Review Questions 4
ID2223
Group: Nicolas Oulianitski Essipova, Peter Lakatos, Marios Chatiras

1. As we go deeper in a multi-layered network, the gradient for the respective layer is calculated as the product of many gradients from previous layers. The gradients start decreasing if start multiplying everything with a factor smaller than 1 and leads to the 'vanishing' of the gradient, meaning that the training will essentially stop, as the weights will be harder to train and the domino effect goes through the whole network. The opposite of this problem is the exploding gradient, when the multiplying factor is too big. To avoid the vanishing gradient, it is recommended to initialize weights in a way that the vanishing potential is minimized, or another common practice is the use of LSTMs.

2. Learn Gate: Combines the short-term memory (STM) and most recent input (E) by concatenating and pushing them through a neural network with a tanh activation function, resulting in output N, then ignores a part of it by multiplying it with an ignore vector factor i that is derived from pushing through the STM and E through another neural network with a sigmoid activation function. Relevant equations:

$$N_t = \tanh(W_n[STM_{t-1}, E_t] + b_n)$$

$$i_t = \sigma(W_i[STM_{t-1}, E_t] + b_i)$$

Forget Gate: Takes in the long-term memory (LTM) and decides which part to forget and which to remember. It does so by multiplying it with the forget f which is derived from pushing through the concatenation of STM and E through another neural network with a sigmoid activation function, similar to the case in the learn gate. Relevant equation:

$$f_t = \sigma(W_f[STM_{t-1}, E_t] + b_f)$$

Remember Gate: Simply combines the output of Forget gate and Learn gate. That's it 😊
Relevant equation:

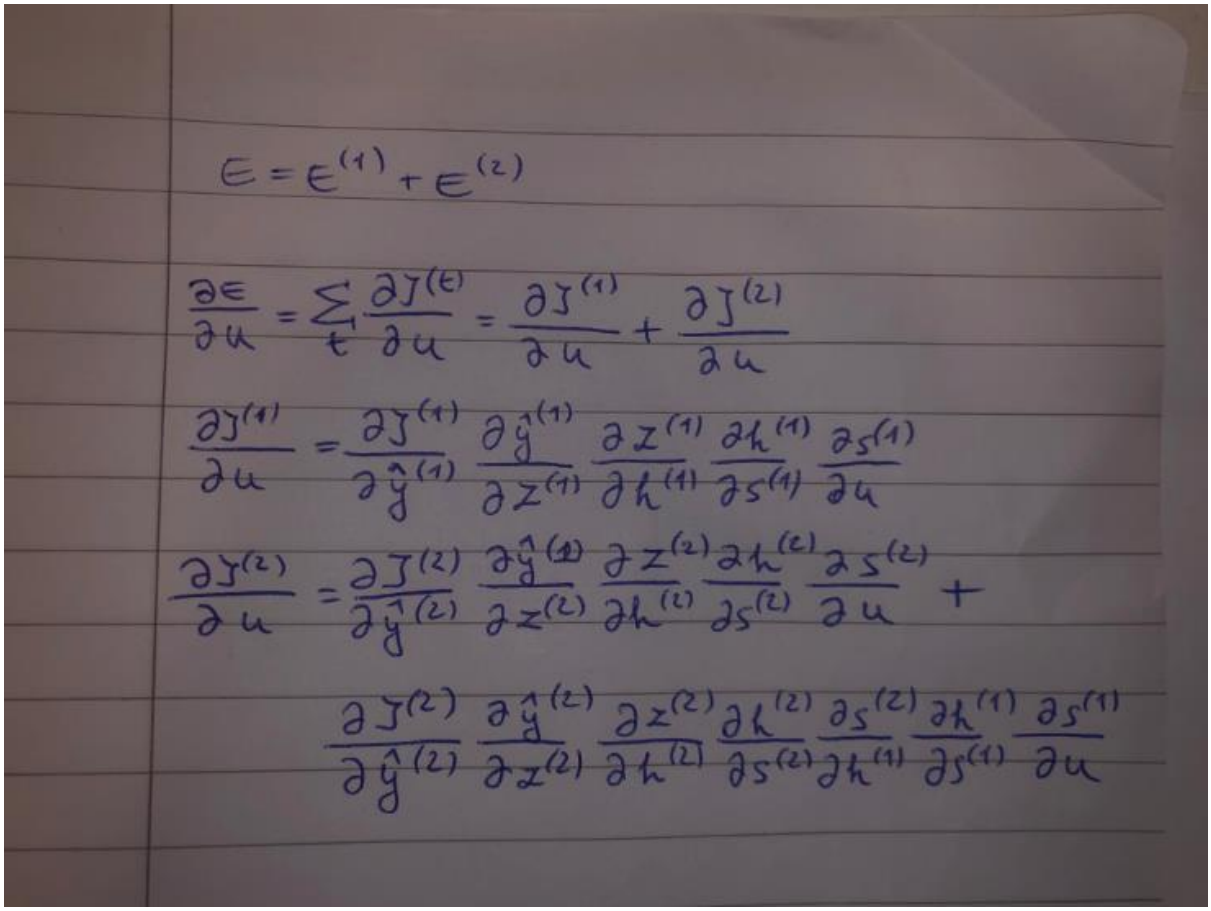
$$LTM_{t-1} f_t + N_t i_t$$

Use Gate: Simply take what's useful from the LTM and STM, and that will become our new LTM. It does so through a similar process as both the learn gate and forget gate, by pushing through the output of the forget gate through a tanh function then multiplies it with the concatenation of STM and E going through a neural network with a sigmoid function output.

$$U_t = \tanh(W_u LTM_{t-1} f_t + b_u)$$

$$V_t = \sigma(W_v[STM_{t-1}, E_t] + b_v)$$

3.



4. The purpose of the autoencoder is to be able to output a lossy, compressed representation of the data, and therefore the autoencoder described in the question is not acceptable, since it would be able to perfectly reconstruct the model. To improve, we could use an activation function in the hidden layers and/or by introducing weight and bias parameters in the hidden layer functions, i.e. $h = \sigma(W x + b)$.

5. How: Iteratively sample from each variable many times. As you do large number of times, you end up with something very close to the theoretical sample of the actual joint distributions.

   When: Used when the distribution is either not known or difficult to sample, but the distribution of each variable is known and easy to sample from.

6. How: Equate the decoder weights to be the transpose of the encoder weights.

   Why: Reduces the number of parameters in the model by half, resulting in lower risk of overfitting the training data while making the model converge faster during training.