Review Questions 3
ID2223
Group: Nicolas Oulianitski Essipova, Peter Lakatos, Marios Chatiras

1. It is not recommended to initialize all the weights to the same value, because neural nets do not perform symmetry breaking and the training will serve no purpose as neurons would learn the same thing. Weight initialization should follow some type of random assigning in a certain range. Regarding the initialization of bias terms, it needs to be coordinated with the weight setting scheme, but setting them to 0 is compatible with most schemes, since bias is a variable that will change independently for each neuron and the random weights are already enough to avoid redundant learning.

2. ELU: If the network's architecture prevents it from self-normalizing, then ELU is a better choice than SELU. It's also slower to compute than ReLU but typically converges faster which compensates for its slower computation.
   Leaky ReLU: When there is a preference for reducing runtime latency.
   ReLU: Doesn't saturate for positive values, unlike the sigmoid function. Also computes much faster and is very often used for the hidden layers. Preferred over leaky ReLU due to its simplicity.
   tanh: Enables stronger gradients since derivates are higher around 0, but these days ReLU/Leaky ReLU/ELU/SELU are preferred for hidden layers.
   Logistic: For the output if classification with probability 0 to 1 is desired.
   Softmax: When you are dealing with a multiclass classification problem, wherein you want the output layer to classify the probability of each class wherein they're mutually exclusive.

3. Batch normalization tackles the challenge of layers having a different distribution of inputs as we get deeper in the network. Instead of chasing an ever-changing environment, batch norm introduces an additional operation to be applied before the activation functions to create new standardized distributions. This stabilizes the training process and reduces the number of epochs needed.

4. Dropdown does indeed tend to significantly slow down convergence, thus increasing training time – but is worth the time and effort as it typically results in a better model when tuned properly. There is no impact on inference speed since it is only applied during training.

5. The momentum parameter specifies the relationship between the current step and the previous gradients in optimization methods like stochastic gradient descent. It accelerates the process by changing the weights towards their optimum value and helps avoid local minima points. When the momentum is set to a value close to 1, the optimization process could be unable to locate the total minimum by oscillating around it. If the Nesterov parameter is set to true, the optimizer may exhibit a better behaviour by making smaller oscillations and potentially find the total minimum, even with a momentum close to 1.

6. For the layers we have:

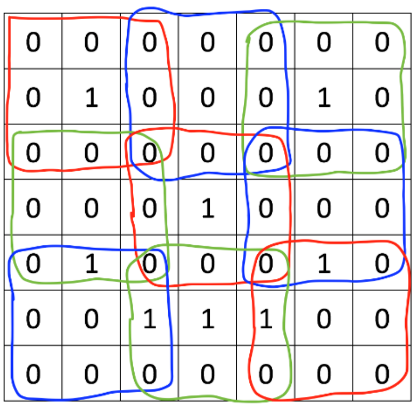
   Lowest Layer:
   3x3 Filters, 3 channels for RGB
   Weights: 3x3x3 = 27 + 1 bias weight
   For 100 feature maps we have 100 x 28 = 2800 parameters

   Middle layer:
   Input: 100
   Filters: 3x3
   Weights: 3x3x100 = 900 + 1 bias weight
   For 200 feature maps we have 200 x 901 = 180200 parameters

   Top layer:
   Input: 200
   Filters: 3x3
   Weights: 3x3x200 = 1800 + 1 bias weight
   For 400 feature maps we have 400 x 1801 = 720400 parameters

   In total: 720,400 + 180200 + 2800 = 903,400 weights

7. The output will be 3x3 and comes from multiplying the 9 shown matrices with the filter. Since the stride was 2, the receptive field was shifted by 2 both horizontally and vertically.



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

   With the output:

0	0	0
1	0	1
0	1	1