Name: Manasa Challa
UB IT No.: 50356441

CSE 676 : Assignment #2

Due Date : 11:59 PM April 25, 2021

Q1. 1. Padding size = 1

$$x = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 0 \\ 0 & 5 & 6 & 7 & 8 & 0 \\ 0 & 9 & 10 & 11 & 12 & 0 \\ 0 & 13 & 14 & 15 & 16 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$ → with padding

as stride = 2, convolutional output is,

$$O_{ij} = (x * w)_{ij} \triangleq \sum_k \sum_l x_{k+i, l+j} w_{kl}$$

assuming matrix indices start from 0.

$$O_{0,0} = x_{0,0} \cdot w_{0,0} + x_{0,1} \cdot w_{0,1} + x_{1,0} \cdot w_{1,0} + x_{1,1} \cdot w_{1,1}$$

$$= 0.4 + 0.2 + 0.5 + 1.1$$

$$= 1$$

moving filter by 2 positions to the right,

$$O_{0,1} = x_{0,2} \cdot w_{0,0} + x_{0,3} \cdot w_{0,1} + x_{1,2} \cdot w_{1,0} + x_{1,3} \cdot w_{1,1}$$

$$= 0.4 + 0.2 + 2.5 + 3.1$$

$$= 15.$$

III^ly

$$O_{0,2} = x_{0,4} \cdot w_{0,0} + x_{0,5} \cdot w_{0,1} + x_{1,4} \cdot w_{1,0} + x_{1,5} \cdot w_{1,1}$$

$$= 0.4 + 0.2 + 4.5 + 0.1 = 20.$$

$$O_{1,0} = x_{2,0} \cdot w_{0,0} + x_{2,1} \cdot w_{0,1} + x_{3,0} \cdot w_{1,0} + x_{3,1} \cdot w_{1,1}$$
$$= 0.4 + 85.2 + 9.1 + 0.5$$
$$= 19$$

$$O_{1,1} = x_{2,2} \cdot w_{0,0} + x_{2,3} \cdot w_{0,1} + x_{3,2} \cdot w_{1,0} + x_{3,3} \cdot w_{1,1}$$
$$= 6.4 + 7.2 + 10.5 + 11.1$$
$$= 99$$

$$O_{1,2} = x_{2,4} \cdot w_{0,0} + x_{2,5} \cdot w_{0,1} + x_{3,4} \cdot w_{1,0} + x_{3,5} \cdot w_{1,1}$$
$$= 8.4 + 0.2 + 12.5 + 0.1$$
$$= 92$$

$$O_{2,0} = x_{4,0} \cdot w_{0,0} + x_{4,1} \cdot w_{0,1} + x_{5,0} \cdot w_{1,0} + x_{5,1} \cdot w_{1,1}$$
$$= 0.4 + 13.2 + 0.5 + 0.1$$
$$= 26.$$

$$O_{2,1} = x_{4,2} \cdot w_{0,0} + x_{4,3} \cdot w_{0,1} + x_{5,2} \cdot w_{1,0} + x_{5,3} \cdot w_{1,1}$$
$$= 14.4 + 15.2 + 0.5 + 0.1 = 86.$$
$$= 86.$$

$$O_{2,2} = x_{4,4} \cdot w_{0,0} + x_{4,5} \cdot w_{0,1} + x_{5,4} \cdot w_{1,0} + x_{5,5} \cdot w_{1,1}$$
$$= 16.4 + 0.2 + 0.2 + 0.1 = 64.$$

$\therefore$ output $= \begin{bmatrix} 1 & 15 & 20 \\ 19 & 99 & 92 \\ 26 & 86 & 64 \end{bmatrix}$

The relation $\Rightarrow O = \frac{I + 2(P) - F}{S} + 1$

As stride value increases the value skips decreases by same ratio. Both row and column at both ends are added due to padding and filter size decreases the number by its size.

2. (i) Given :- input = 32 x 32

Layer 1 → 6 filters

Each filter → dimension of 5 x 5

~~Number of parameters & filter~~

No. of parameters per filter = (5 x 5) + 1 (bias) = 26.

Total no. of parameters = 26 x No. of filters.

= 26 x 6

= 156

Dimension of o/p :

∴ o/p dimension of each filter $= O = \frac{I + 2(P) - F}{S} + 1$

$O = \frac{32 + 2(0) - 5}{1} + 1$

$O = 28$

- Each filter will output a 28 x 28 matrix
- 6 filters → 28 x 28 x 6 output dimension.

(2) So the LeNet Architecture has 2 convolutions layer followed by sub sampling layer which is the max pooling layers and after flattening the output of second convolutional block it is fed to 2 fully connected layers.

We can take the following steps to reduce the no. of parameters. Due to the accuracy trade off and also poorer performance due to the model complexity they can lead to poorer performance:

→ Reducing kernel size to 4 x 4.

~~[redacted]~~

→ change value of stride 1 to 2

→ Reducing no. of kernal : 6 to 5

According to above changes,

- No. of parameters per filter = (4 x 4) + 1 = 17

∴ Total no. of parameters = 17 x No. of filters
= 17 x 5
= 85.

- O/P dimension of each filter = $O = \frac{I + 2(P) - F}{S} + 1$

$$O = \frac{32 + 2(0) - 4}{2} + 1$$

$$O = 15$$

- 15 x 15 matrix is the o/p of each filter.
- 6 filters therefore o/p dimension is 15 x 15 x 5
- The direct effect of decrease in o/p dimensionality is the number of parameters of further layers.

- The number of parameters has gone from 156 to 85 in just the first layer by 2 small changes that were made.

Q2. 1. Given: Gradients calculated for timestamp t+1.
Need to calculate backpropagate based on given values and need to calculate the gradients for time t

First let us calculate $\frac{\partial L}{\partial h_t}$ in terms of gradients from timestamp t+1. And then using the value of $\frac{\partial L}{\partial h_t}$ to update all the gradient value at time t.

In the LSTM, architecture can be seen from the figure the Backpropagation through time takes place by 2 nodes i.e c_{t+1} and h_t.

So, now solving the equations at timestamp t+1 to calculate the gradient value for h_t which would be required for the calculation of all other gradient at time t.

$$\Rightarrow z_t = \begin{bmatrix} \hat{i}_t \\ \hat{f}_t \\ \hat{o}_t \\ \hat{g}_t \end{bmatrix} = \begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$z_t = W \times I_t$$
$$i_t = \sigma(\hat{i}_t)$$
$$f_t = \sigma(\hat{f}_t)$$
$$O_t = \sigma(\hat{O}_t)$$
$$g_t = \sigma(\hat{g}_t)$$

So we have $\frac{\partial L}{\partial i_{t+1}}, \frac{\partial L}{\partial f_{t+1}}, \frac{\partial L}{\partial o_{t+1}}, \frac{\partial L}{\partial g_{t+1}}, \frac{\partial L}{\partial c_{t+1}}$ and $\frac{\partial L}{\partial h_{t+1}}$ i.e

$\delta i_{t+1}, \delta f_{t+1}, \delta o_{t+1}, \delta g_{t+1}, \delta g_{t+1}, \delta c_{t+1}$ and δh_{t+1}

To calculate δh_t, we are applying partial derivatives in equation 2 for time $t+1$

$$\delta \hat{i}_{t+1} = \delta i_{t+1} \odot i_t \odot (1 - i_{t+1})$$
$$\delta \hat{f}_{t+1} = \delta f_{t+1} \odot f_t \odot (1 - f_{t+1})$$
$$\delta \hat{O}_{t+1} = \delta o_{t+1} \odot O_t \odot (1 - O_{t+1})$$
$$\delta \hat{g}_{t+1} = \delta g_{t+1} \odot (1 - \tanh^2(\hat{g}_{t+1}))$$

So now,

$$\delta_{z_{t+1}} = \begin{bmatrix} \hat{i}_{t+1} \\ \hat{f}_{t+1} \\ \hat{O}_{t+1} \\ \hat{g}_{t+1} \end{bmatrix}$$

we get

$$\delta I_{t+1} = W^T \times \delta_{z_{t+1}}$$

$$I_{t+1} = \begin{pmatrix} h_t \\ x_{t+1} \end{pmatrix}$$

$$\delta_{I_{t+1}} = \begin{pmatrix} \delta h_t \\ \delta x_{t+1} \end{pmatrix}$$

we get δh_t from above matrix which is being back propagated by further time steps.

So Secondly, considering the state of the cell at time t. The δh_t will contain 2 parts i.e the backpropagated error calculated earlier and the error difference calculated for that particular time stamp.

$$\Rightarrow \quad \delta h_t = \Delta_t + \delta h_t \text{ (from time } t+1)$$

Now solving the equations for time t
The front pass equations are given as -

$$c_t = f_t \odot c_{t-1} + i_t \odot + g_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Since we have δh_t, we can use it to calculate δi_t, δf_t, δo_t, δg_t and δc_t.

Starting with eq. 9. calculating δc_t. Using partial derivatives.

$$\delta o_t = \frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial o_t}$$
$$\delta o_t = \delta h_t \cdot \tanh(c_t)$$
$$\delta o_t = \delta h_t \odot \tanh(c_t)$$

Now, calculating δc_t using eq. 9. The backpropagation path to c_t comes from h_t and c_{t+1}.

So, $$\delta c_t = \frac{\partial L}{\partial c_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial c_t} + \frac{\partial L}{\partial c_{t+1}} \cdot \frac{\partial c_{t+1}}{\partial c_t}$$

$$\delta c_t = \delta h_t \odot o_t \odot (1 - \tanh^2(c_t)) + \delta c_{t+1} \cdot f_t$$

Now, calculating gradients i.e δi_t, δf_t and δg_t as they depend on δc_t.

$$\delta i_t = \frac{\partial L}{\partial i_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial i_t}$$

$$\delta i_t = \delta c_t \cdot g_t$$
$$\delta i_t = \delta h_t \odot g_t$$

Similarly,

$$\delta f_t = \frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial f_t}$$

$$\delta f_t = \delta c_t \cdot c_{t-1}$$

$$\delta f_t = \delta h_t \odot c_{t-1}$$

similarly,

$$\delta g_t = \frac{\partial L}{\partial g_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial g_t}$$

$$\partial g_t = \delta c_t \cdot i_t$$

$$\delta g_t = \delta h_t \odot i_t$$

So, to summarize the solution. We obtain the value of δh_t. The value is calculated as the sum of gradient calculated at that times tamp and the back propagated value from δI_{t+1} which is a stack of δh_t & δx_t

$$\delta o_t = \delta h_t \odot \tanh(c_t)$$

$$\delta c_t = \delta h_t \odot o_t \odot (1 - \tanh^2(c_t)) + \delta c_{t+1} \odot f_t$$

$$\delta i_t = \delta h_t \odot g_t$$

$$\delta f_t = \delta h_t \odot c_{t-1}$$

$$\delta g_t = \delta h_t \odot i_t$$

Q3. (1) The (a), (b) and (c) can be learned using the standard VAE.

(2) The standard VAE is based on the assumption that the latent space variables are a Gaussian Distribution. This assumption is made to simplify the calculation of Variational Inference problem.

So, the fourth distribution is not a Gaussian distribution so it can't be learnt.

$$KL\left(\frac{q(x)}{p(z|x)}\right) = -\sum q(x)\log\left(\frac{p(z|x)}{q(z)}\right)$$

$$= -\sum q(x)\log\left(\frac{p(x,z)}{q(z)*p(x)}\right)$$

$$= -\sum q(x)\left(\log\left(\frac{p(x,z)}{q(z)}\right)+\log\left(\frac{1}{p(x)}\right)\right)$$

$$= -\sum q(x)\left(\log\left(\frac{p(x,z)}{q(z)}\right)-\log(p(x))\right)$$

$$= -\sum q(x)\left(\log\left(\frac{p(x,z)}{q(z)}\right)\right) + \log(p(x))\sum q(x)$$

$$= -\sum q(x)\left(\log\left(\frac{p(x,z)}{q(z)}\right)\right) + \log(p(x))$$

$$KL\left(\frac{q(x)}{p(z|x)}\right) = -\sum q(x)\left(\log\left(\frac{p(x,z)}{q(z)}\right)\right) + \log(p(x))$$

$$KL\left(\frac{q(x)}{p(z|x)}\right) = -\text{Variational Lowerbound} + \log(p(x))$$

In the above equation the $\log(p(x))$ is the constant term. Here we are trying to minimize the KL divergence. Minimizing KL Divergence is same as maximizing Lower Bound.

$$\text{Lower Bound} = \sum q(x)\left(\log\left(\frac{p(x,z)}{q(z)}\right)\right)$$

$$= \sum q(x)\left(\log\left(\frac{p(x|z)\cdot p(z)}{q(z)}\right)\right)$$

$$= \sum q(x)\left(\log(p(x|z)) + \log\left[\frac{p(z)}{q(z)}\right]\right)$$

$$= \sum q(x)\log(p(x|z)) - D_{KL}[q(z)||p(z)]$$

$$= E_{q(z)}\log(p(x|z)) - D_{KL}[q(z)||p(z)]$$

Now, while optimizing we try to maximize the lower Bound. The first term corresponds to the reconstruction loss ie how close is the reconstructed output with respect to the input. Since here we assume that $q(z)$ distribution is Gaussian, the second term corresponding to the KL Divergence tries to Normalize the latent space vector as close to the Gaussian Distribution as possibly by reducing the divergence.

Q4.1. Generative Adversarial Nets (GANs)

- Method to generate samples (Based on game theory)
- they generate new samples from the same data distribution

training data $\sim p_{data}(x)$; generated samples $\sim p_{model}(x)$

The goal is to learn $p_{model}(x)$ such that it is close to $p_{data}(x)$

GAN training framework: It corresponds to a minmax two-player game.

- For arbitary G & D, a unique solution exists
- If G and D are MLP's can be trained using backdrop.

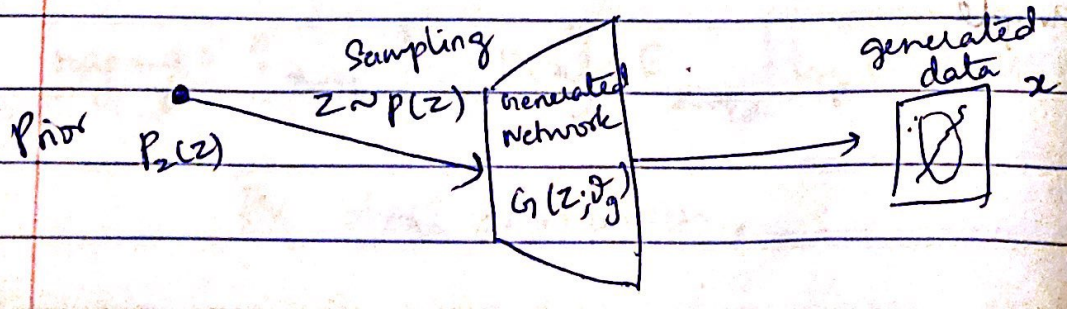
- In the space of arbitary function G & D, a unique sol. exist

Resulting G is used to generate images.

Generator Network: To learn the generator distribution p_g over data x, we define a prior on input noise variables $p_z(z)$.

Represent a mapping to data spaces as $G(z; \theta_g)$.

where g → differentiable generator function.



- Generator Network produces samples
$$x = G(z; \theta_g)$$
- Discriminator network attempts to distinguish samples drawn from training data and samples drawn from generator and also emits a propability value. $D(x; \theta^d)$
- We also have a second multilayer perceptron $D(x; \theta^d)$ that outputs a single scalar.

Training D & G: Train D to maximize the prob. of assigning the correct lable to both training examples and samples from G

Simultaneously train G to minimize $\log(1 - D(G(z)))$

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

So,

D is accurate over real data by maximizing

$$E_{x \sim p_{data}(x)}[\log D(x)]$$

G is trained to increase D's prob. for a false example.

Combined together D and G are playing a minmax game in which we optimize:

- To find best value for D consider

$$L(G,D) = \int_x \left(P_{data}(x) \log D(x) + P_g(x) \log(1 - D(x)) \right) dx$$

- Setting the derivative to zero is

$$D^*(x) = x^* = \frac{A}{A+B} = \frac{P_r(x)}{P_r(x) + P_g(x)} \in [0,1]$$

- Once G is trained to optimal, P_g is very close to P_{data}
- When $P_g = P_{data}$, $D^*(x)$ become $1/2$

D is trained to discriminate samples from data, converging to $D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)}$

After an update of G, gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data.

★ GAN Objective: Value function of Minmax played by G, D.

$$\min_G \max_D \; \mathbb{E}_{x \sim P_x} \left[\log D(x) \right] + \mathbb{E}_{z \sim P_z} \left[\log (1 - D(G(z))) \right]$$

P_x: data distribution,

$P_{G(z)}$: model distribution, z modeled as Gaussian.

- We must implement the game using an iterative numerical approach

Optimizing D to completion in inner loop of training is computationally prohibitive and results in overfitting in finite data sets.

Instead we alternate between k steeps of optimizing D and one step of optimizing G.

Alternating gradient updates :-

1: fix G and perform a gradient step for maximization

$$\max_D E_{x \sim P_x}[\log D(x)] + E_{z \sim P_z}[\log(1 - D(G(z)))]$$

2: fix D and perform a gradient step for minimizati

In theory:

$$\min_G E_{z \sim P_z}[\log(1 - D(G(z)))]$$

In practice

$$\max_G E_{z \sim P_z}[\log D(G(z))]$$

2. Mode collapse happens when few modes generated Partial mode collapse, a hard problem to solve in GAN. A complete collapse is not common but a partial collapse happens often

The Reason for Mode Collapse:

Objective of G: create images to fool D.

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$$

Case when G is trained without updates to D.

- Generated images converge to x^* that fool D the most -- most realistic from the D perspective.
  - In this extreme, x^* will be independent of z.

$$x^* = \text{argmax}_x D(x)$$

Mode collapses to a single point

- Gradient associated with z approaches zero $\frac{\partial J}{\partial z} \approx 0$

When we restart the training in D, the most effective way to detect generated images is to detect this single mode.

- Since G ~~desensitizes~~ desensitizes the impact of z already, the gradient from D will likely push the single point around for the next most vulnerable mode.

- This is not hard to find. It produces such an imbalance of modes in training that it deteriorates its capability to detect others.

- Now, both network are overfitted to exploit the shortterm opponent weakness. This turns into a cat and mouse chase and the model will not converge.