Let’s fill this in together! 😀 (Please feel free to dunk on my answers!)

1ai)

[[1, 1, 1, 1],

[-1, 2, 2, -1],

[0, 1, 1, 0],

[1, 0, 0, 1]]

Flip the filter kernel horizontally and vertically before doing the multiplications

Why does it need to be flipped? ^ - You need to flip the kernel if you want to actually perform convolution, otherwise you are doing cross-validation. The below answer is correct

This should be the correct one:

[[1, 0, 0, 1],

[-1, 1, 1, -1],

[0, 2, 2, 0],

[1, 1, 1, 1]]

1aii)

[[1, 1, 1, 1],

[0, 2, 2, 0],

[0, 1, 1, 0],

[1, 0, 0, 1]]

[[1, 0, 0, 1],

[0, 1, 1, 0],

[0, 2, 2, 0],

[1, 1, 1, 1]]

1aiii)

Sigmoid and Tanh

1aiv)

Max isn’t a multiplication op so ignore it

6 \* 6 \* 9 = 324

^looks correct, with zero padding of 1 around the image, there are 6 \* 6 “stops” for the convolution, each stop requires 9 multiplications (and 8 additions). No multiplications from ReLU as it is a max,

so 6 \* 6 \* 9 = 324

~~I would think it’s (6 x 6) x ((3 x 3) + 1) = 360~~

~~The +1 is due to ReLU~~

~~Is this not 4\*4\*9 =144 (No, because padding is used)~~

1av)

We add padding of 1.

1avi)

(M – m + 1) x (N – n + 1)

1avii)

* Add approximate shift invariance
* Down sampling by extracting most important features

1bi)

1. Let f\_out be the number of out features for the layer.

Number of parameters = 64 \* 64 \* 1 \* f\_out + f\_out <- to account for biases

1. Let f\_out be the number of out features(channels) for the conv layer

Number of parameters = (5 \* 5 \* 1 + 1) \* f\_out <- +1 for biases

1bii)

To approximate a Lipschitz continuous function f: R^d -> R with accuracy epsilon one will need of order O(epsilon^(-d)) examples

epsilon = 1 – 0.1 = 0.9

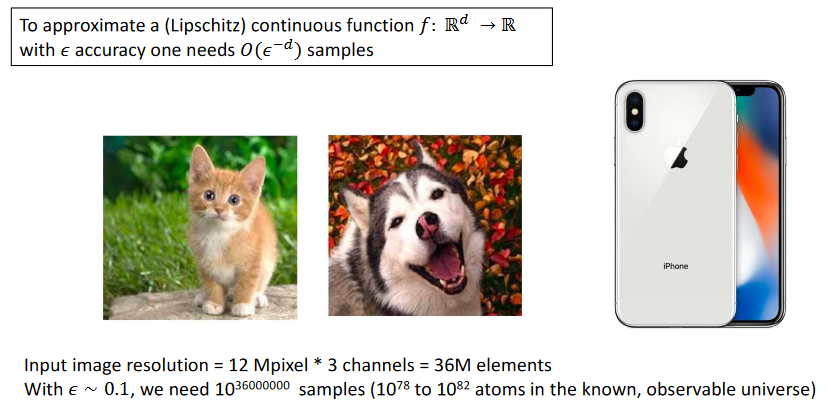
D = 1024

0.9^(-1024) ~ 10^46

0.93^(-1024) ~ 10^32

I think the answer is 10^42 since it’s the only one around this range

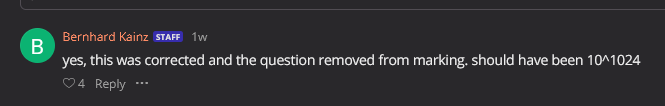
What is d here, I don’t think it’s 1024 since that’s the size of the latent representation? Coz the answer is ~ 0.9^(-d) similar to this example:



I guess this question is just here to show us how insanely large the number of samples required to adequately cover the input space.

But, as epsilon approaches 1, doesn’t the number of samples go down? This seems very counterintuitive if epislon is to be interpreted as the desired accuracy, because at close to perfect accuracy, you would only need a handful of samples.

\*\*Edstem indicates that the answer should be .1^-1024



1biii)

No, this is because convolutional networks are not translational/shift invariant. One can approx shift invariance using a max pooling layer (which isn’t used in this network), but we would not expect the network to perfectly classify every example (unless you had a massive dataset with all possible permutations which the network could be overfitted to).

Alternative: z

No, because the pattern wraps around, and hence the pattern is not shift-invariant to our convolution.

Yes, because the pattern wraps around, and we can compute convolution by wrapping around also (padding the end of the sequence with the values from the start of the sequence), thus making the pattern shift-invariant and allowing detection.

Alternative:

No, convolution is a linear operation, even if the number of linear layers and the linear activation function are followed, it is still equal to a single-layer MLP and therefore cannot solve the linear indivisibility problem.o

Alternative:

Yes, but only with circular padding. Assuming that warped around means that the pattern is cyclic.   
For example, a 1D convolution of size 2 could be used to detect whether we have a pair of blocks. After that step separating between A and B becomes the matter of checking whether we have 2 or more pairs of blocks (linearly separable). However, without circular padding such a classifier would misclassify a block when there is a pair that wraps around. With a circular padding we can still detect it.

Discussion:

I think a lot of the answers above talk about shift invariance, when we don’t actually care about this because this is a classification problem. We only care about shift equivariance, which means that the response for the input being shifted is the same as the response being shifted after the fact, which is held by convolutional layers which do not use max-pooling.

Shift invariance is what’s more useful for classification, so I’m not sure why equivariance is the problem here (equivariance is something we mention for tasks like segmentation, as according to the prof)

ChatGPT-4 says:

No, as convolutions don’t have shift invariance. Shift invariance makes sure that when we shift the objects in the image, we would still classify them identically to what we classified them before shifting.

1biv) ??? Batch gradient update directio aligns more with the true underlying update direction.

Less variance in our gradient estimators so we e n xpect to converge in fewer iterations

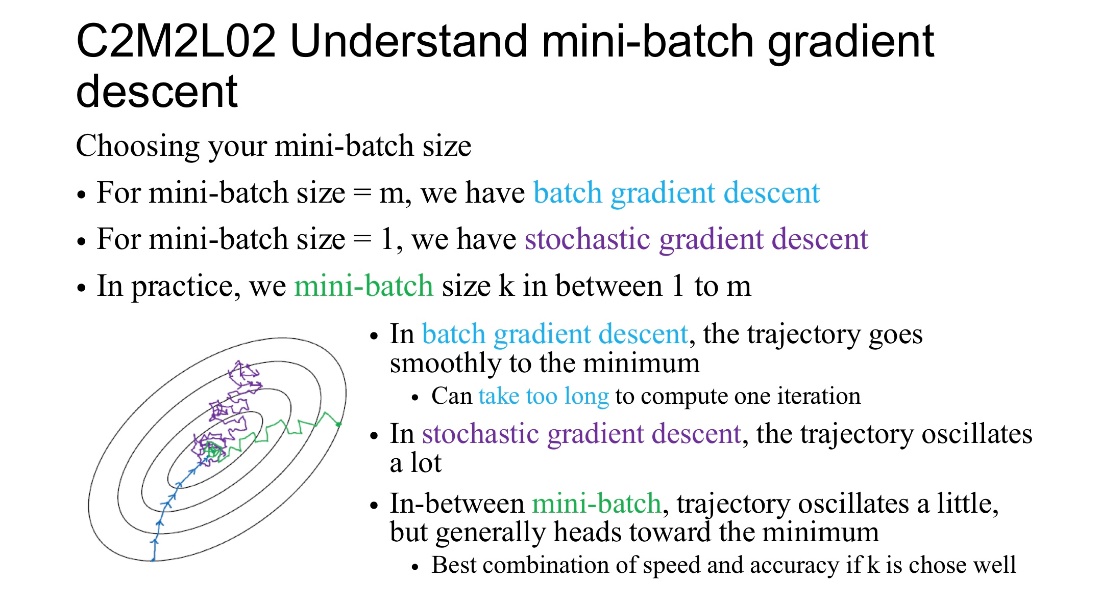
For small batch sizes, we are far below the number of samples required to get a representative sample of the training set. The smaller the batch size, the more iterations it takes to accrue a sample size that is likely to be a representative sample.

1bv) ??? More conservative in terms of update due to averaging effect.

* Forgetting problem when batch size is large?

The batch size is large enough to represent the full dataset, so increasing the batch size won’t have much effect?`

For larger batch sizes, we may already have enough samples in each batch to be a representative sample of the training set. Increasing the number of iterations doesn’t give us a much more representative sample.



1bvi) Yes. By using dropout and BatchNorm can be more robust…

Using either is good but using both isn’t any better? Problem is that both act as capacity control and so achieve the similar thing:  
“so you're basically getting a random shift and a random scale from any batch. This is one of the reasons why if you use batch normalization you really don't need drop out in the same network. they kind of do similar things in terms of capacity control.” - <https://www.doc.ic.ac.uk/~bkainz/teaching/DL/notes/BatchNorm.pdf>, pages 6 and 7

They both try to regularize the data via noise injection so using both techniques has no benefit

This question is tricky. However, after reading more opinions about that I can conclude that indeed batchnorm adds as well regularisation by noise injection which is also performed by Dropout. However, this is not the main purpose of BatchNorm. It’s mainly used to reduce the internal covariate shift – the difference between the distributions of the outputs of different layers. This difference slows down the network. So we use mainly batchnorm to mitigate this and speed up the learning process. So, yes, it could be beneficial in some cases to use them together. As batchnorm will speed up the learning process, and Dropout will regularise. It may indeed happen in some scenarios that the regularisation provided by batchnorm has the same effect as the one provided by dropout, but this is not true for all cases. Thus, in my opinion the correct right answer should be ‘depends’ rather then ‘yes’ or ‘no’.

BatchNorm makes covariate shift worse – it works because it acts as a regularizer by injecting noise, which is mostly its only benefit discussed in class.

ci) Estimates the residual function H(x) = f(x) - x. Acts like ‘Taylor expansion’ for parameterisation of the neural network.

cii) 1. Mitigate vanishing gradient since there is a ‘highway’ (I.e. x) for the gradient to backpropagate instead of going through the convolutions.

2. Adding new resnet blocks would not worsen the function approximation performance.

3. Improve learning efficiency sometimes, the learning efficiency is at least as good as the plain network

ciii) If the batch size is 1 then x\_{t+1} = beta from batchnorm equation (mu = 0), where beta is a learnable parameter. Hence the ResNet block is approximating f(x)+beta.

2ai)

p(x1, x2, z1, z2) = p(x1|z1, z2)p(x2|z2)p(z1)p(z2)

2aii)

Chat GPT:

The objective of a Variational Autoencoder (VAE) can be stated as minimizing the Kullback-Leibler (KL) divergence between the approximate posterior distribution qφ(z1, z2|x1, x2) and the prior distribution p(z1, z2), while maximizing the lik”elihood of the observed data p(x1, x2|z1, z2). This can be expressed as the following loss function:

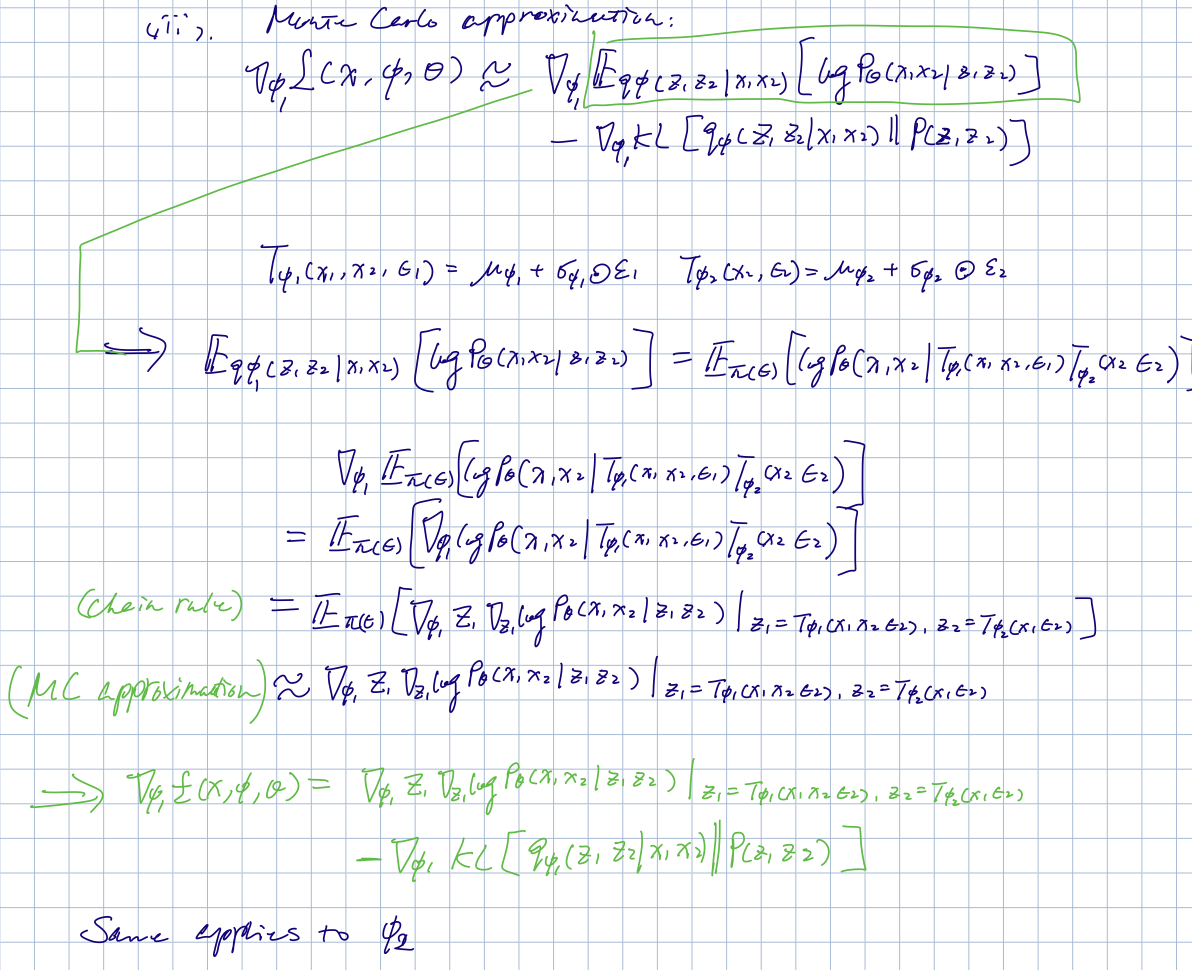
L(φ,θ;x1,x2) = -KL(qφ(z1,z2|x1,x2) || p(z1,z2)) + E\_{qφ(z1,z2|x1,x2)}[log pθ(x1,x2|z1,z2)]

where θ and φ are the parameters of the decoder and encoder, respectively, and E\_{qφ(z1,z2|x1,x2)} represents the expected value taken with respect to the approximate posterior distribution.

2aiii)

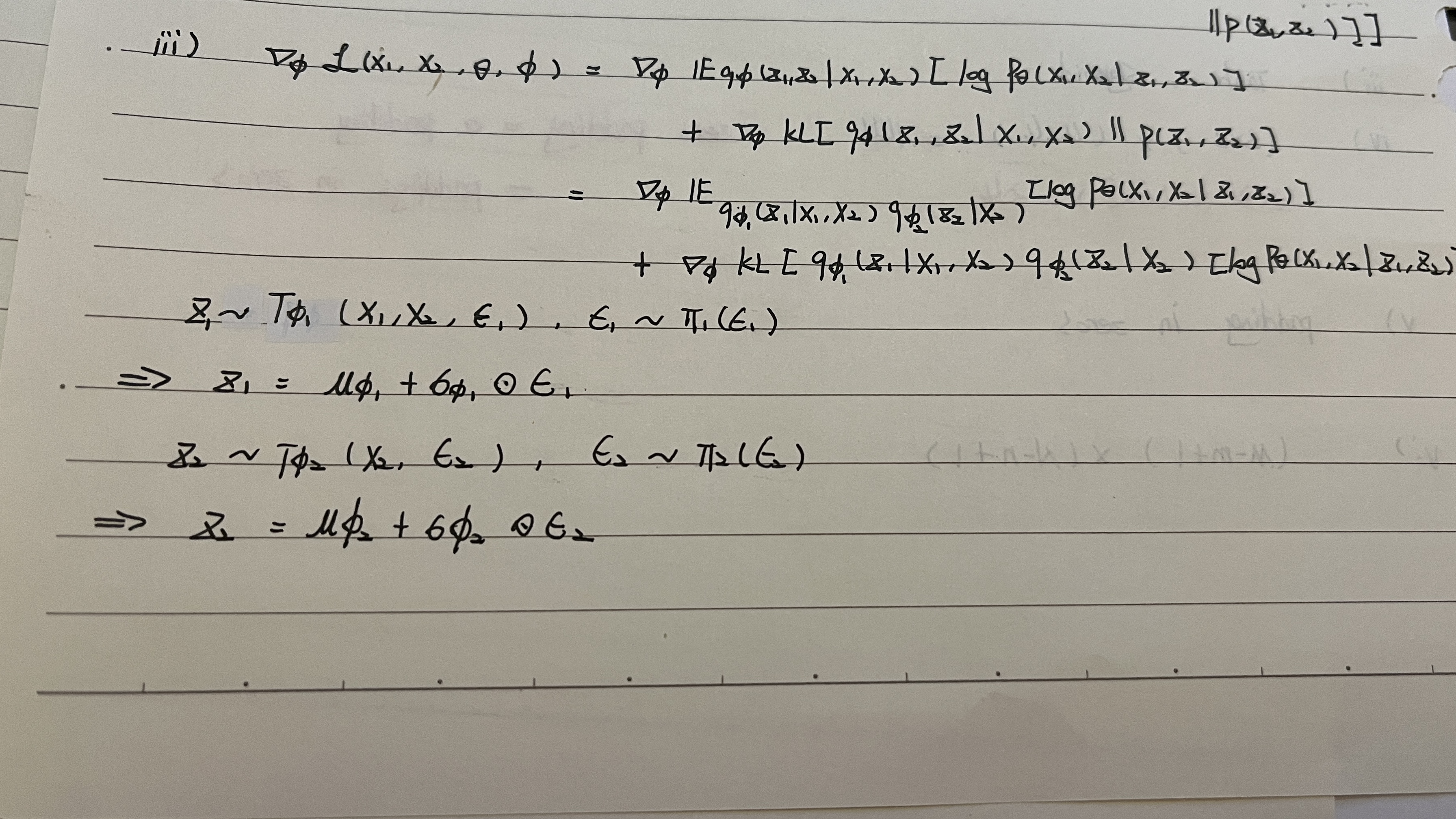
Not entirely sure about this. And not sure if we are expected to expand KL derivatives as well... help...

I think we should note that we used LOTUS rule somewhere

/CJ\*

Can the likelihood here not be separated depending on the factorization of q\_z? I don’t think we should be finding it as a joint

Alternative:



2aiv) None of the above

2bi) The answer is in the example question

2.bii)

This is because at equilibrium the discriminator cannot tell the difference between real and fake samples and therefore outputs a prob of ½. We then simply rearrange the answer given in 2bi) :)

2.biii) 2,4

- see the 2021 example questions and answers PDF (Q8) for explanation

2.ci)

2.cii) last singular eigenvalue = 2.09

2.09 > 1 therefore will explode

2. ciii) 1, 2, 3 are true (I think 5 is also true, from her notes)

She said “Alternatively, one can construct the recurrent weight matrix Wh to be orthogonal or unitary

Matrix"

civ) During training, to handle arbitrary input(source)/output(target) length, we can add padding to the source and target sentences to make all source/target training examples have the same length (max length of the source/target sentences), so that decoder and encoder have a fixed number of layers. During testing, we modify the decoder so that it takes the predicted word at t-1 as decoder input at t to predict the output word at t. It stops predicting when the predicted output indicates it is the end of the sentence.

2.di)

2.dii) 4/H

The query projection matrix dimensions scales down by 1/H

2.diii) O((H-h+1)^2(W-w+1)^2wh)

2.div) 3 and 4 (unsure on 2)