IFT6390-fundamentals of machine learning
Assignment 3

Jonathan Guymont, Marzieh Mehdizadeh, Mohammad Bardestani

Question 1

(a)

$$\begin{aligned}
0.5(\tanh(0.5x)+1) =&0.5(\frac{e^{0.5x}-e^{-0.5x}}{e^{0.5x}+e^{-0.5x}}+1)\\
=&0.5(\frac{1-e^{-x}}{1+e^{-x}}+\frac{1+e^{-x}}{1+e^{-x}})\\
=&0.5\frac{2}{1+e^{-x}}\\
=&\frac{1}{1+e^{-x}}
\end{aligned} \tag{1}$$

(b)

$$\begin{aligned}
\log \text{sigmoid}(x) =&\log(1+e^{-x})^{-1}\\
=&-\log(1+e^{-x})\\
=&-\text{softmax}(-x)
\end{aligned} \tag{2}$$

(c)

$$\begin{aligned}
\frac{d}{dx}\text{sigmoid}(x) =&\frac{d}{dx}(1+e^{-x})^{-1}\\
=&-(1+e^{-x})^{-2}\frac{d}{dx}1+e^{-x}\\
=&(1+e^{-x})^{-2}e^{-x}\\
=&(1+e^{-x})^{-1}\frac{e^{-x}}{1+e^{-x}}\\
=&(1+e^{-x})^{-1}(\frac{1+e^{-x}-1}{1+e^{-x}})\\
=&(1+e^{-x})^{-1}(1-\frac{1}{1+e^{-x}})
\end{aligned} \tag{3}$$

(4)

$$\begin{aligned}\tanh'(x) =&(\frac{e^x - e^{-x}}{e^x + e^{-x}})' \\ =&\frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\ =&\frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\ =&1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\ =&1 - \tanh^2(x)\end{aligned} \tag{4}$$

(5) $\text{sign}(x) = -1 + 2 \cdot \mathbb{1}_{x>0}$

(6) $\text{abs}'(x) = \text{sign}(x)$

(7) $\text{rect}'(x) = \mathbb{1}_{x>0}$

(8) Let $f(\boldsymbol{x}) = \sum_{x_i \in \boldsymbol{x}} x_i^2$, then $f'(\boldsymbol{x}) = (f'_{x_1}, ..., f'_{x_{|\boldsymbol{x}|}}) = (2x_1, ..., 2x_{|\boldsymbol{x}|})$

(9) Let $f(\boldsymbol{x}) = \sum_{x_i \in \boldsymbol{x}} |x_i|$, then $f'(\boldsymbol{x}) = (f'_{x_1}, ..., f'_{x_{|\boldsymbol{x}|}}) = (\text{sign}(x_1), ..., \text{sign}(x_{|\boldsymbol{x}|}))$

Gradient Computation for Parameters optimizations in a neural net for multiclass classification

(1) The dimension of $\boldsymbol{b}^{(1)}$ is $d_h \times 1$. The formula of the preactivation vector is

$$\boldsymbol{h}^a = \boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$$

and the formula for obtaining the value of the element j is

$$\boldsymbol{h}^a_j = \boldsymbol{W}^{(1)}_{\boldsymbol{j,\cdot}}\boldsymbol{x} + \boldsymbol{b}^{(1)}_j = \boldsymbol{b}^{(1)}_j + \sum_{i=1}^{d} \boldsymbol{W}^{(1)}_{\boldsymbol{j,i}}\boldsymbol{x}_i.$$

The output vector of the activation is given by

$$\boldsymbol{h}^s = \text{relu}(\boldsymbol{h}^a)$$

where relu$(\cdot)$ is applied element wise, i.e. $\boldsymbol{h}^s_j = \max(0, \boldsymbol{h}^a_j)$, $j = 1, ..., d_h$.

(2) The dimension of $\boldsymbol{W}^{(2)}$ is $m \times d_h$ and the dimension of $\boldsymbol{b}^{(2)}$ is $m \times 1$.

$$\boldsymbol{o}^a = \boldsymbol{W}^{(2)}\boldsymbol{h}^s + \boldsymbol{b}^{(2)}$$

$$\boldsymbol{o}^a_k = \boldsymbol{W}^{(2)}_{k,\cdot}\boldsymbol{h}^s + \boldsymbol{b}^{(2)}_k = \boldsymbol{b}^{(2)}_k + \sum_{i=1}^{d_h} \boldsymbol{W}^{(2)}_{\boldsymbol{k,i}}\boldsymbol{h}^s_i$$

for $k = 1, .., m$.

(3)

$$\boldsymbol{o}_k^s = \frac{\exp(\boldsymbol{o}_k^a)}{\sum_{k=1}^m \exp(\boldsymbol{o}_k^a)} \tag{5}$$

They are all positive because $\exp\colon \mathbb{R} \mapsto \mathbb{R}^+$. Also a sum of positive number is positive. And the ratio of a posive number over a positive number is also positive.

$$\sum_{k=1}^m \boldsymbol{o}_k^s = \frac{1}{\sum_{k=1}^m \exp(\boldsymbol{o}_k^a)} \sum_{k=1}^m \exp(\boldsymbol{o}_k^a) = 1$$

(4) Let $Z = \sum_{k=1}^m \exp(\boldsymbol{o}_k^a)$. Then $\boldsymbol{o}^s = \frac{1}{Z}(\exp(\boldsymbol{o}_1^a), ..., \exp(\boldsymbol{o}_m^a))^\top$ and

$$L(\boldsymbol{x}, y) = -\log \text{onehot}_m(y)(\exp(\boldsymbol{o}_1^a(\boldsymbol{x}))/Z, ..., \exp(\boldsymbol{o}_m^a(\boldsymbol{x}))/Z)^\top = -\log \text{onehot}_m(y)\boldsymbol{o}^s(\boldsymbol{x})$$

where $\text{onehot}_m(y)$ is a $1 \times m$ onehot representation for y.

(5) $\hat{R}$ is an estimation of the expected value of the loss function (minus the loglikelihood in our case)

$$\hat{R} = \frac{1}{n}\sum_{i=1}^n L(x^{(i)}, y^{(i)}) = \frac{1}{n}\sum_{i=1}^n -\log \text{onehot}_m(y^{(i)})\boldsymbol{o}^s(\boldsymbol{x}^{(i)})$$

The set of trainable parameters is $\boldsymbol{\theta} = \{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}\}$. The number of scalar parameters is $n_\theta = d_h \cdot d + d_h + m \cdot d_h + m = d_h(d+1) + m(d_h+1)$.

Optimization problem. First we need to initialize the parameters properly. To find the parameters that minimize the loss, we need to compute the derivative of the loss function w.r.t each parameters. Then we update each parameters by moving them in the opposite direction of their gradient (since we minimize). We repeat this step until a stopping criterion is met (e.g. maximum number of iteration is reached when using early stopping).

(6)

Algorithm 1 Pseudocode for Batch Gradient Descent

```
Require: Step size $\eta$
Require: Initial parameter $\boldsymbol{\omega}_0$
Require: Number of iterations $T$
  for $i = 1$ to $T$ do
    Compute gradient $\boldsymbol{g}_t = \frac{1}{m}\nabla_{\boldsymbol{\omega}} \sum_i L(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\omega}_t = \boldsymbol{\omega}_{t-1} - \eta \boldsymbol{g}_t$
  end for
```

(7)

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{o}^a} L =& \frac{\partial}{\partial \boldsymbol{o}^a} - \log \operatorname{onehot}_m(y)\boldsymbol{o}^s \\
=& - \frac{1}{\operatorname{onehot}_m(y)\boldsymbol{o}^s} \frac{\partial}{\partial \boldsymbol{o}^a} \operatorname{onehot}_m(y)\boldsymbol{o}^s \\
=& - \frac{1}{\operatorname{onehot}_m(y)\boldsymbol{o}^s} \frac{\partial}{\partial \boldsymbol{o}^a} \operatorname{onehot}_m(y)\operatorname{softmax}(\boldsymbol{o}^a) \\
=& - \frac{1}{\operatorname{onehot}_m(y)\boldsymbol{o}^s} \operatorname{onehot}_m(y)\operatorname{softmax}(\boldsymbol{o}^a)(\operatorname{onehot}_m(y) - \operatorname{softmax}(\boldsymbol{o}^a)) \\
=& - (\operatorname{onehot}_m(y) - \operatorname{softmax}(\boldsymbol{o}^a))
\end{aligned} \tag{6}$$

(8)

```
onehot = np.zeros(m)
onehot[y] = 1
grad_oa = os - onehot
```

(9) From the previous exercise we have

$$\frac{\partial L}{\partial \boldsymbol{o}^a} = \boldsymbol{o}^s - \operatorname{onehot}_m(y).$$

and so

$$\frac{\partial L}{\partial o_i^a} = o_i^s - \operatorname{onehot}_m(y)_i, \tag{7}$$

where $\operatorname{onehot}_m(y)_i$ is the i'th component of $\operatorname{onehot}_m(y)$. From (**??**) we deduce that

$$\frac{\partial o_i^a}{\partial W_{kj}^{(2)}} = \begin{cases} h_j^s & i = k; \\ 0 & \text{otherwise.} \end{cases}$$

Moreover from (**??**)

$$\frac{\partial o_i^a}{\partial b_k^{(2)}} = \begin{cases} 1 & i = k; \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Putting these together we obtain

$$\begin{aligned}
\frac{\partial L}{\partial W_{kj}^{(2)}} &= \sum_{i=1}^{m} \frac{\partial L}{\partial o_i^a} \frac{\partial o_i^a}{\partial W_{kj}^{(2)}} = \left(o_k^s - \operatorname{onehot}_m(y)_k\right) h_j^s \\
\frac{\partial L}{\partial b_k^{(2)}} &= \sum_{i=1}^{m} \frac{\partial L}{\partial {o_i}^a} \frac{\partial o_i}{\partial b_k^{(2)}} = o_k^s - \operatorname{onehot}_m(y)_k.
\end{aligned} \tag{9}$$

(10) Since $\frac{\partial \boldsymbol{o}^a}{\partial \boldsymbol{W}^{(2)}} = \boldsymbol{h}^s$ and $\frac{\partial L}{\partial \boldsymbol{o}^a} = \boldsymbol{o}^s - \operatorname{onehot}_m(y)$, then

$$\frac{\partial L}{\partial \boldsymbol{W}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{o}^a} \cdot \frac{\partial \boldsymbol{o}^a}{\partial \boldsymbol{W}^{(2)}} = (\boldsymbol{o}^s - \operatorname{onehot}_y(y))\boldsymbol{h}^s. \tag{10}$$

Moreover, since $\frac{\partial \boldsymbol{o}^a}{\partial \boldsymbol{b}^{(2)}} = \mathbf{1}$, we obtain

$$\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = \frac{\partial L}{\partial \boldsymbol{o}^a} \cdot \frac{\partial \boldsymbol{o}^a}{\partial \boldsymbol{b}^2} = \boldsymbol{o}^s - \text{onehot}_m(y). \tag{11}$$

We remark that $\boldsymbol{W}^2 \in \mathbb{R}^{d_h \times m}$, $\boldsymbol{b}^{(2)} \in \mathbb{R}^m$, $\boldsymbol{o}^a \in \mathbb{R}^m$, $\boldsymbol{h}^s \in \mathbb{R}^{d_h}$ and $\frac{\partial L}{\partial \boldsymbol{o}^a} \in \mathbb{R}^m$.

```
grad_b_2 = grad_oa
grad_W_2 = grad_ha.dot(X)
```

(11) From (**??**) we see that

$$\frac{\partial o_k^a}{\partial h_j^s} = W_{kj}^{(2)}.$$

From (7) we deduce that

$$\frac{\partial L}{\partial h_j^s} = \sum_{k=1}^{m} \left(o_k^s - \text{onehot}_m(y)_k\right) w_{kj}^{(2)}.$$

We recall that $o_k^s - \text{onehot}_m(y)_k$ is the k'th component of the vector $\boldsymbol{o}^s - \text{onehot}_m(y)$
(12)

$$\nabla L = (\boldsymbol{o}^s - \text{onehot}_m(y))\boldsymbol{W}^{(2)\top}.$$

```
grad_hs = grad_oa.T.dot(self.W_2)
```

(13) We jest need to emphasis that

$$\frac{\partial h_j^s}{\partial h_j^a} = \begin{cases} 0 & h_j^a < 0; \\ 1 & h_j^a > 0. \end{cases} \tag{12}$$

Note that $\frac{\partial h_j^s}{\partial h_j^a}$ is not defined $h_j^a = 0$.
(14) We have

$$\frac{\partial L}{\partial \boldsymbol{h}^a} = \frac{\partial L}{\partial \boldsymbol{h}^s} \frac{\partial \boldsymbol{h}^s}{\partial \boldsymbol{h}^a}.$$

where the components of $\frac{\partial \boldsymbol{h}^s}{\partial \boldsymbol{h}^a}$ is computes as in (12).

Part 15 From the chain rule we have

$$\frac{\partial L}{\partial W_{kj}^{(1)}} = \sum_{i=1}^{d_h} \frac{\partial L}{\partial h_i^a} \frac{\partial h_i^a}{\partial W_{kj}^{(1)}}.$$

Since

$$\boldsymbol{h}^a = \boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)},$$

we get

$$\frac{\partial h_i^a}{\partial W_{kj}^{(1)}} = x_k.$$

For $\boldsymbol{b}^{(1)}$, from the chain rule we obtain

$$\frac{\partial L}{\partial b_j^{(1)}} = \sum_{i=1}^{d_h} \frac{\partial L}{\partial h_i^a} \frac{\partial h_i^a}{\partial b_j^{(1)}}.$$

But

$$\frac{\partial h_i^a}{\partial b_j^{(1)}} = 1.$$

(16)

$$\frac{\partial L}{\partial \boldsymbol{W}^{(1)}} = \frac{\partial L}{\partial \boldsymbol{h}^a}\boldsymbol{x}$$

and

$$\frac{\partial L}{\partial \boldsymbol{b}^1} = \mathbf{1}.$$

(17)

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial \boldsymbol{h}^a}\frac{\partial \boldsymbol{h}^a}{\partial \boldsymbol{x}}.$$

But

$$\frac{\partial \boldsymbol{h}^a}{\partial \boldsymbol{x}} = \boldsymbol{W}^\top.$$

(18) The gradient L_1 and L_2 regularization in term of two parameters $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ is a follows:

$$\nabla_{\boldsymbol{W^{(1)}}}(L) = \lambda_{11}\text{sign}(\boldsymbol{W^{(1)}}) + 2\lambda_{12}\boldsymbol{W}^{(1)} \tag{13}$$

$$\nabla_{\boldsymbol{W^{(2)}}}(L) = \lambda_{21}\text{sign}(\boldsymbol{W^{(2)}}) + 2\lambda_{22}\boldsymbol{W}^{(2)} \tag{14}$$

where the sign is the matrix of sing of components of each $\boldsymbol{W^{(1)}}$ and $\boldsymbol{W^{(2)}}$.

$$\boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} - \eta\,(\nabla_{\boldsymbol{W^{(1)}}}\mathcal{R} + \nabla_{\boldsymbol{W^{(1)}}}\mathcal{L})$$
$$\boldsymbol{W}^{(2)} \leftarrow \boldsymbol{W}^{(2)} - \eta\,(\nabla_{\boldsymbol{W^{(2)}}}\mathcal{R} + \nabla_{\boldsymbol{W^{(2)}}}\mathcal{L})$$

Question 3: Practical part

Loop backward	Matrix calculus backward
∼40 sec.	∼5 sec.

Table 1: Training time on the cicles dataset. The training set represent 70% if the data. The hidden layer size was set to 10, the learning rate was 0.05, the batch size was 32, the number of epoch was 1000.

6. For loop to matrix calculus

```
def backward(self, X, Y):
    """Backward probagation

    Loop implementation. Afer each iteration the gradient
    of the current example is added to the batch gradient.
    the gradient is divided by the size of the batch.

    Args
        X: (array) input batch of dimension <k x d>
        Y: (array) target batch of dimension <k x 1>
    """
    for x, y in zip(X, Y):
        # x as shape <1 x d>
        # y is int (y is change to onehot in self._onehot())
        self.forward(x, train=True)
        grad_oa    = self.os - self._onehot(y).T
        grad_hs    = grad_oa.T.dot(self.W_hy)
        grad_hs_ha = self._relu_prime(self.ha)
        grad_ha    = grad_hs_ha * grad_hs.T

        self.grad_W_hy  += grad_oa.dot(self.hs.T) / batch_size
        self.grad_b_hy  += grad_oa.reshape(grad_oa.shape[0], 1) / batch_size
        self.grad_W_xh  += grad_ha.dot(x) / batch_size
        self.grad_b_xh  += grad_ha.reshape(grad_ha.shape[0], 1) / batch_size
```

```
def backward(self, X, Y):
    """Backward probagation (without loop)

    Args
        X: (array) input batch of dimension <k x d>
        Y: (array) target batch of dimension <k x 1>
    """

    self.forward(X, train=True)
    grad_oa    = self.os - self._onehot(Y).T
    grad_hs    = grad_oa.T.dot(self.W_hy)
    grad_hs_ha = self._relu_prime(self.ha)
    grad_ha    = grad_hs_ha * grad_hs.T

    self.grad_W_hy  = grad_oa.dot(self.hs.T) / batch_size
    self.grad_b_hy  = np.sum(grad_oa, axis=1).reshape(grad_oa.shape[0], 1) / batch_size
    self.grad_W_xh  = grad_ha.dot(X) / batch_size
    self.grad_b_xh  = np.sum(grad_ha, axis=1).reshape(grad_ha.shape[0], 1) / batch_size
```

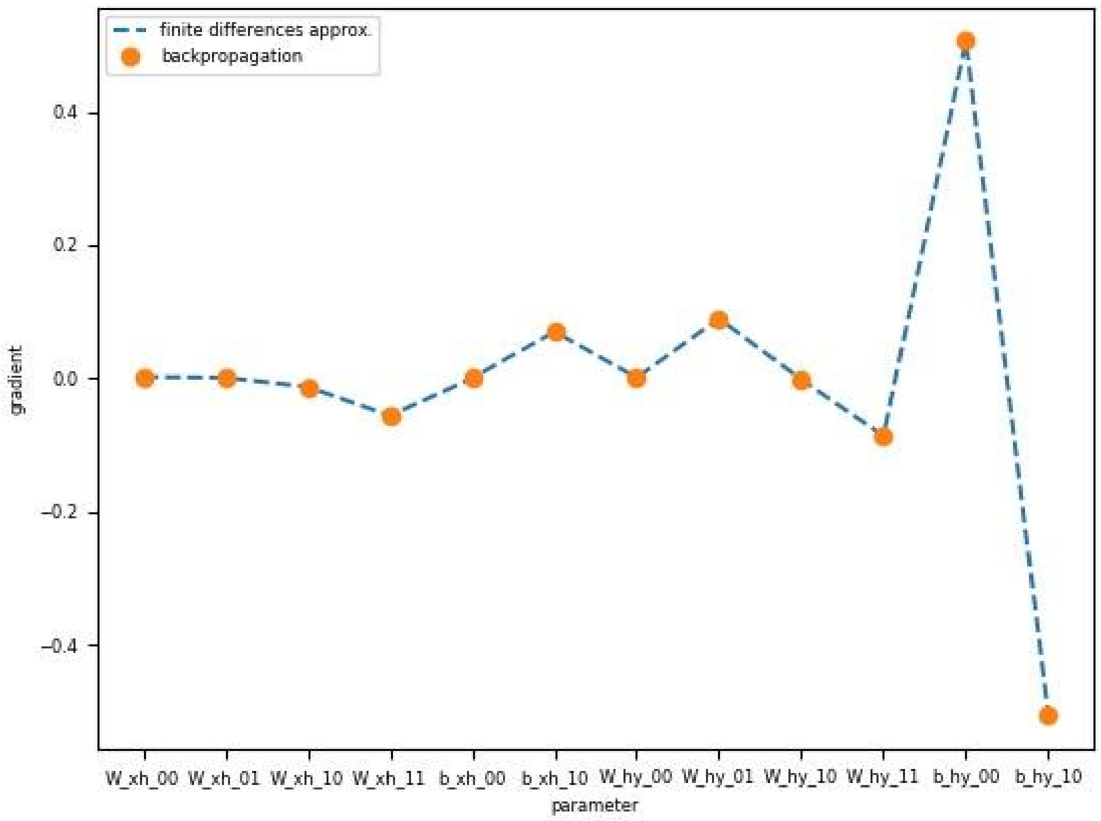

Figure 1: Question 2: Gradients for one example.

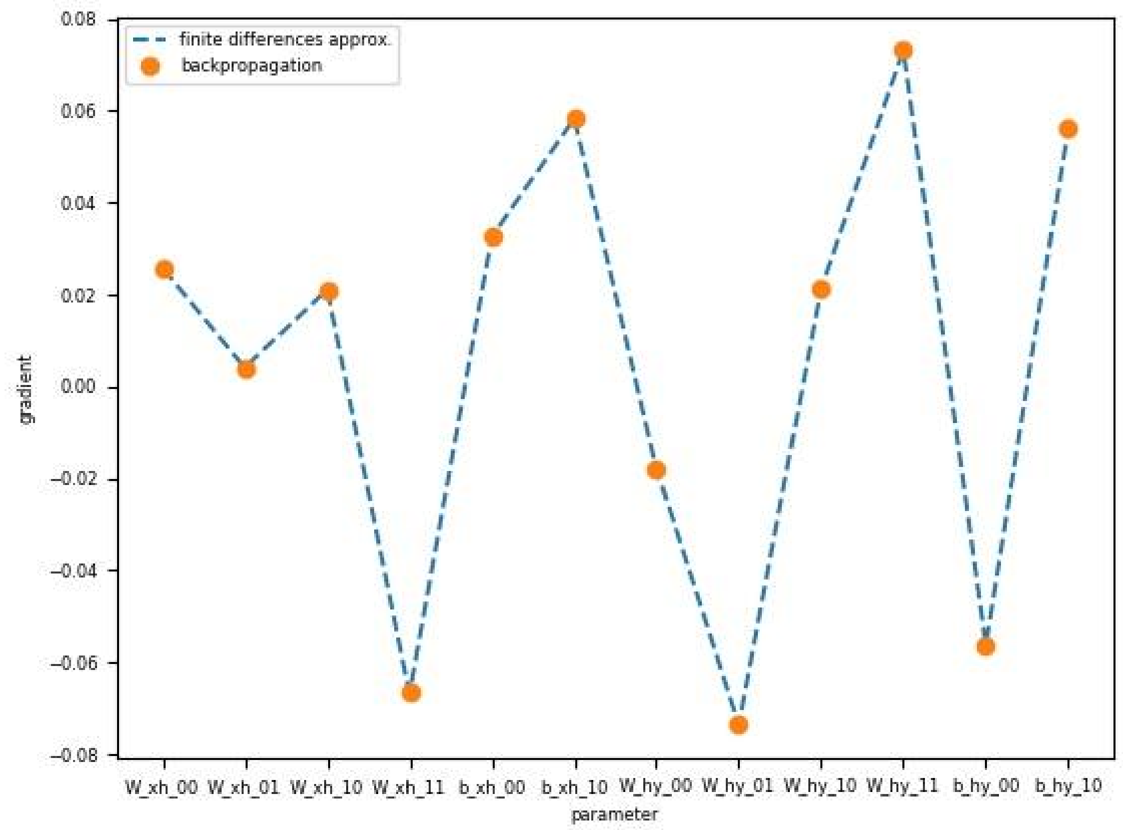

Figure 2: Question 4: Gradients for 10 example.

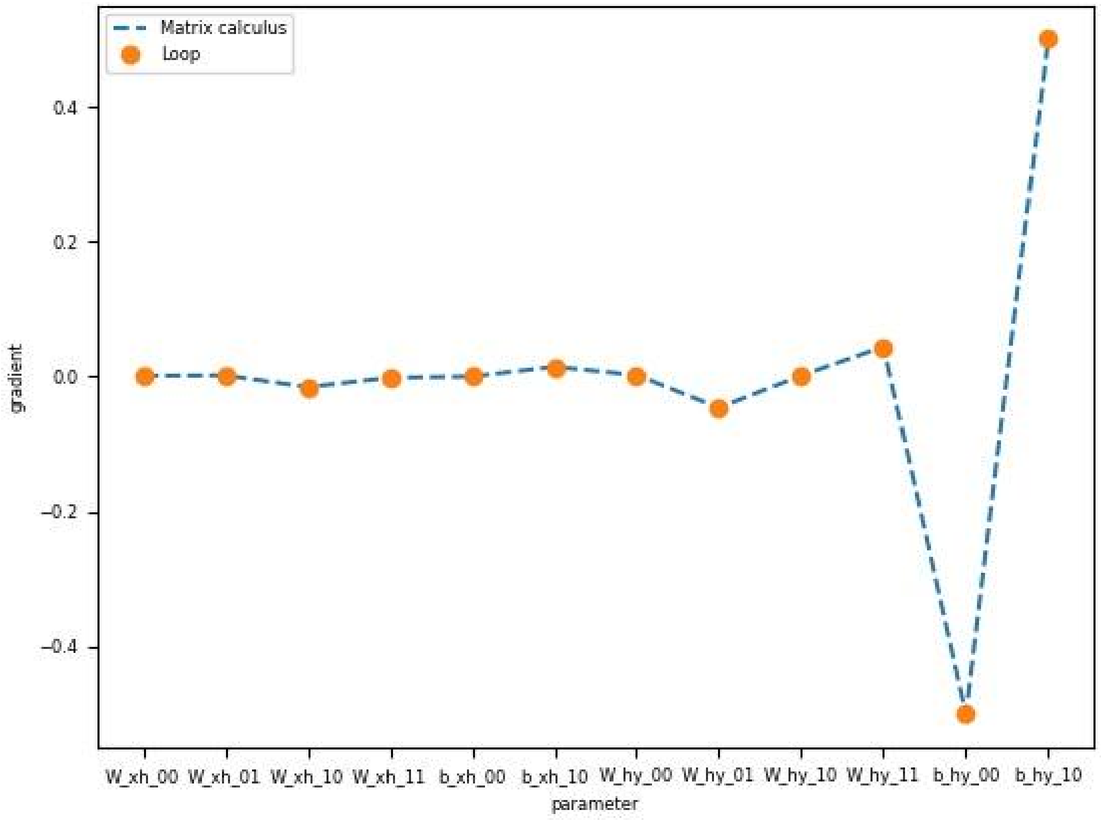

Figure 3: Question 7 a): Gradients for a batch of size 1.

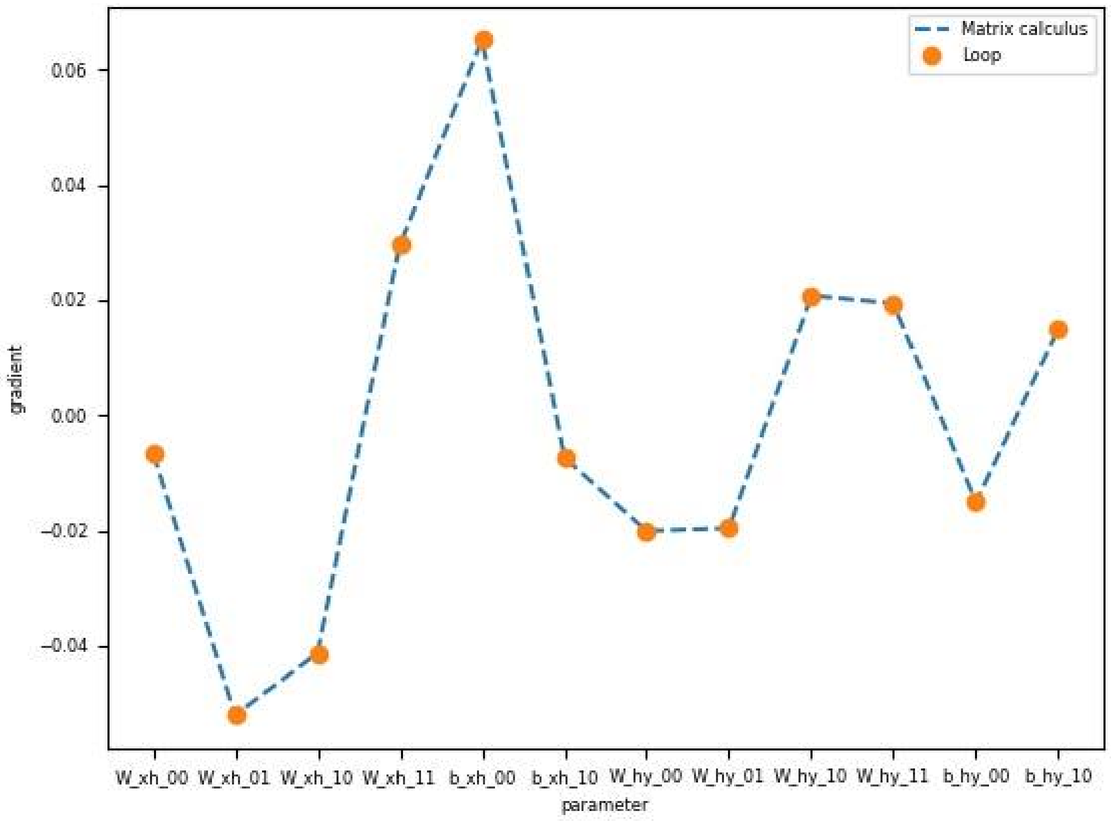

Figure 4: Question 7 b): Gradients for 10 example.

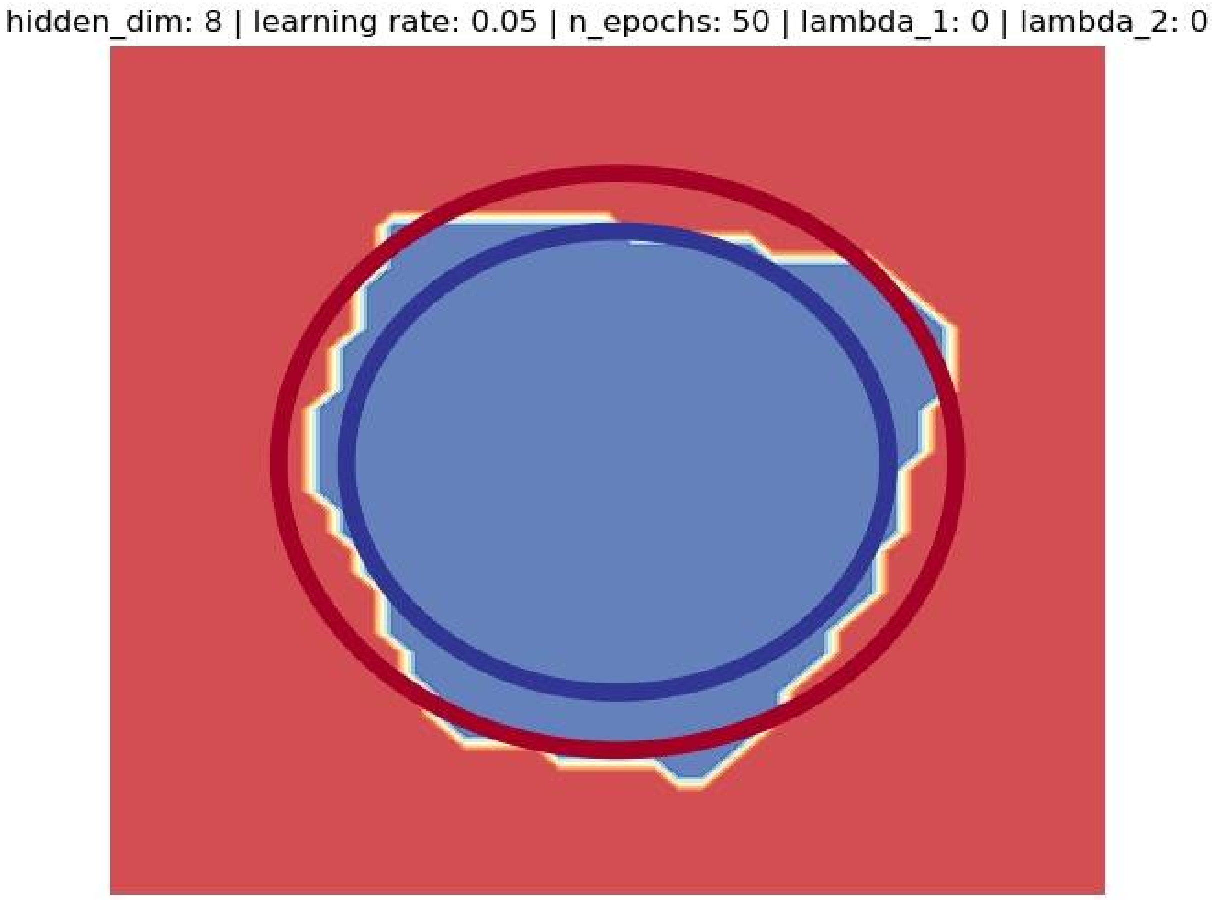

Figure 5: Decision boundary 1

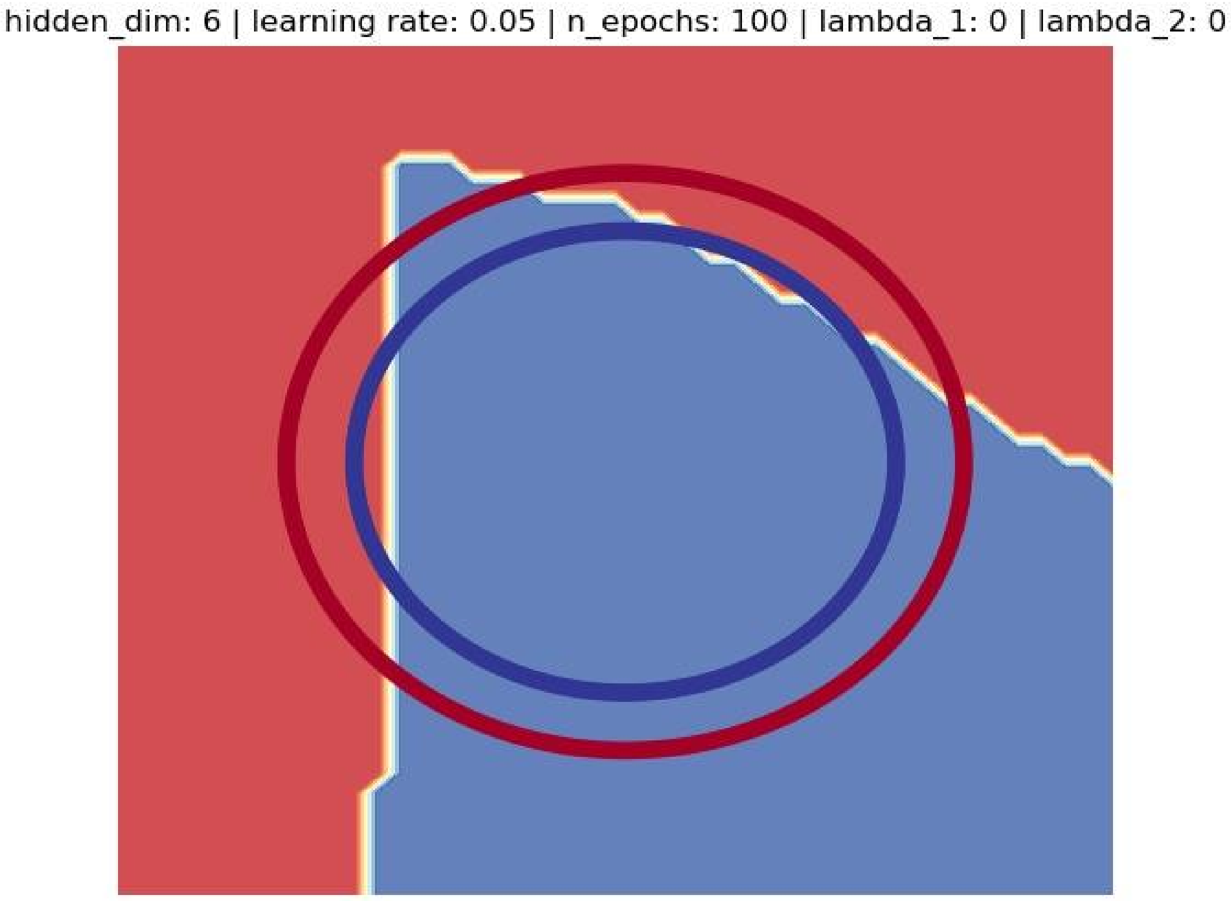

Figure 6: Decision boundary 2

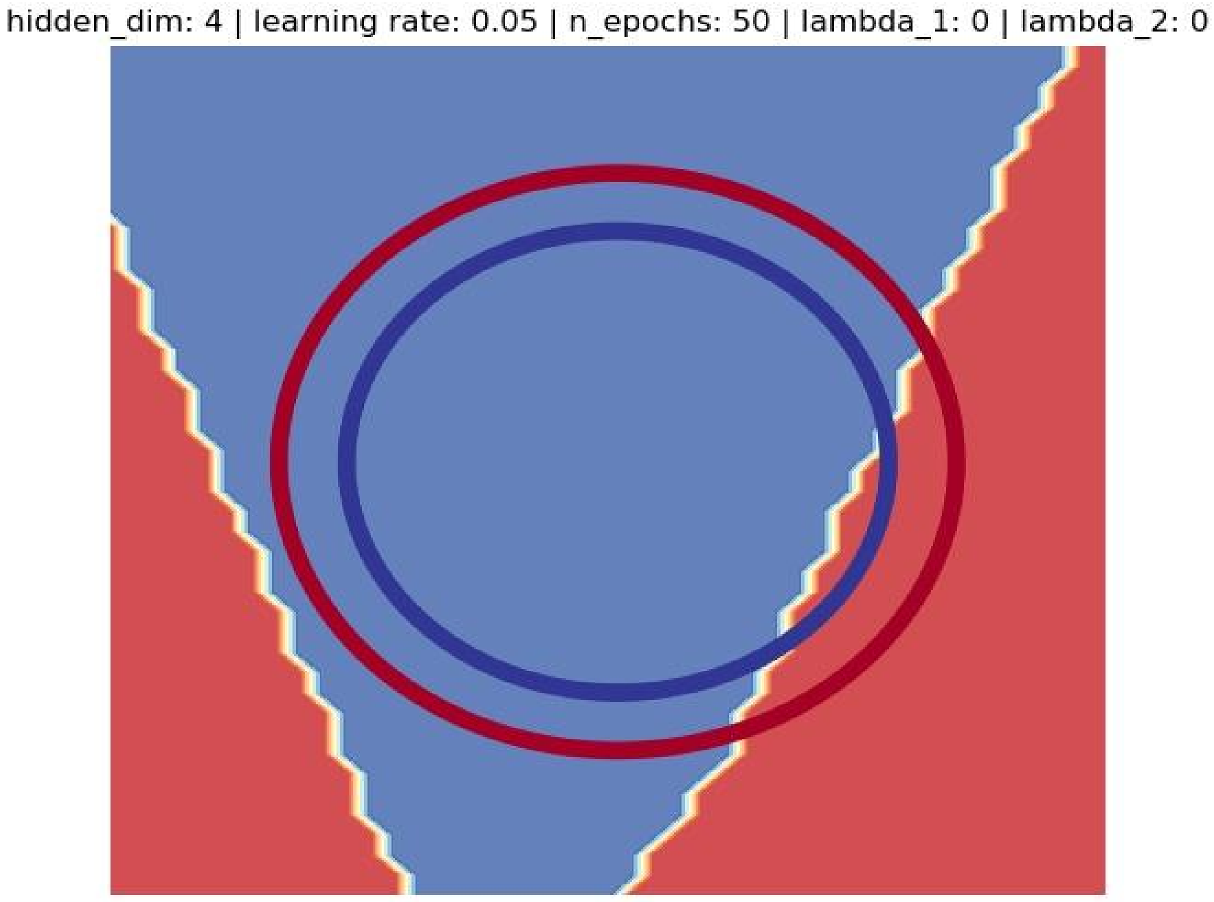

Figure 7: Decision boundary 3

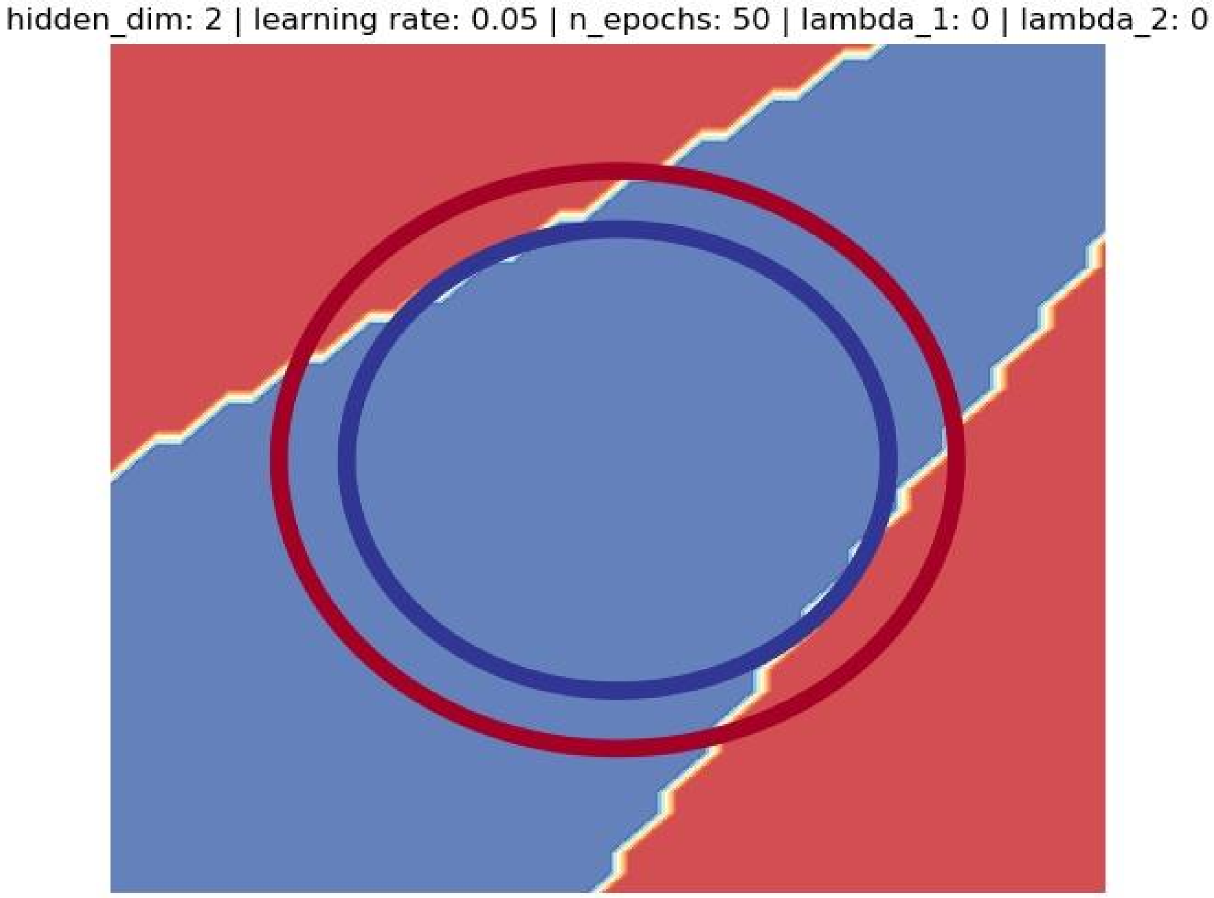

Figure 8: Decision boundary 4

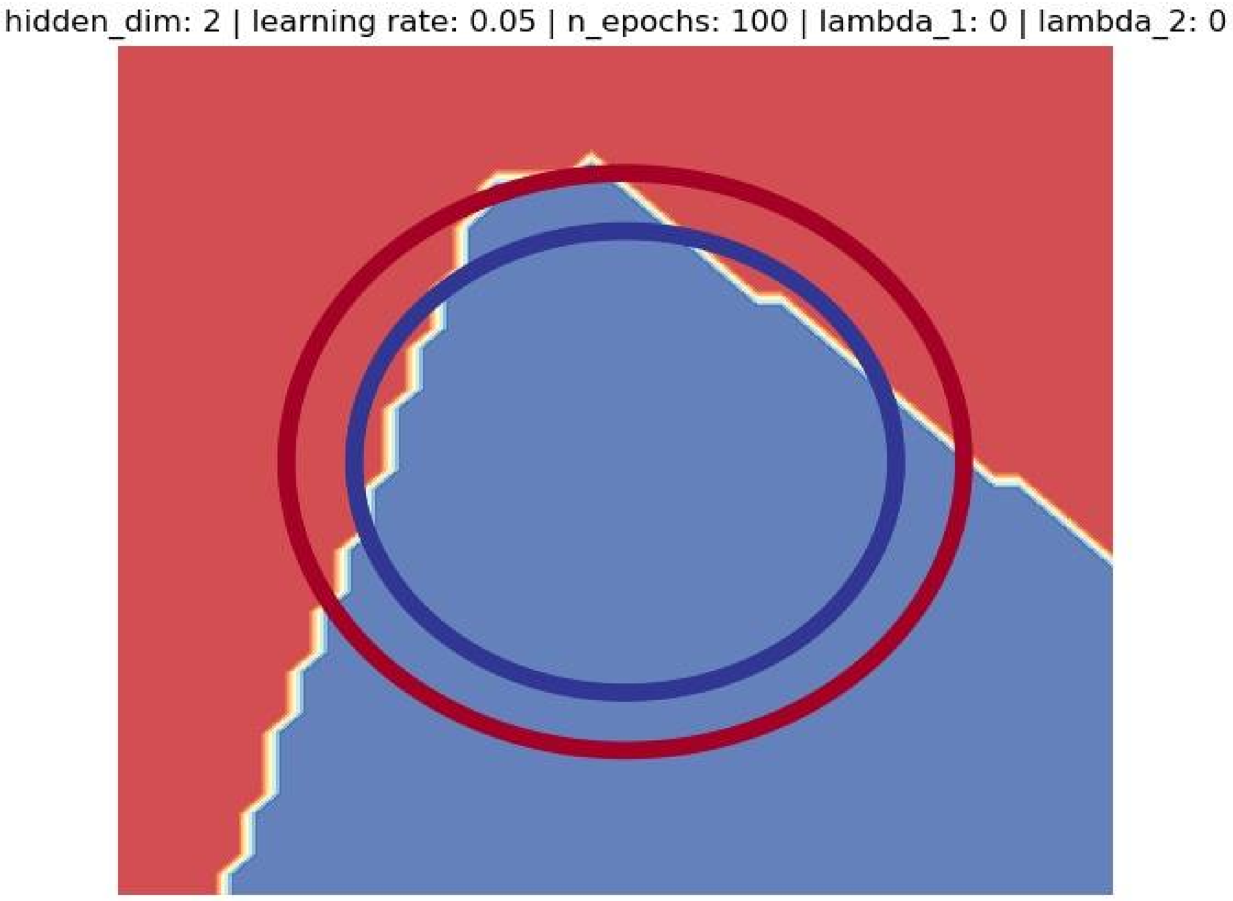

Figure 9: Decision boundary 5

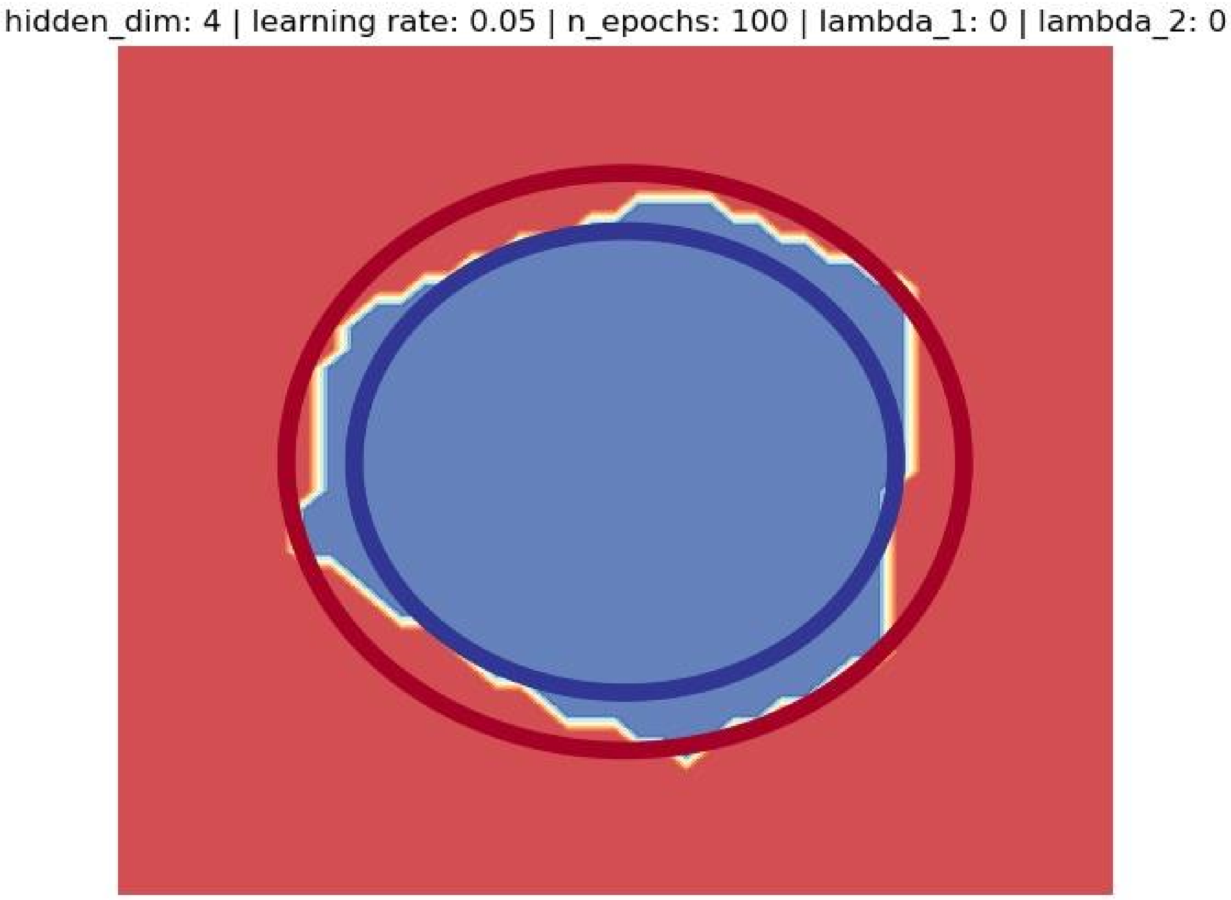

Figure 10: Decision boundary 6

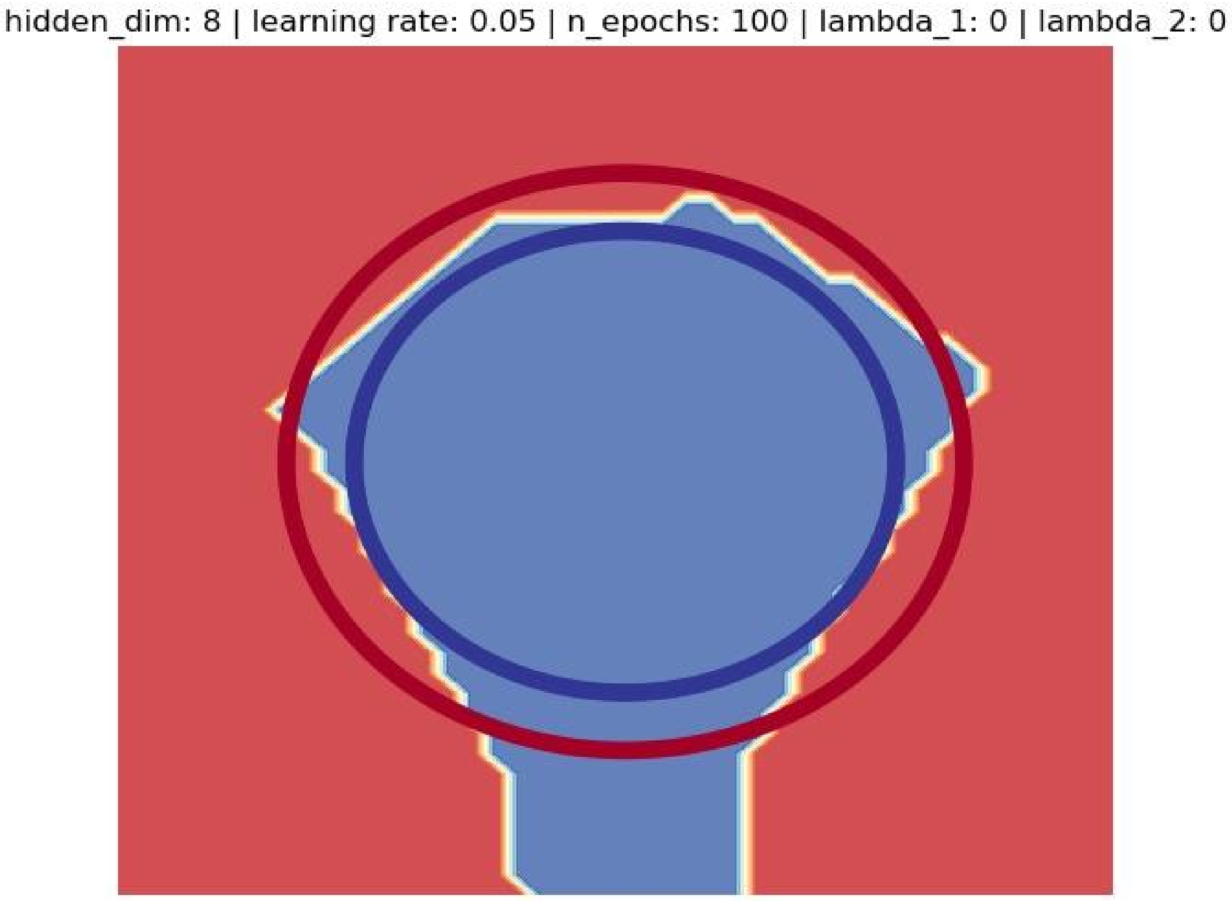

Figure 11: Decision boundary 7

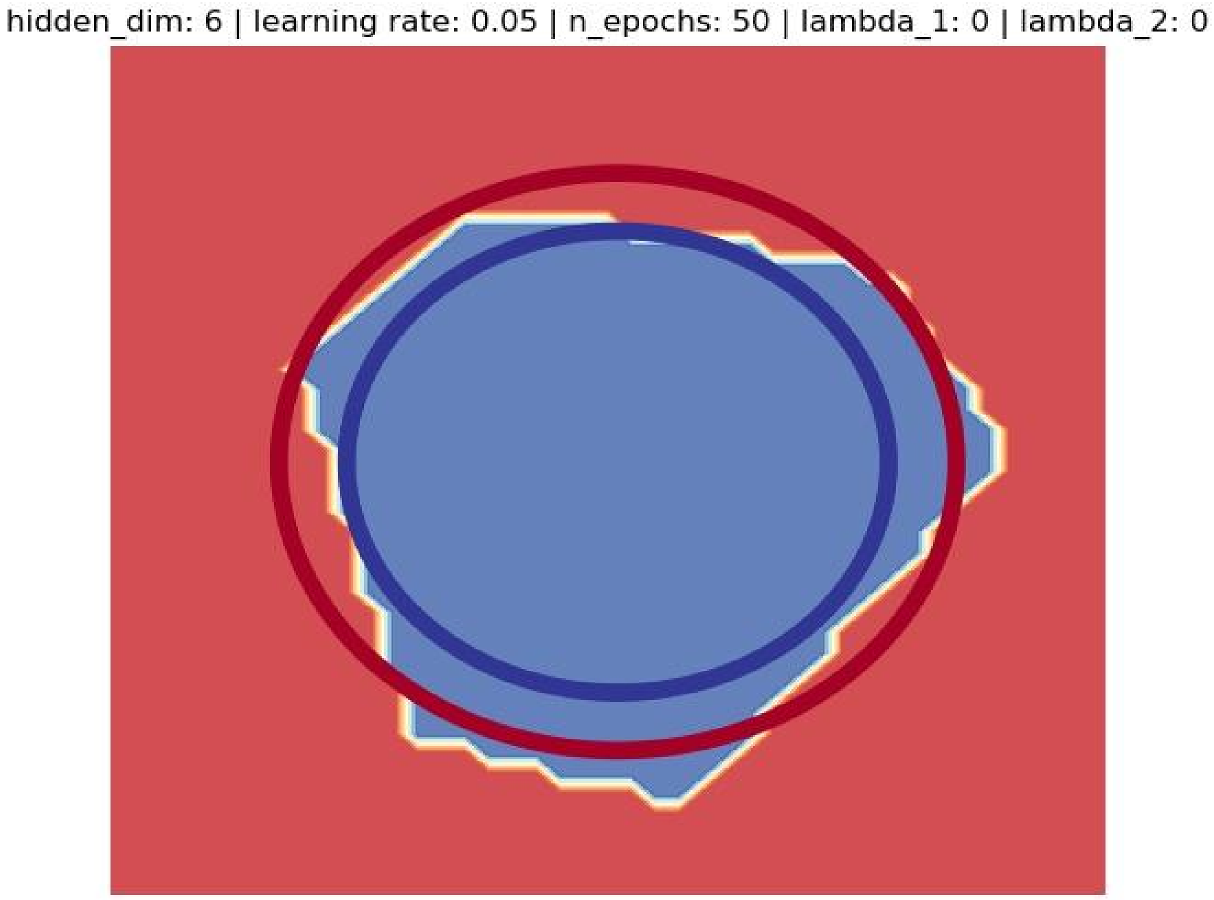

Figure 12: Decision boundary 8

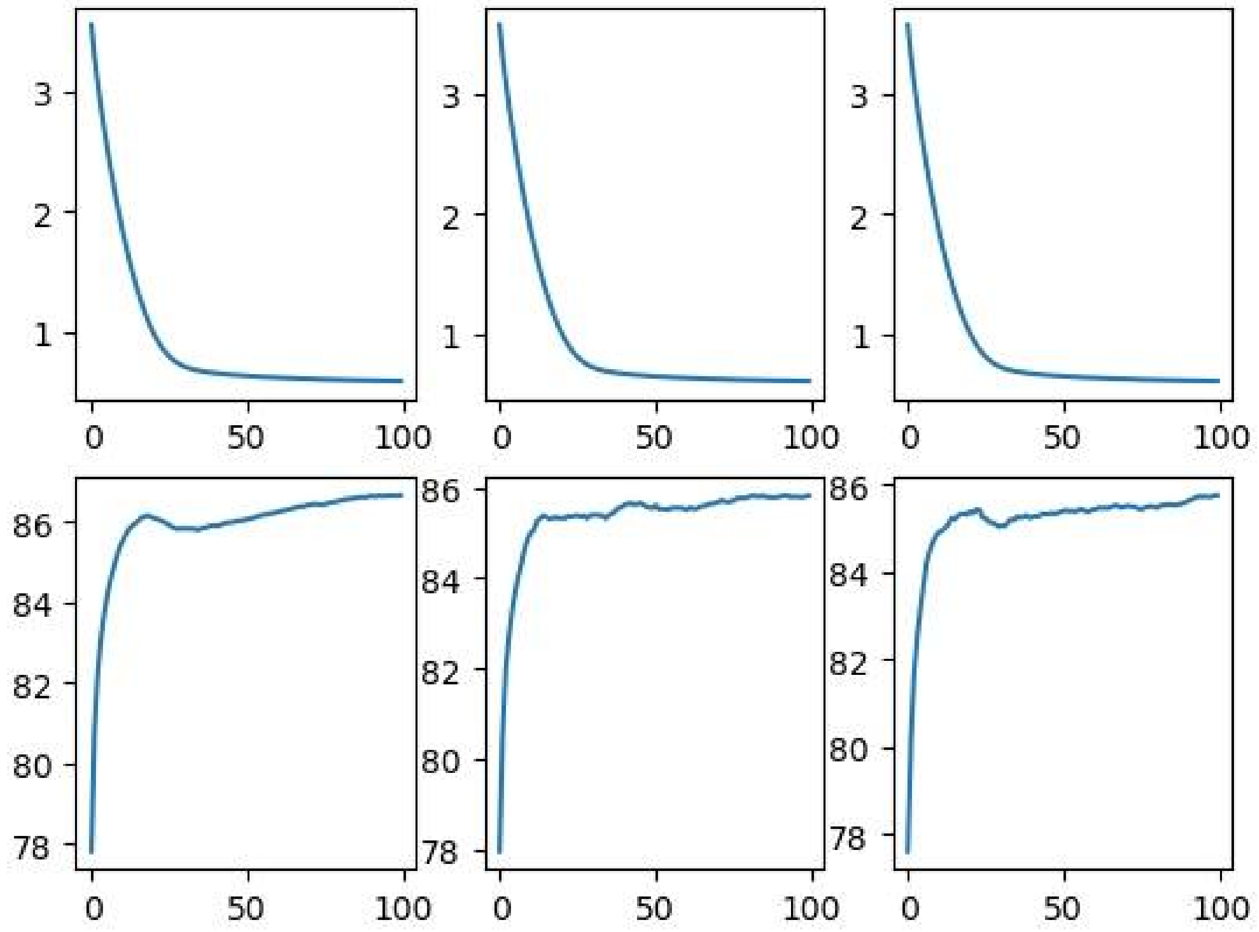

Figure 13: The first row is the loss function plots for each train/valid/test sets, and the second row is the accuracy for each train/valid/test sets. Hyperparamerers: Hidden layer size = 50, learning rate = 0.01, number of epochs = 100, all regularization constant set to 0.001.