ID2223 - Scalable Machine Learning and Deep Learning

- Review Questions 3 -

PAOLO TETA RALFS ZANGIS

teta | zangis @kth.se

November 27, 2021

1 Question 1

In a neural network, it's better to initialize the weights randomly given some boundaries limits. For instance, considering the *ReLU* activation function, we can use the He initialization which assumes random weights initialization from a standard normal distribution with a specific variance of the output close to 1, specifically using $\sigma^2 = \frac{2}{N}$, where N is the number of input neurons to a particular layer whose weights are being initialized. So, initializing all the weights to the same value is not recommended because doing this during the training each neuron will learn nothing new, thus it would learn just the same output as others. Considering the bias initialization, it's not a problem setting these terms to 0, we just need to consider the weights scheme of the neural network. Anyway, the bias b (considering a simple linear regression model $y = ax + b$) is a parameter which changes singularly for each neuron of the network and typically is set to a constant value.

2 Question 2

The following activation functions are used for:

- *ELU* $\Longrightarrow$ it avoids the "*dead*" *ReLU* problem, where the weights of the neural network are most likely never updated to a new value due to the vanishing of the gradient, but there is a higher computational cost due to the *exp()* operator. However, this comes with a longer execution time, but a better accuracy of the result.
- *leaky ReLU* $\Longrightarrow$ also this variant avoids the "*dead*" *ReLU* problem and has better runtime performance rather than *ELU* function.
- *ReLU* $\Longrightarrow$ a computationally efficient non-linear activation function, usually used in the hidden layers, that requires less resources than *logistic* and *tanh* functions.
- *tanh* $\Longrightarrow$ it provides similar results to the *logistic* function, but offers faster convergence.
- *logistic* $\Longrightarrow$ it's used for the binary classification problem where the probability p_i of the class is between 0 and 1.
- *softmax* $\Longrightarrow$ it's used for the multiclass classification problem, in which the output layer has to classify the probability p_i of each class considering that they are mutually exclusive.

3 Question 3

In order to overcome the vanishing gradient problem we can use the batch normalization technique. It makes the learning process of layers in the network more independent of each other, considering that the distribution of each input layer changes during the training process as we go deeper through the network. Batch normalization introduces an additional operation in the model before the activation function of each layer, thus it allows to scale and normalize the resulting distributions. As a result, this technique improves the training process and allows to decrease the number of epochs required. The formulations are the following:

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$z^{(i)} = \gamma \hat{x}^{(i)} + \beta$$

where $\mu = \frac{1}{m_B} \sum_{i=1}^{m_B} x^{(i)}$ and $\sigma^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (x^{(i)} - \mu_B)^2$.

4 Question 4

Dropout technique is used to avoid overfitting on the dataset during the training process. The idea is that at each training step, each neuron drops out temporarily with a probability p. Thus, it may increase the total training time of the neural network, but the final result is usually better given a proper tuning of the parameters. As said before, dropout is just applied during the training, after which all the neurons don't get dropped anymore, so there is no consequence on the speed and execution time of inference.

5 Question 5

Given a CNN composed of three convolutional layers and the input images as RGB images of 200×300 pixels, we have the following:
LOWEST LAYER $\longrightarrow$ considering 3 channels as input and 1 single feature map we have $3 \times 3 \times 3 = 27$ weights plus 1 bias weight, thus for 100 feature maps we have $28 \times 100 = 2800$ parameters.
MIDDLE LAYER $\longrightarrow$ considering 100 as input and 1 single feature map we have $3 \times 3 \times 100 = 900$ weights plus 1 bias weight, thus for 200 feature maps we have $901 \times 200 = 180200$ parameters.
TOP LAYER $\longrightarrow$ considering 200 as input and 1 single feature map we have $3 \times 3 \times 200 = 1800$ weights plus 1 bias weight, thus for 400 feature maps we have $1801 \times 400 = 720400$ parameters.s
Thus, the total number of parameters w is the sum of the previous results: 903400 weights.

6 Question 6

Given a CNN with only one convolutional layer and a 3×3 filter with a stride of 2, the output size is 3×3 considering the following formula $\frac{W-K+2P}{S} + 1$. Here, W is the input size ($W = 7$), K is the kernel size ($K = 3$), P is the padding ($P = 0$) and S is the stride ($S = 2$).
So, the normalized output of the convolutional layer is the following matrix:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0.11 & 0 & 0.11 \\ 0 & 0.11 & 0.11 \end{bmatrix}$$