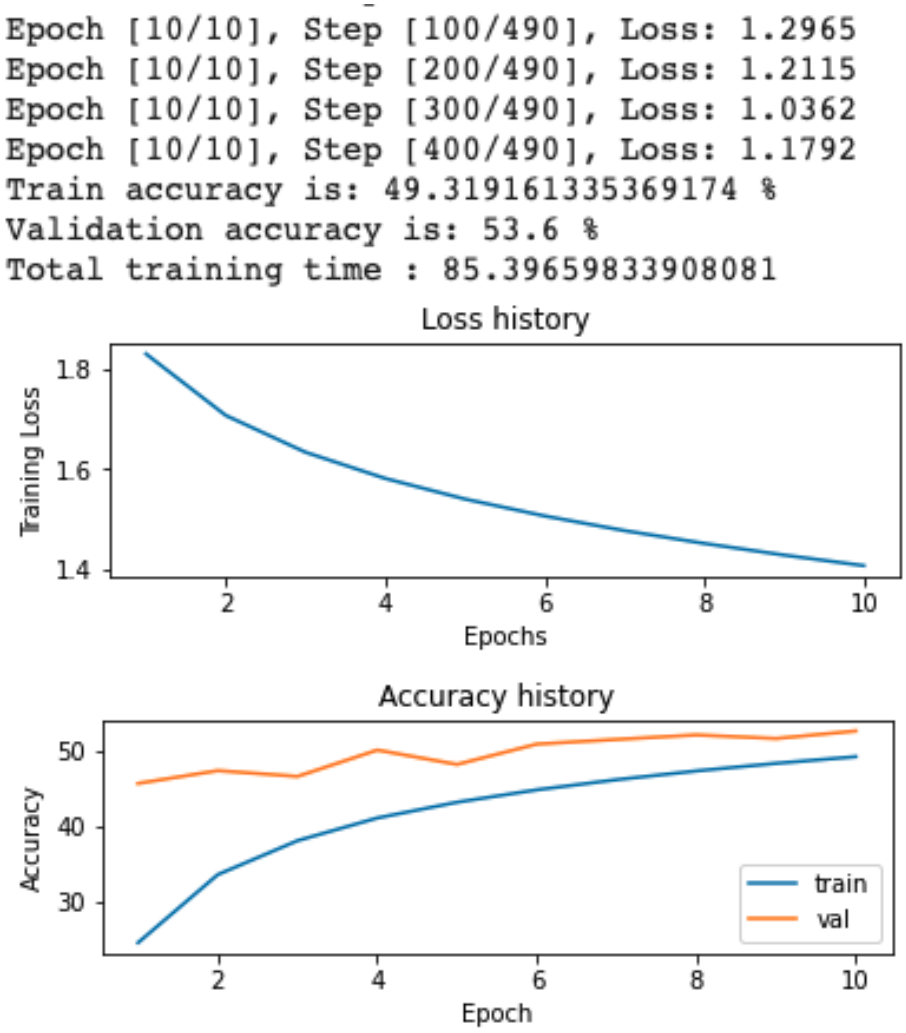


Figure 8: Best Model Statistics

```
Using device: cuda
MultiLayerPerceptron(
  (layers): Sequential(
    (0): Linear(in_features=3072, out_features=70, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=70, out_features=70, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=70, out_features=10, bias=True)
  )
)
Accuracy of the network on the 1000 test images: 53.5 %
```

Figure 9: Best Model Test Accuracy

Question 2 - Derivations(Continued)

Part a

let $\exp(z_{y_i}^3) = g_{y_i}$ & $\sum_j \exp(z_j^3) = h_j$

$a^3 = g_{y_i}/h_j = \psi(z^3)$ (softmax output)

so, $\frac{\partial J}{\partial z^3} = \frac{\partial J}{\partial a^3} \cdot \frac{\partial a^3}{\partial z^3}$

$\frac{\partial J}{\partial a^3} = \frac{-1}{N} \frac{\partial}{\partial a^3} \sum_{i=1}^{N} \log a^3 = -\frac{1}{N} \left(\frac{1}{a^3}\right)$

So, $\frac{\partial J}{\partial z^3} = \frac{-1}{N}\left(\frac{1}{a^3}\right)\frac{\partial a^3}{\partial z^3}$

The derivative $\frac{\partial a^3}{\partial z^3}$ will have two cases, when $z_{y_i}^3 = z^3 = z_j^3$ or $y_i = j$ ① because then only the derivative of $z_{y_i}^3$ by z^3 is possible. The derivative of z_j^3 is possible and by z^3 all the cases as z_j^3 has all the z^3 summed up. so only one will remain after derivative.

The other case is $z_{y_i}^3 \neq z^3$ or $y_i \neq j$ here the derivative of $z_{y_i}^3$ by z^3 will be zero.

if $y(i) \neq j$, then $\frac{\partial g_{y_i}}{\partial z^3} = 0$

$$\frac{\partial J}{\partial z^3} = \frac{-1}{N}\left(\frac{1}{a^3}\right)\left(\frac{g_{y_i}}{h_j^2}\exp(z_j^{(3)})\right)$$

$$= +\frac{1}{N} \cdot \psi(z^3_{y_j}) = \frac{1}{N}\psi(z^3_{y_i})$$

if $y_i = j$, the $\frac{\partial g_{y_i}}{\partial z^3} = g_{y_i}$ as only the part of z_{y_i} differentiated by z^3 one contained.

$$\frac{\partial J}{\partial z^3} = \frac{-1}{N}\left(\frac{1}{a^3}\right)\left(\frac{g_{y_i}}{h_j} - \frac{g_{y_i}\exp(z_j^3)}{h_j^2}\right)$$

$$= -\frac{1}{N}\left(1 - \frac{\exp(z_j^3)}{h_j}\right)$$

$$= \frac{1}{N}\left(\frac{\exp(z_j^3)}{\sum \exp(z_j^3)} - 1\right) = \frac{1}{N}(\psi(z^3_{y_j}) - 1)$$

$$= \frac{1}{N}(\psi(z^3) - 1)$$

so,

$$\boxed{\frac{\partial J}{\partial z^3} = \frac{1}{N}(\psi(z^3) - \Delta)}$$

where $\Delta = \begin{cases} 1 & \text{when } y_i = j \\ 0, & \text{otherwise} \end{cases}$

Part b & c

(b) The derivat" of L·f with w^2

$$\frac{\partial J}{\partial w^2}\left(\{x_i, y_i\}_{i=1}^{N}\right) = \frac{\partial J}{\partial z^3}$$

$$= \frac{\partial J}{\partial z^3} \cdot \frac{\partial z^3}{\partial z^3} \cdot \frac{\partial z^3}{\partial w^2} = \frac{\partial J}{\partial z^3} \cdot \frac{\partial z^3}{\partial w^2}$$

$$= \frac{1}{N}\left(\psi(z^3) - \Delta\right) \cdot a^{2'}$$

$$\therefore \frac{\partial z^3}{\partial w^2} = \frac{\partial}{\partial w^2}\left(a^2 w^2 + b^2\right) = a^{2'}$$

(c) considering the partial derivatives in case of regularization.

$$\frac{\partial \tilde{J}}{\partial w^2} = \underbrace{\frac{1}{N}\left(\psi(z^3) - \Delta\right) \cdot a^{2'}}_{\text{from previous case.}} + \frac{\lambda}{\partial w^2}\partial\left(\|w^{(1)}\|_2^2 + \|w^{(2)}\|_2^2\right)$$

$$= \frac{1}{N}\left(\psi(z^3) - \Delta\right) \cdot a^{2'} + 2\lambda w^{(2)}$$

(c) The partial derivatives of loss w.r.t all the model parameters ($w^{(1)}$, $w^{(2)}$, $b^{(1)}$, $b^{(2)}$)

① $\frac{\partial J}{\partial b^{(2)}} = \frac{\partial J}{\partial z} \frac{\partial J}{\partial z^3} \cdot \frac{\partial z^3}{\partial b^2}$

$= \frac{1}{N} (\psi(z^{(3)}) - \Delta) \cdot 1$

② $\frac{\partial J}{\partial w^{(1)}} = \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(1)}}$

$= \frac{1}{N} (\psi(z^{(3)}) - \Delta) \cdot w^{(2)} \cdot \phi(z^{(2)}) \cdot$

② $\frac{\partial J}{\partial w^{(1)}} = \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(1)}}$

$+ \frac{\partial}{\partial w^{(1)}} \cdot \lambda (\|w^{(1)}\|_2^2 + \|w^{(2)}\|_2^2)$

$\Rightarrow \frac{\partial J}{\partial w^{(1)}} = \frac{1}{N} (\psi(z^{(3)}) - \Delta) \cdot w^{(2)} \cdot \phi'(z^{(2)}) \cdot a^{(1)}$

$+ 2\lambda w^{(1)}$

③ $\frac{\partial J}{\partial b^{(1)}} = \frac{\partial J}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(1)}}$

$= \frac{1}{N} (\psi(z^{(3)}) - \Delta) \cdot w^{(2)} \cdot \phi'(z^{(2)})$

NB: $\phi'(z^{(2)})$ → derivative of relu.