1 Activation Functions

1.1 First Question

If we only use linear layers with linear activation functions, we would only learn a linear function over the input, since a combination of linear transformations can be replaced by a single linear transformation.

1.2 Second Question

- The ReLU function:

$$\text{ReLU}(z) = \begin{cases} z & z > 0 \\ 0 & z <= 0 \end{cases} \tag{1}$$

$$\frac{\partial \,\text{ReLU}(z)}{\partial z} = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases} \tag{2}$$

  Note that the derivative at $z = 0$ is undefined!

- The sigmoid function: We start off with a basic derivation

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

$$\frac{\partial \sigma(z)}{\partial z} = \frac{\partial (1 + e^{-z})^{-1}}{\partial z} \tag{4}$$

$$= -1 \cdot (1 + e^{-z})^{-2} \cdot (-e^{-z}) \tag{5}$$

$$= \frac{e^{-z}}{(1 + e^{-z})^2} \tag{6}$$

  We can simplify by re-writing the terms in the derivative in terms of $\sigma(z)$:

$$1 + e^{-z} = \frac{1}{\sigma(z)} \tag{7}$$

$$e^{-z} = \frac{1 - \sigma(z)}{\sigma(z)} \tag{8}$$

  Plugging this in then yields

$$\frac{\partial \sigma(z)}{\partial z} = \frac{1 - \sigma(z)}{\sigma(z)} \cdot \sigma(z)^2 \tag{9}$$

$$= (1 - \sigma(z)) \cdot \sigma(z) \tag{10}$$

- The tanh function:

$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{11}$$

$$\frac{\partial \tanh(z)}{\partial z} = \frac{2 \cdot e^{2z} \cdot (e^{2z}+1) - (e^{2z}-1) \cdot 2 \cdot e^{2z}}{(e^{2z}+1)^2} \tag{12}$$

$$= \frac{2e^{4z} + 2e^{2z} - 2e^{4z} + 2e^{2z}}{(e^{2z}+1)^2} \tag{13}$$

$$= \frac{e^{4z} + 2e^{2z} - e^{4z} + 2e^{2z}}{(e^{2z}+1)^2} \tag{14}$$

$$= \frac{e^{4z} + 2e^{2z} + 1 - e^{4z} + 2e^{2z} - 1}{(e^{2z}+1)^2} \tag{15}$$

$$= \frac{(e^{4z} + 2e^{2z} + 1) - (e^{4z} - 2e^{2z} + 1)}{(e^{2z}+1)^2} \tag{16}$$

$$= \frac{(e^{2z}+1)^2 - (e^{2z}-1)^2}{(e^{2z}+1)^2} \tag{17}$$

$$= \frac{(e^{2z}+1)^2}{(e^{2z}+1)^2} - \frac{(e^{2z}-1)^2}{(e^{2z}+1)^2} \tag{18}$$

$$= 1 - \tanh^2(z) \tag{19}$$

1.3 Third Question

ReLU as an activation function:

- Fixes vanishing gradient problems.
- Not computationally expensive to calculate.
- ReLU units are fragile during training and can 'die'. (Can be overcome with Leaky ReLU, Parametric ReLU)

Sigmoid as an activation function:

- Computationally more expensive than ReLU.
- It is not zero centered.
- It has a smooth gradient.
- Can be used on the output layer to model probability in binary classification problems.
- Suffers from vanishing gradient problems.

Tanh as an activation function:

- Computationally more expensive than ReLU.
- It is zero centered.
- It has a smooth gradient.

- Used more frequently compared to the sigmoid as an activation function in hidden layers.
- Suffers from vanishing gradient problems.

The choice of the activation function usually depends on the task at hand, however, the most used activation function in practice is the ReLU activation function.

1.4 Fourth Question

$$P(y = i|x) \approx \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Given the unnormalized log probability scores z per class with the following values $z_A = 7, z_B = 1, z_C = 2.2$.

- First, we compute the unnormalized probability scores:
  - $e^{z_A} = e^7 \approx 1097.$
  - $e^{z_B} = e^1 \approx 2.72,$
  - $e^{z_C} = e^{2.2} \approx 9.03,$

  Given those, the probabilitiy scores for the classes will be:
  - Class A:

$$P(y = A|x) \approx \text{softmax}(z_A) = \frac{1097}{1097 + 2.72 + 9.03} \approx 98.9\% \tag{20}$$

  - Class B:

$$P(y = B|x) \approx \text{softmax}(z_B) = \frac{2.72}{1097 + 2.72 + 9.03} \approx 0.25\% \tag{21}$$

  - Class C:

$$P(y = C|x) \approx \text{softmax}(z_C) = \frac{9.03}{1097 + 2.72 + 9.03} \approx 0.81\% \tag{22}$$

- The predicted class label will be A as it has the highest probability among the given classes.
- Given our classes $Y = \{A, B, C\}$ and the true label $y = A$, the loss for the single sample x will be:

$$J(w, x) = -\sum_{i \in Y} 1_{y_i = i} \cdot \log P(y = i \mid x) \tag{23}$$

$$= -1 \cdot \log(0.989) \approx 0.011 \tag{24}$$

2 Neural Networks

2.1 First Question

Basis functions are functions that apply non-linear transformations to the input features, they are difficult to design and depend on the task at hand. Neural Networks can be thought as learning basis functions in an automatic way from the task at hand to achieve optimal performance.

2.2 Second Question

The following calculations correspond to the weights of the neurons connecting them with the neurons of the previous layer and the biases of each neuron.

- The first layer will have: $32 \cdot 6 + 32 = 224$ weights
- The second layer will have: $32 \cdot 32 + 32 = 1056$ weights
- The third layer will have: $32 \cdot 32 + 32 = 1056$ weights
- The output layer will have: $4 \cdot 32 + 4 = 132$ weights

In total, the network has 2468 weights.

2.3 Third Question

The network can be written as

$$h^{(1)} = \text{ReLU}\left(w_{1,1}^{(1)} \cdot x\right) \tag{25}$$

$$h^{(2)} = \text{ReLU}\left(w_{1,1}^{(2)} \cdot h^{(1)}\right) \tag{26}$$

Given the input, weight $w_{1,1}^{(1)}$ and the ReLU activation function, the network will always predict 0 for every example. Decreasing the weight $w_{1,1}^{(1)}$ will not change the loss, since, the output will be the same. On the other hand, increasing the weight to above 0 has an effect and can decrease the loss (e.g., set $w_{1,1}^{(1)} = 1$).

2.4 Bonus Question

Let us expand

$$h^{(1)} = \text{ReLU}\left(a^{(1)}\right), \quad a^{(1)} = w_{1,1}^{(1)} \cdot x. \tag{27}$$

Then we obtain the derivative for $w_{1,1}^{(1)}$ as

$$\frac{\partial J(w)}{\partial w_{1,1}^{(1)}} = \frac{\partial J(w)}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}} \cdot \frac{\partial h^{(1)}}{\partial w_{1,1}^{(1)}} \tag{28}$$

$$= \frac{\partial J(w)}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}} \cdot \frac{\partial h^{(1)}}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial w_{1,1}^{(1)}}. \tag{29}$$

Since we only have positive x values, $a^{(1)}$ will always be negative. The derivative of ReLU with a negative value as input is 0 and therefore, the gradient at this ReLU will always be zero, i.e.,

$$\frac{\partial h^{(1)}}{\partial a^{(1)}} = 0. \tag{30}$$

This means that (with our dataset) any gradient based optimization procedure will not change this weight. This is a typical case of a 'dead' ReLU, where the network is not learning.
Note that if we would set $w_{1,1}^{(2)} < 0$, this would not even depend on the input!