EX 1 – DL Basics

Nimrod Curtis, 311230924, nimrodcurtis@mail.tau.ac.il
Alon Mizrahi, 312284706, alonmizrahi2@mail.tau.ac.il

Spring 2023

1 Theory

1.1 Q1

Assuming MLP composed of an input of size 10, followed by one hidden layer with output of size 50, and finally one output layer with 3 output neurons. Given batch size $= m$.

a. The shape of the input X is $(m, 10)$

b. The shape of the weights vector W_h is $[10, 50]$ and the shape of the bias vector b_h is $[50]$

c. The shape of the weights vector W_o is $[50, 3]$ and the shape of the bias vector b_o is $[3]$

d. The shape of the output Y is $[m, 3]$

e. $Y = ReLu(W_o^T \cdot (ReLu(W_h^T \cdot X + b_h)) + b_o)$

1.2 Q2

The number of parameters in convolutional networks is determined by the size of the kernel and the number of channels. (The number of parameters does not depend on the size of the image). We will divide this into 3 parameter layers, and calculate the total number of parameters:
First layer has Kernel 3×3 and from 3 channels to 100 Therefore the calculation:
$3 \times 3 \times 3 \times 100 + 100 = 2800$ parameters.
Second layer has same Kernel and from 100 channels to 200 Therefore the calculation:
$3 \times 3 \times 100 \times 200 + 200 = 180200$ parameters.
Third layer has same Kernel and from 200 channels to 400 Therefore the calculation:
$3 \times 3 \times 200 \times 400 + 400 = 720400$ parameters.
A total of 903400 parameters.

1.3 Q3

Given a one-dimensional batch normalization (BN) layer and a mini-batch of size m, the calculations for the following gradients below, can be obtained using the chain rule and the notations provided in the exercise text:

a. $\frac{\partial f}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \hat{x_i}$

b. $\frac{\partial f}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i}$

c. $\frac{\partial f}{\partial \hat{x_i}} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \hat{x_i}} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \gamma$

d. $\frac{\partial f}{\partial \sigma^2} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \hat{x_i}} \cdot \frac{\partial \hat{x_i}}{\partial \sigma^2} = \sum_{i=1}^{m} -0.5 \cdot (\sigma^2 + \epsilon)^{-\frac{3}{2}} \cdot \gamma \cdot \frac{\partial f}{\partial y_i} \cdot (x_i - \mu)$

e. $\frac{\partial f}{\partial \mu} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \hat{x_i}} \cdot \frac{\partial \hat{x_i}}{\partial \mu} = \gamma \cdot \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial \hat{x_i}}{\partial \mu}$
$where : \frac{\partial \hat{x_i}}{\partial \mu} = \frac{\sigma^2 + \epsilon + \frac{x_i - \mu}{m} \cdot \sum_{i=1}^{m} (x_i - m)}{(\sigma^2 + \epsilon)^{\frac{3}{2}}}$

f. $\frac{\partial f}{\partial x_i} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \hat{x_i}} \cdot \frac{\partial \hat{x_i}}{\partial x_i} = \gamma \cdot \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \frac{\partial \hat{x_i}}{\partial x_i}$
$where : \frac{\partial \hat{x_i}}{\partial x_i} = \frac{1}{\sqrt{\sigma^2 + \epsilon}} + \frac{\partial \hat{x_i}}{\partial \mu} \cdot \frac{\partial \mu}{\partial x_i} + \frac{\partial \hat{x_i}}{\partial \sigma^2} \cdot \frac{\partial \sigma^2}{\partial x_i} = \frac{(m-1)(\sigma^2 + \epsilon) - (x_i - \mu)^2}{m(\sigma^2 + \epsilon)^{\frac{3}{2}}}$

2 Practical

In this section, we conducted multiple training sessions of the *Lenet*5 neural network, incorporating different forms of regularization in each iteration to facilitate comparative analysis of the results.
LeNet-5 is a convolutional neural network architecture developed by Yann LeCun in 1998 for handwritten digit recognition. It consists of seven layers, including convolutional, pooling, and fully connected layers, and was one of the earliest successful examples of deep learning.
You can see the net architecure below:

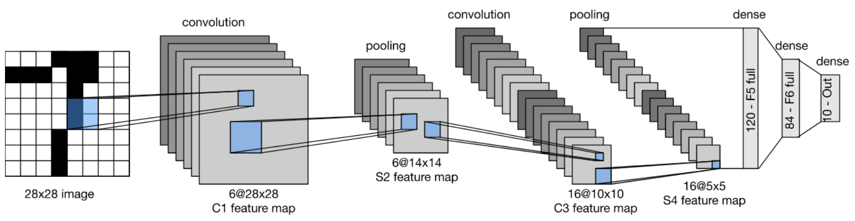
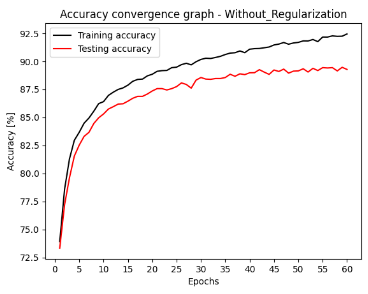


Figure 1: LeNet5 Architecture

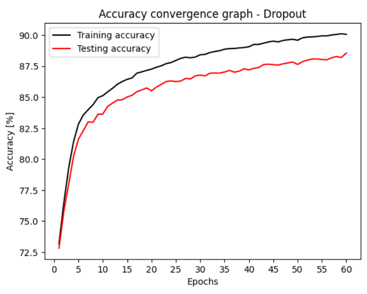
Throughout all the experiments, we used an Adam optimizer with a learning rate of 0.0001 and Cross-entropy as the loss function.
The regularization types that were evaluated include:

1. Batch normalization
2. Dropout
3. Weights decay

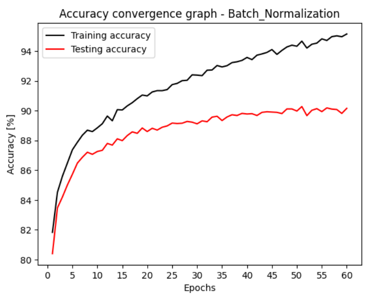
Furthermore, an additional training session was conducted without any form of regularization, to facilitate comparison of the results with the original network. The training outcomes are visualized through convergence graphs depicting the model accuracies, and a final accuracy table.



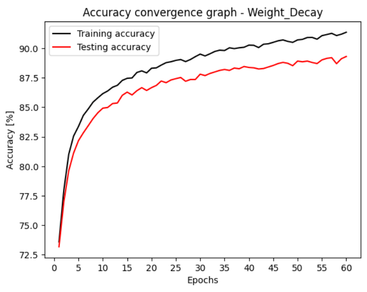
(a)



(b)



(c)



(d)

Figure 2: (a) Without regularization, (b) With dropout (c) with batch norm (d) with weight decay.

Type	Train Accuracy	Test Accuracy
Without Regularization	92.44	89.27
Dropout (0.25)	90.07	88.55
Batch Normalization	95.16	**90.16**
Weight Decay	91.35	89.30

Table 1: Training-Testing metrics using different regularization methods

Conclusions:

- The convergence graphs exhibit comparable patterns, with slight indications of over-fitting performance - the training curve displays a persistent increase with increasing experience.
- After about 50 epochs, the test accuracy curves converged, while training beyond this point may result in over-fitting as we experienced. To address this issue and save computational resources, we can consider implementing an early stopping technique that terminates training once the performance no longer improves significantly.
- By implementing the Batch Normalization method, we can achieve higher accuracy levels in a relatively shorter training time of around 40 epochs. However, this approach may also result in higher over-fitting performance.