CSE616 Neural Networks and Their Applications
Assignment 2

Shahira Hany Hussein Mohamed Amin (2101341)

May 2022

1 Problem 1

The shape of the weight matrix of this layer (without the bias) is $100 \times (500 \times 500 \times 3)$. The shape of the bias is 100×1.

2 Problem 2

The number of weights of the convolutional layer equals $5 \times 5 \times 3 \times 10$, and the number of biases of the convolutional layer equals 10. The number of parameters that the convolutional layer has equals the number of weights plus the number of biases which is equal to $5 \times 5 \times 3 \times 10 + 10$.

3 Problem 3

To get the image on the left, the values of the filter used were

$$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$$

This filter finds the edges in the vertical direction because the zeros column is in the vertical direction. As the center column of the mask consists of all zeros, then it does not include the original values of the image, but it calculates the difference between the right and left pixel values around that edge.

To get the image on the right, the values of the filter used were

$$\begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}$$

This filter finds the edges in the horizontal direction because the zeros row is in the horizontal direction. As the center row of the mask consists of all zeros, then it does not include the original values of the image, but it calculates the difference between the above and below pixel values around that edge.

4 Problem 4

Fig. 1 illustrates what happens in the adam optimizer as the number of steps gets larger. In Fig. 1, the dark spot in the center is the optimum point to be reached as the number of steps gets large. The circles are the contours of the loss function. The arrows represent the steps taken to reach the optimum point with adam optimizer (red path) and mini-batch SGD (black path).

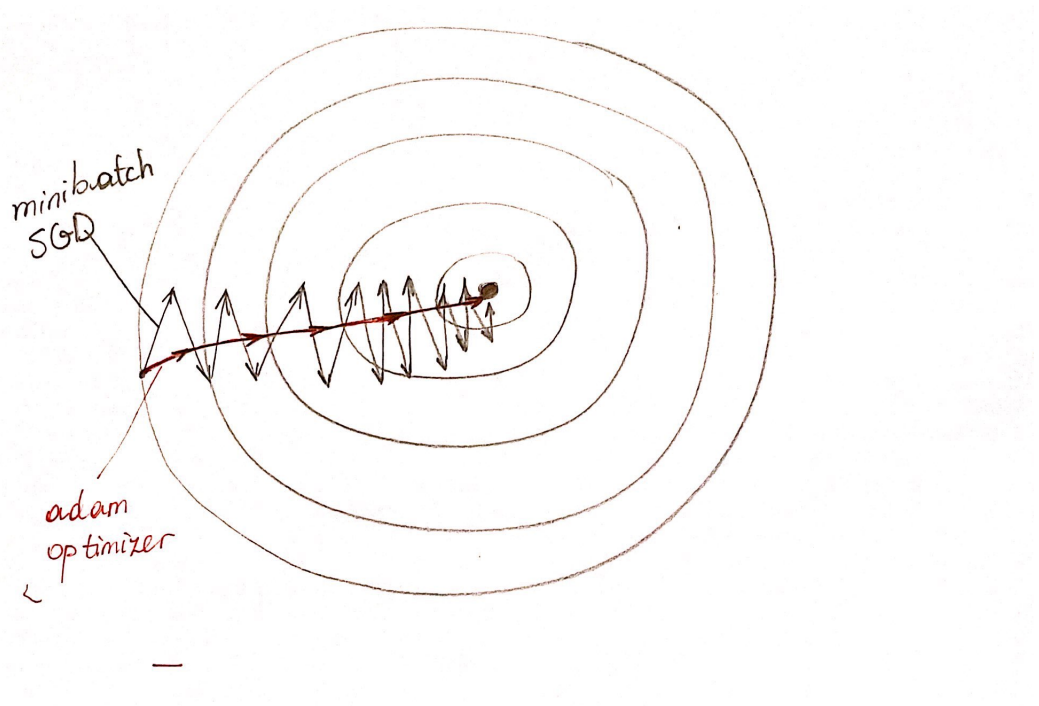

Figure 1: Path to reach optimum point with adam optimizer and mini-batch SGD

It is clear from Fig. 1 that, starting at the same initial point, the path of learning in mini-batch SGD is zigzag (not straight). However, the adam optimizer increases the horizontal movement and reduces the vertical movement, thus making the zigzag path straighter, and thus reducing the training time.

Adam combines both momentum optimizer and RMSprop optimizer. Table 1 shows the weight update and bias update at each step for momentum, RMSprop, and adam.

	Momentum	RMSprop	Adam
Weight update	$W = W - \alpha V_{dW}$, $V_{dW} = \beta_1 V_{dW,prev.} + (1-\beta_1)\frac{\partial Cost}{\partial W}$	$W = W - \alpha \frac{\frac{\partial Cost}{\partial W}}{\sqrt{S_{dW}+\epsilon}}$, $S_{dW} = \beta_2 S_{dW,prev.} + (1-\beta_2)\left(\frac{\partial Cost}{\partial W}\right)^2$	$W = W - \alpha \frac{V_{dW}}{\sqrt{S_{dW}+\epsilon}}$
Bias update	$B = B - \alpha V_{dB}$, where $V_{dB} = \beta_1 V_{dB,prev.} + (1-\beta_1)\frac{\partial Cost}{\partial B}$	$B = B - \alpha \frac{\frac{\partial Cost}{\partial B}}{\sqrt{S_{dB}+\epsilon}}$, $S_{dB} = \beta_2 S_{dB,prev.} + (1-\beta_2)\left(\frac{\partial Cost}{\partial B}\right)^2$	$B = B - \alpha \frac{V_{dB}}{\sqrt{S_{dB}+\epsilon}}$

Table 1: Comparison Between Momentum, RMSprop, and Adam Weight Update and Bias Update

In Table 1: α is the learning rate, β_1 and β_2 are the weighting factors in the exponential moving average in momentum and RMSprop, respectively. Initially, $V_{dW,prev.} = 0, V_{dB,prev.} = 0, S_{dW,prev.} = 0, S_{dB,prev.} = 0, \beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 10^{-3}$.

5 Problem 5

Given a batch size of m as input to the batch normalization layer, $\mathbf{z} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$, the output equation of this layer is

$$y_i = BN_{\alpha,\beta}(\mathbf{z}^{(i)}) = \alpha\hat{\mathbf{z}}^{(i)} + \beta, \quad i = 1, \ldots, m \tag{1}$$

where α and β are parameters to be learned in this layer, and

$$\hat{\mathbf{z}}_i = \frac{\mathbf{z}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2}$$

with

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{z}^{(i)} \quad \text{and} \quad \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{z}^{(i)} - \mu_{\mathcal{B}})^2 \tag{3}$$

Reasons for using batch normalization are:

- Improves gradient flow through the network
- Reduces strong dependence on initialization
- Acts as a form of regularization

6 Problem 6

Fig. 2 shows two-layered fully convolutional neural network with a 3×3 kernel in each layer. The shaded area marks the receptive field of one pixel in the second layer (convolutional layer) while the dotted area marks the receptive field of one pixel in the third layer (pooling layer). From Fig. 2, the size of the receptive field for a single unit in the pooling layer is $5 \times 5 \times 3 = 75$.

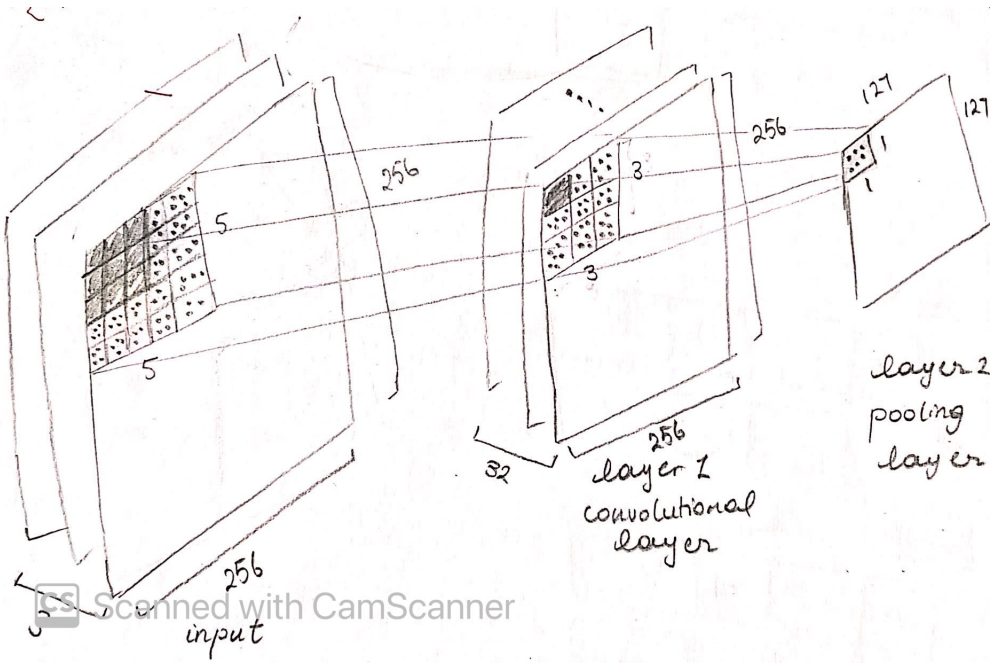

Figure 2: Receptive Field

7 Problem 7

The spatial dimension of the output data block is $\lfloor \frac{128+2\times3-7}{2} \rfloor + 1 = 64$. Therefore, the dimension of the output data block is $128 \times 64 \times 64$

8 Problem 8

Inverted dropout is a variant of the original dropout technique. Just like traditional dropout, inverted dropout randomly keeps some weights and sets others to zero. This is known as the "keep probability" p. The one difference is that, during the training of a neural network, inverted dropout scales the activations by the inverse of the keep probability. The advantage of this is that it prevents network's activations from getting too large and does not require any changes to the network during evaluation.

9 Problem 9

Fig. 3 shows two images with a flower, but the flower is in a different location in both images. Fully connected neural networks are sensitive to the location of the flowers. That is, shifting the flower results in an entirely different input that the network will not recognize. Therefore, fully connected neural networks do not work well for image classification because we need a network that is shift invariant while a fully connected neural network is sensitive to shift.

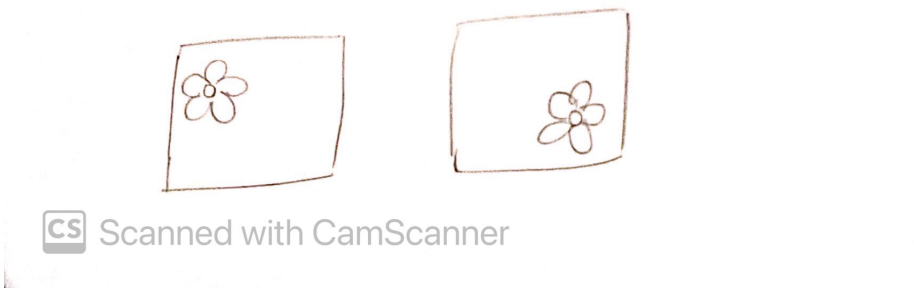

Figure 3: Two Images With a Shifted Flower

10 Problem 10

Fig. 4 shows the steps at which the convolution is calculated. In step 1, the result of the convolution is $4 \times -2 = -8$. In step 2, the result is $4 \times 1 + 1 \times -2 = 2$. In step 3, the result is $1 \times 1 + -1 \times -2 = 3$. In step 4, the result is $-1 \times 1 + 3 \times -2 = -7$. In step 5, the result is $3 \times 1 = 3$. The final result of the convolution is a vector of length 5 which is equal to $\begin{bmatrix} -8 & 2 & 3 & -7 & 3 \end{bmatrix}$

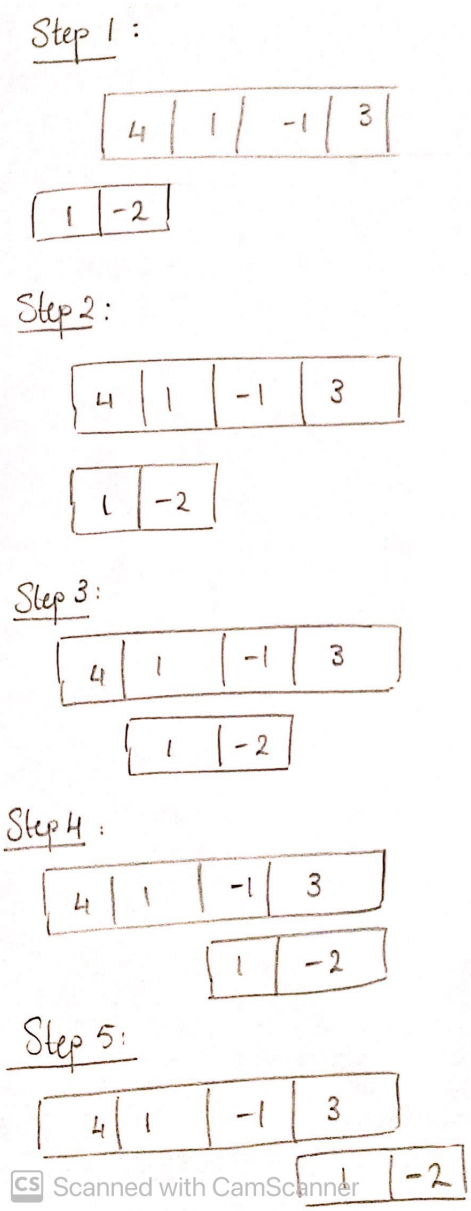

Figure 4: Convolution Steps

11 Problem 11

The drop in error happens at the point when the learning rate is decreased. Learning rate scheduling is used to automatically reduce the learning rate when the optimization reaches a plateau.

12 Problem 12

Convolutional layers are more commonly used that fully connected (FC) layers for image processing for the following reasons:

- In FCs, one input as a whole entity passes through all the activation units while convolutional layers use a floating window that takes into account a specific number of pixels at a time. Therefore, convolutional layers have better computation time and memory usage than FCs.
- Since convolutions are not densely connected, then not all inputs affect all output nodes. This gives convolutional layers more flexibility in learning.
- Convolutional layers summarize the image features and yield concise feature maps for the consecutive convolution layers on the pipeline. This provides dimension reduction and reduce computational complexity which is not possible with FCs.
- FCs enforce static image sizes whereas convolution layers allow to work on arbitrarily sized images.
- FCs have a large number of weights (parameters). Thus, are highly subject to overfitting while a single convolution operation reduces the number of parameters significantly which makes it less prone to overfitting. Moreover, smaller number of weights helps a lot with high dimension inputs such as image data.

13 Problem 13

Drop out layers work as follows:

At training time, randomly set some neurons to zero in the forward pass. For example, consider the network shown in Fig. 5 and assume that the probability that the node value is retained, $p = 0.5$.

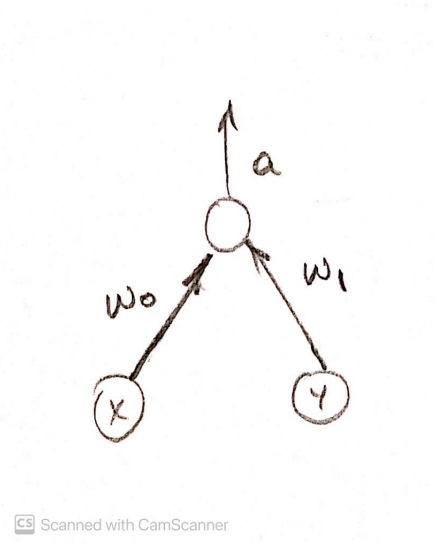

Figure 5: A Simple Network

Then, the average activation during training is

$$\begin{aligned}\mathbb{E}\left[a\right] = \frac{1}{4}\Big(& w_0 \times 0 + w_1 \times 0 \\ &+ w_0 \times 0 + w_1 \times y \\ &+ w_0 \times x + w_1 \times 0 \\ &+ w_0 \times x + w_1 \times y\Big) \\ &= \frac{1}{4}(2w_0 \times x + 2w_1 \times y) \\ &= \frac{1}{2}(w_0 \times x + w_1 \times y) \end{aligned} \tag{4}$$

where the binary masks $(0,0), (0,1), (1,0)$, and $(1,1)$ are applied one at a time on the input (x, y), and then the average activation is calculated. We can see that the activation is scaled by the factor $p = 0.5$.

At test time, all neurons are active, so we must scale the activations by p so that for each neuron, the output at test time equals the expected output at training time. Therefore, during testing, $a = \frac{1}{2}(w_0 \times x + w_1 \times y)$.

14 Problem 14

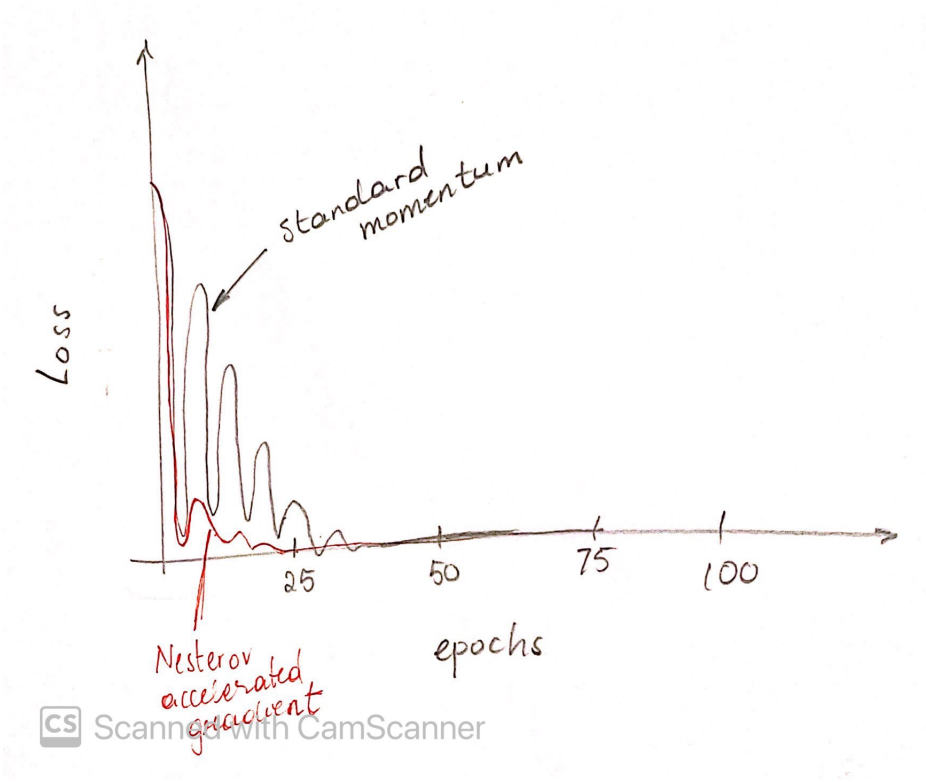

Figure 6: A Comparison Between Standard Momentum and Nesterov Accelerated Gradient

Nesterov accelerated gradient helps limit the overshoot that might occur with standard momentum when it is about to reach the optimum point. In standard momentum, the weight update is

$$W = W - \alpha V_{dW}, \text{ where } V_{dW} = \beta V_{dW,prev.} + (1-\beta)\frac{\partial Cost}{\partial W} \tag{5}$$

When approaching the optimum point, the weight update in (5) will come down with a lot of momentum causing overshoot.

On the other hand, with Nesterov accelerated gradient, the weight update is

$$W_{new} = W_{old} - v_t, \text{ where } v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\underbrace{\theta - \gamma v_{t-1}}_{\text{look ahead term}}) \tag{6}$$

As we approach the optimum point, the look ahead term in (6) will slow down (decrease) the velocity compared to the previous time step, thereby reducing the overshoot that happens near the optimum point. This is clear in Fig. 6, which shows the that the loss function overshoots near the optimum point when using standard momentum (black plot) while with Nesterov accelerated gradient (red plot), the overshoot does not occur.

15 Problem 15

Adagrad uses a different learning rate for each parameter θ_i at every time step t. Let $g_{t,i}$ be the partial derivative of the objective function with respect to the parameter θ_i at time step t.

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \tag{7}$$

Adagrad update rule is

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{8}$$

where $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element i, is the sum of squares of the gradient with respect to θ_i up to time step t, while ϵ is a smoothing term that avoids division by zero ($\epsilon \approx 10^{-8}$). Therefore, adagrad modifies the actual learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i.