Introduction to Machine Learning: Ex03

Theoretical Questions

1.

(15 points) SVM with multiple classes. One limitation of the standard SVM is that it can only handle binary classification. Here is one extension to handle multiple classes. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and now let $y_1, \ldots, y_n \in [K]$, where $[K] = \{1, 2, \ldots, K\}$. We will find a separate classifier $\mathbf{w}_j$ for each one of the classes $j \in [K]$, and we will focus on the case of no bias ($b = 0$). Define the following loss function (known as the *multiclass hinge-loss*):

$$\ell(\mathbf{w}_1, \ldots, \mathbf{w}_K, \mathbf{x}_i, y_i) = \max_{j \in [K]} (\mathbf{w}_j \cdot \mathbf{x}_i - \mathbf{w}_{y_i} \cdot \mathbf{x}_i + 1(j \neq y_i))$$

Define the following multiclass SVM problem:

$$f(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}_1, \ldots, \mathbf{w}_K, \mathbf{x}_i, y_i)$$

After learning all the $\mathbf{w}_j$, $j \in [K]$, classification of a new point $\mathbf{x}$ is done by $\arg\max_{j \in [K]} \mathbf{w}_j \cdot \mathbf{x}$. The rationale of the loss function is that we want the "score" of the true label, $\mathbf{w}_{y_i} \cdot \mathbf{x}_i$, to be larger by at least 1 than the "score" of each other label, $\mathbf{w}_j \cdot \mathbf{x}_i$. Therefore, we pay a loss if $\mathbf{w}_{y_i} \cdot \mathbf{x}_i - \mathbf{w}_j \cdot \mathbf{x}_i \leq 1$, for $j \neq y_i$.

Consider the case where the data is linearly separable. Namely, there exists $\mathbf{w}_1^*, \ldots, \mathbf{w}_K^*$ such that $y_i = argmax_y \mathbf{w}_y^* \cdot \mathbf{x}_i$. Show that any minimizer of $f(\mathbf{w}_1, \ldots, \mathbf{w}_K)$ will have zero classification error.

We want to show that any minimizer of $f(w_1, \ldots, w_k)$ will have zero classification error when the data is linearly separable.

$$f(w_1, \ldots, w_k) = \frac{1}{n} \sum_{i=1}^{n} \ell(w_1, \cdots w_k, x_i, y_i) = \frac{1}{n} \sum_{i=1}^{n} \max_{j \in [K]} \{\boldsymbol{w_j} \cdot \boldsymbol{x_i} - \boldsymbol{w_{y_i}} \cdot \boldsymbol{x_i} + 1_{j \neq y_i}\}$$

First we will show that the loss function is non negative:

$$(j = y_i) \rightarrow \ell(w_1, \cdots w_k, x_i, y_i) = \boldsymbol{w_j} \cdot \boldsymbol{x_i} - \boldsymbol{w_{y_i}} \cdot \boldsymbol{x_i} + 1 = 0 \rightarrow \max_{j \in [K]} \{\boldsymbol{w_j} \cdot \boldsymbol{x_i} - \boldsymbol{w_{y_i}} \cdot \boldsymbol{x_i} + 1(j \neq y_i)\} \geq 0$$

Since $f(\cdot)$ is a multiplication of a positive number $\left(\frac{1}{n}\right)$ with a sum of non-negative numbers we see that $f(\cdot) \geq 0$.

Now we will show that the minimizer $\omega = \omega_1, \cdots \omega_k$ of f results $f(\omega) \leq 0$:

We know that there are exists $w^* = w_1^1 \cdots w_k^*$ such that $\forall i.\, y_i = argmax_y(w_y^* x_i)$ so

$$\forall j \neq i.\, w_{y_i}^* x_i > w_j^* x_i \rightarrow (w_j^* - w_{y_i}^*) \cdot x_i < 0$$

For j = I we will define a new set of vectors $v_{i,j} = \begin{cases} -1 & i = j \\ (w_{y_i}^* - w_j^*) \cdot x_i & i \neq j \end{cases}$ and $\beta = \min_{i \neq j} \{v_{i,j}\}$ so:

$$\max_{j \in [K]} \{w_j^* \cdot x_i - w_{y_i}^* \cdot x_i + 1_{j \neq y_i}\} = \max_{j \in [K]} \{1 - v_{ji}\}$$

Defining $u_i^* = \frac{1}{\beta} w_i^*$ so $v_{i,j}^* = \begin{cases} (u_{y_i}^* - u_j^*) x_i & i \neq j \\ 1 & i = j \end{cases}$ so we can see that:

$$f(u_1^*, \cdots, u_k^*) = \frac{1}{n} \sum_{i=1}^{n} \max_{j \in [k]} \{1 - v_{ji}^*\}$$

So we achieved an input that makes $f(\,) = 0$ and since we already showed that $f \geq 0$ so if ω minimizes $f(\,)$ it results $f(\,) = 0$.

Let ω be a minimizer of f with non-zero empiric error. This means that $\exists i.\, j \neq y_i . \omega_j \cdot x_i > \omega_{y_i} \cdot x_i$.

This results that $\exists i.\, \ell(w_1, \cdots w_k, x_i, y_i) > 1$ which would result $f(\omega) \geq 1$, in contradiction to the fact that $f(\omega) = 0$.

2.

(10 points) Consider the soft-SVM problem with seperable data:

$$\min_{\mathbf{w},\xi} 0.5\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$s.t\ \forall i: \quad y_i\mathbf{w}\cdot\mathbf{x}_i \geq 1-\xi_i$$
$$\xi_i \geq 0$$

Let $\mathbf{w}^\star$ be the solution of **hard SVM**. Show that if $C \geq \|\mathbf{w}^\star\|^2$ then the solution of soft SVM separates the data. (Hint: Show that in the optimal solution $(\mathbf{w}', \boldsymbol{\xi}')$ of the soft SVM problem, the sum of ξ_i''s is bounded by some constant smaller than 1).

Since (w', ξ') is the optimal solution for the soft-SVM and $C \geq \|w\|^2$problem we get:

$$\frac{1}{2}\|w'\|^2 + C\sum_{i=1}^{n}\xi_i' \leq \frac{1}{2}\|w^*\|^2 \qquad \|w^*\|^2\sum_{i=1}^{n}\xi_i' \leq \frac{1}{2}\|w'\|^2 + C\sum_{i=1}^{n}\xi_i'$$

$$\|w^*\|^2\sum_{i=1}^{n}\xi_i' \leq \frac{1}{2}\|w^*\|^2 \rightarrow \sum_{i=1}^{n}\xi_i' \leq \frac{1}{2} < 1$$

Deciding that $\forall i.\, \xi_i' \leq \frac{1}{2}$ will preserve the equations so we get $y_i(w'\cdot x_i) > \frac{1}{2} \rightarrow \forall i.\, sign(w'x_i) = y_i$ so w separates the data

3.

(15 points) Separability using polynomial kernel. Let $x_1, ..., x_n \in \mathbb{R}$ be distinct real numbers, and let $q \geq n$ be an integer. Show that when using a polynomial kernel, $K(x, x') = (1 + xx')^q$, hard SVM achieves zero training error. Use the following fact: Given distinct values $\alpha_1, ..., \alpha_n$, the Vandermonde matrix defined by,

$$\begin{pmatrix} 1 & \alpha_1^1 & \dots & \alpha_1^q \\ 1 & \alpha_2^1 & \dots & \alpha_2^q \\ \vdots & & & \\ 1 & \alpha_n^1 & \dots & \alpha_n^q \end{pmatrix} \text{ is of rank } n$$

Lemma: let $\mathbf{X} = \begin{pmatrix} \mathbf{x_1^T} \\ \vdots \\ \mathbf{x_n^T} \end{pmatrix}$ be the fata matrix. If $\mathbf{rank(X) = n}$ then the data is linearly separable.

In Order to show that using $K(x, x')$ will result zero training error using hard SVM we will show that the data is linearly separable.

$$K(x, \tilde{x}) = (1 + xx')^q = \sum_{k=0}^{q} \binom{q}{m} x^m \tilde{x}^m = \sum_{k=0}^{q} \sqrt{\binom{q}{m}} x^m \sqrt{\binom{q}{m}} \tilde{x}^m = \phi(x)^T \phi(\tilde{x})$$

We will define the matrix $M_\phi = \begin{pmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \end{pmatrix}$ such that $M[i:]$ (the i^{th} row) and $M[:i]$ (the i^{th} column) satisfies:

$M_\phi[i:] = \phi(x_i)^T = \begin{pmatrix} \sqrt{\binom{q}{1}} x_i^0 & \cdots & \sqrt{\binom{q}{q}} x_i^q \end{pmatrix}$ and the i^{th}column satisfies $M_\phi[:i] = \begin{pmatrix} \sqrt{\binom{q}{i}} x_1^i \\ \vdots \\ \sqrt{\binom{q}{i}} x_n^i \end{pmatrix}$ therefore:

$$M_\phi = \begin{pmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \end{pmatrix} = \begin{pmatrix} 1 & \sqrt{\binom{q}{1}} x_1 & \dots & \sqrt{\binom{q}{q}} x_1^q \\ & & \vdots & \\ 1 & \sqrt{\binom{q}{1}} x_n & \dots & \sqrt{\binom{q}{q}} x_n^q \end{pmatrix}$$

We know that elementary operation on columns of a matrix doesn't affect the rank of the matrix.
So we will use the following elementary operations on the columns of M_ϕ:

$\forall c \in \{0,1,\cdots,n-1\}. M_\phi[:i] \leftarrow \widetilde{M_\phi}[:i] \cdot \binom{q}{c}^{-\frac{1}{2}}$ *so* $\widetilde{M_\phi}[:c] = \begin{pmatrix} x_1^i \\ \vdots \\ x_n^i \end{pmatrix}$ *meaning* $\widetilde{M_\phi} = \begin{pmatrix} \mathbf{1} & \mathbf{x_1^1} & \dots & \mathbf{x_1^q} \\ \mathbf{1} & \mathbf{x_2^1} & \dots & \mathbf{x_2^q} \\ \vdots & & & \\ \mathbf{1} & \mathbf{x_n^1} & \dots & \mathbf{x_n^q} \end{pmatrix}$

Since $x_1, \cdots, x_n$ *are distinct* $rank(M_\phi) = rank(\widetilde{M_\phi}) = n$

Using the quoted lemma from recitation 6 will lead us to conclude that $(x_1, \cdots, x_n)$ ***in linearly separable so SVM will have zero training error*** *(there is a linear separator for the n samples)*

4.

(10 points) Expressivity of ReLU networks. Consider the ReLU activation function:

$$h(x) = \max\{x, 0\}$$

In this exercise you will show how to implement the maximum function $f(x_1, x_2) = \max\{x_1, x_2\}$ using a neural network with one hidden layer and ReLU activations.

(a) Prove the following equalities:

- $x = \max\{x, 0\} - \max\{-x, 0\}$
- $|x| = \max\{x, 0\} + \max\{-x, 0\}$
- $\max\{x_1, x_2\} = \frac{x_1+x_2}{2} + \frac{|x_1-x_2|}{2}$

(b) Use (a) to show that it is possible to implement the maximum function using a ReLU network with one hidden layer which has 4 neurons, using ReLU activations (you can assume that there is no activation after the last layer). Explicitly write the parameters (weights) of the neural network you obtain.

a. We will use $s(x)$ as sign function that returns ± 1 according to sign

First we will prove $|x| = \max\{x, 0\} + \max\{-x, 0\}$:

If $x \geq 0$ we get that $\max\{x, 0\} = x$, $\max\{-x, 0\} = 0$ hence $|x| = x + 0 = x$ and since we assumed x is positive it holds.

If $x < 0$ we get that $\max\{x, 0\} = 0$, $\max\{-x, 0\} = -x$ hence $|x| = -x + 0 = -x$ and since we assumed x is negative it holds.

Therefore $\forall x \in \mathbb{R}. |x| = \max\{x, 0\} + \max\{-x, 0\}$

Using that we will prove that $x = \max\{x, 0\} - \max\{-x, 0\}$*:*

Since $x = s(x) \cdot |x|$ *we can see that* $x = s(x)(\max\{x, 0\} + \max\{-x, 0\})$ *so we can rewrite it as* $s(x)\max\{x, 0\} + s(x)\max\{-x, 0\}$ *and.*

for $x \geq 0$ *we get* $s(x) = 1, \max\{-x, 0\} = 0, \max\{x, 0\} = x$*:* $\forall x \geq 0. x = 1 \cdot x + 1 \cdot 0 = x$ *so* $x = x$*. For* $x < 0$ *we get* $s(x) = -1, \max\{-x, 0\} = -x, \max\{x, 0\} = 0$*:* $\forall x < 0. x = -1 \cdot 0 - 1 \cdot (-x) = x$ *so* $x = x$*.*

Therefore $\forall x \in \mathbb{R}. x = \max\{x, 0\} - \max\{-x, 0\}$

In order to show that $\max\{x_1, x_2\} = \frac{1}{2}(x_1 + x_2) + \frac{1}{2}|x_1 - x_2|$ *we will use the fact that* $\max\{x_1, x_2\} + x_3 = \max\{x_1 + x_3, x_2 + x_3\}$ *(Regardless of* $s(x_3)$*) and the identity about* $|x|$*.*

WLOG if $x_1 \geq x_2$ *so as* $x_1 + x_3 \geq x_2 + x_3$ *so* $\max\{x_1, x_2\} + x_2 = x_1 + x + 3 = \max\{x_1 + x_3, x_2 + x_3\}$

$\frac{1}{2}(x_1 + x_2 + |x_1 - x_2|) = \frac{1}{2}(x_1 + x_2 + \max\{x_1 - x_2, 0\} + \max\{x_2 - x_1, 0\}$ *so we'll apply the addition inside max() according to colors added:*

$$\frac{1}{2}(x_1 + x_2 + |x_1 - x_2|) = \frac{1}{2}(\max\{x_1 - x_2 + x_2, 0 + x_2\} + \max\{x_2 - x_1 + x_1, 0 + x_1\})$$
$$= \frac{1}{2}(\max\{x_1, x_2\} + \max\{x_2, x_1\}) = \frac{1}{2} \cdot 2\max\{x_1, x_2\} = \max\{x_1, x_2\}$$

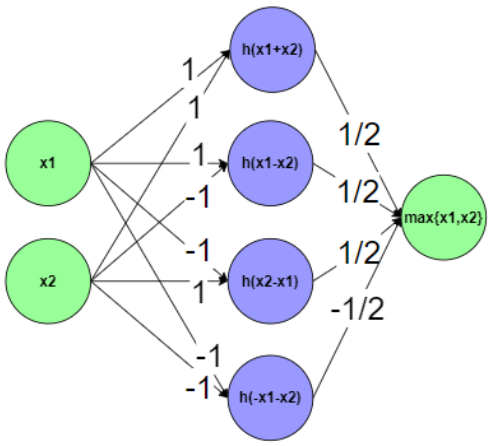
So $\forall x_1, x_2 \in \mathbb{R}. \max\{x_1, x_2\} = \frac{1}{2}(x_1 + x_2) + \frac{1}{2}|x_1 - x_2|$

b. In previous section we showed that we can implement $\max\{x_1, x_2\}$ based on $I(x) = x, abs(x) = |x|$ which can both be implemented using $ReLU(\cdot)$ function and basic operations $(+, -)$.

Let us rephrase the previous section result:$\max\{x_1, x_2\} = \frac{1}{2}(x_1 + x_2) + \frac{1}{2}|x_1 - x_2|$

$$\max\{x_1, x_2\} = \frac{1}{2}h(x_1 + x_2) - \frac{1}{2}h(-x_1 - x_2) + \frac{1}{2}h(x_1 - x_2) + \frac{1}{2}h(x_2 - x_1)$$

We see that $\max\{x_1, x_2\}$ is independent of $h\left(f\left(h(\cdot)\right)\right)$ and is linear function of $h\left(f(x)\right)$ for 4 different fs so we understand that it's possible to have a single hidden layer with 4 neurons: the weights of hidden neurons will be ± 1 and the weights of the last layer will be $\pm\frac{1}{2}$.



5.

(10 points) Implementing boolean functions using ReLU networks. Consider n boolean input variables, i.e. $x_1, \ldots, x_n \in \{0,1\}$. Your goal is to construct a neural network with ReLU activations, which implements the AND function:

$$f(x_1, \ldots, x_n) = x_1 \wedge x_2 \wedge \ldots \wedge x_n$$

Show how to construct a ReLU network with a constant number of layers which implements f. Describe the network's structure and parameters.

We will show a description of a 2 layers network.

In order to achieve a network with a constant layer count we will have to use De-Morgan rule:

$f(x_1, \cdots x_n) = x_1 \wedge x_2 \wedge \cdots \wedge x_{n-1} \wedge x_n = \overline{\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_{n-1}} \vee \overline{x_n}}$.

Since $\forall j \in \{1, \cdots, n\}. x_j \in \{0,1\}$ we can represent $\overline{x_j} = 1 - x_j$ which is linear function. Applying $\vee$ on all $\overline{x_j}$ will be achieved using sum on the layer.

Let us remember that when all inputs are 1 all the inverses are 0 so the sum is 0, in any other case the sum will be a non-zero positive integer .So what we actually want is to see the value of:

$$v_1 = \sum_{j=1}^{n}(1 - x_j) = \sum_{j=1}^{n}(1) + \sum_{j=1}^{n}(-x_j) = n + \sum_{j=1}^{n}(-1 \cdot x_j)$$

So we see **that for 1st layer all the weights of the inputs are -1 and we use $b = n$** s.t:

$$\forall x_j. x_j = 1 \rightarrow v_1 = n + \sum_{j} -x_j \rightarrow v_1 = 0 \quad , \quad \exists x_j. x_j \neq 1 \rightarrow v_1 = n + \sum_{i \neq j} -x_j \rightarrow v_1 > 0$$

Hence applying $ReLU(v_1)$ will result 0 when all inputs are positive and a positive non-zero integer otherwise.

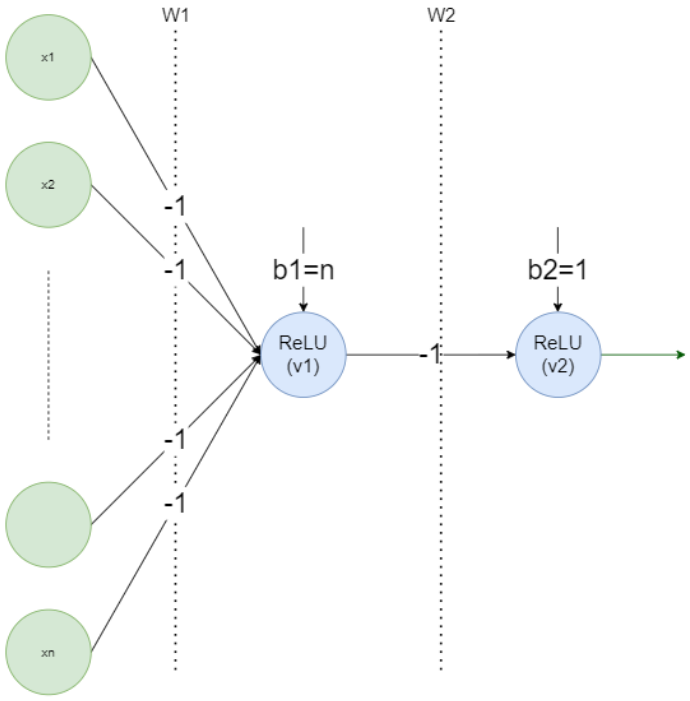
But this is not enough since we need the opposite result.

So the second layer wants to compute $v_2 = 1 - z_1$ so we use $b = 1, w = -1$ such that:

$$\begin{cases} v_2 = 1 & z_1 = 0 \\ v_2 < 0 & z_1 > 0 \end{cases} \rightarrow \begin{cases} \boldsymbol{v_2 = 1} & \boldsymbol{\forall j. x_j = 1} \\ \boldsymbol{v_2 < 0} & \boldsymbol{\exists x_j. x_j \neq 1} \end{cases}$$

and then we apply $z_2 = ReLU(v_2)$so:

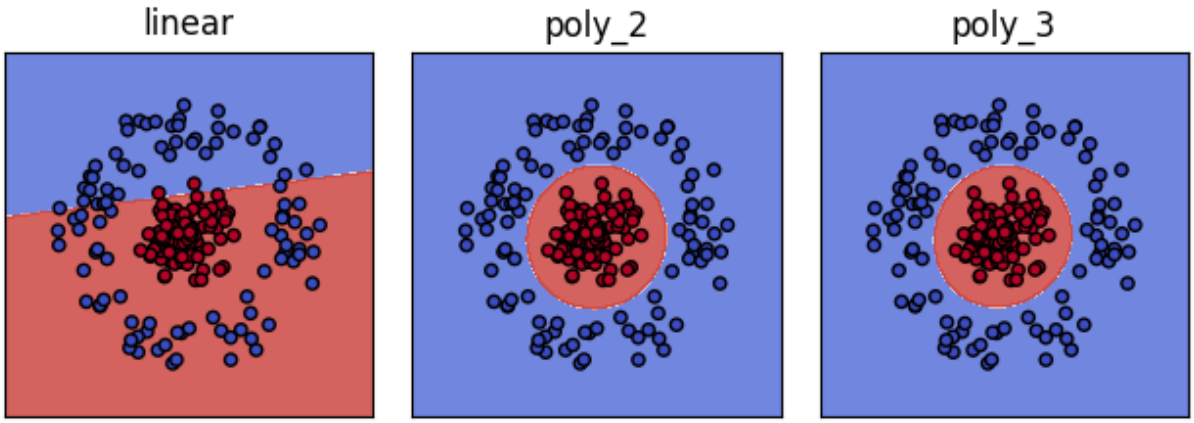
$$z_2 = \begin{cases} 1 & v_2 = 0 \\ 0 & v_2 < 0 \end{cases} = \begin{cases} 1 & \forall j. x\_j = 1 \\ 0 & \exists x_j. x_j \neq 1 \end{cases} \rightarrow \boldsymbol{z_2 = x_1 \wedge x_2 \wedge \cdots \wedge x_{n-1} \wedge x_n}$$



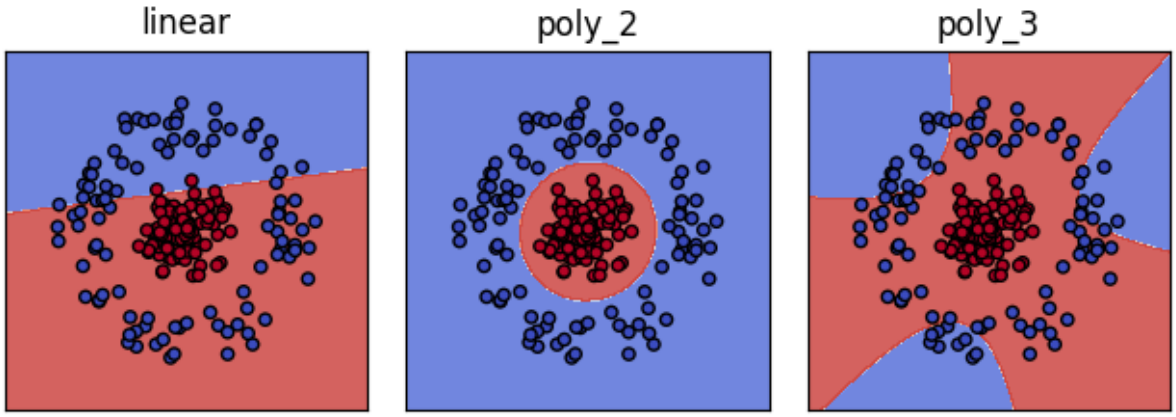
Programming Assignment

1.

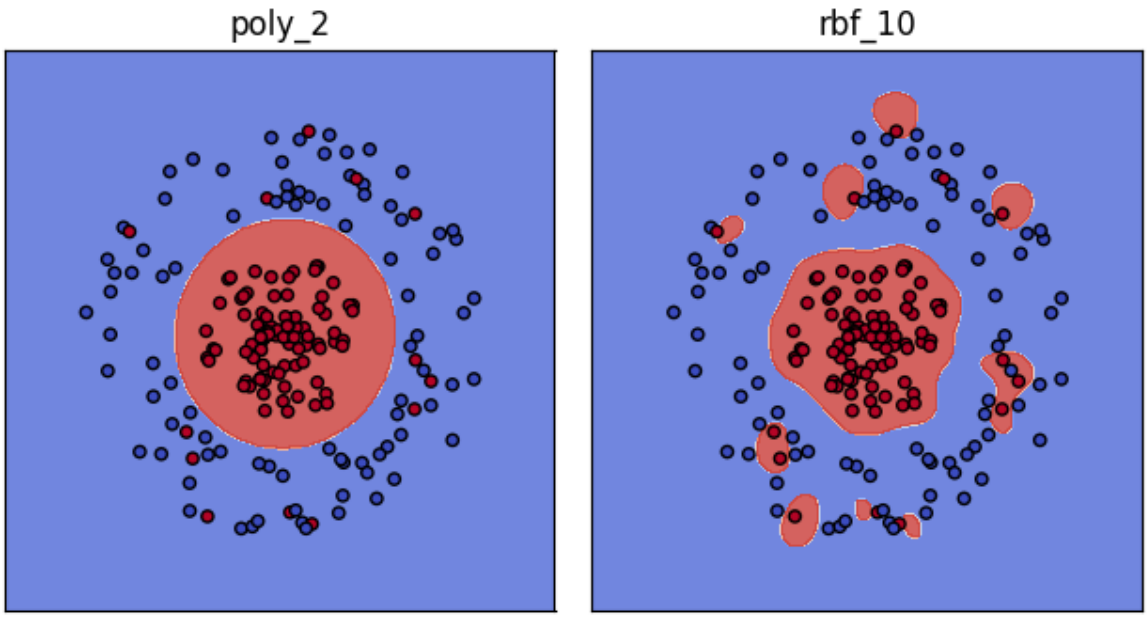
a. As expected, the linear kernel does not fit the data well ,since the data is not linearly separatable. The quadratic kernel (2nd degree polynomial) fits the data pretty good, since the data's distribution is actually quadratic ($x^2 + y^2 \leq R$). *Using 3rd degree polynomial results pretty good accuracy as well since it is homogenous.*



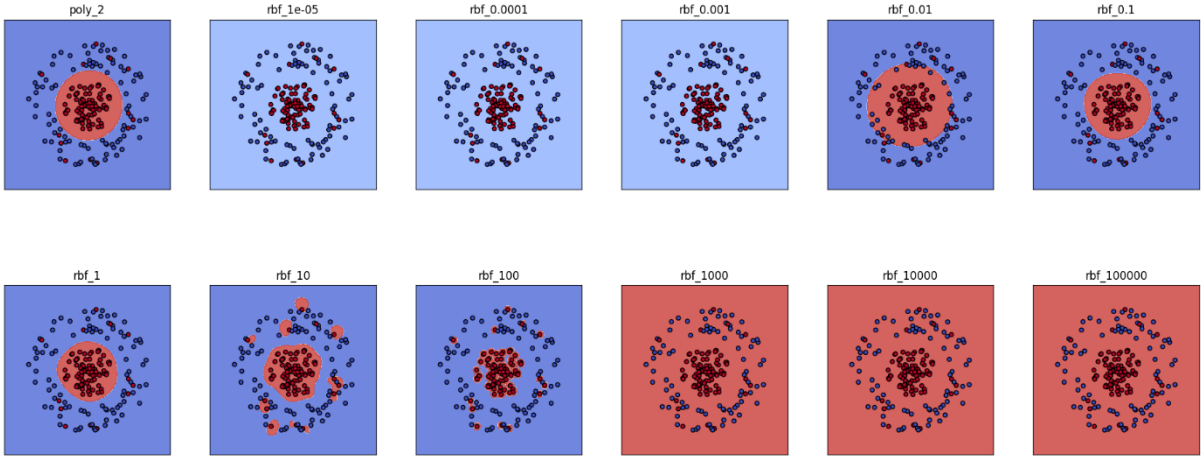
b. As expected, the linear kernel does not fit the data well ,since the data is not linearly separatable. The quadratic kernel (2nd degree polynomial) fits the data pretty good, since the data's distribution is actually quadratic ($x^2 + y^2 \leq R$). *Using non-homogenous 3rd degree polynomial results over-fitting to the training data so it doesn't fit well.*



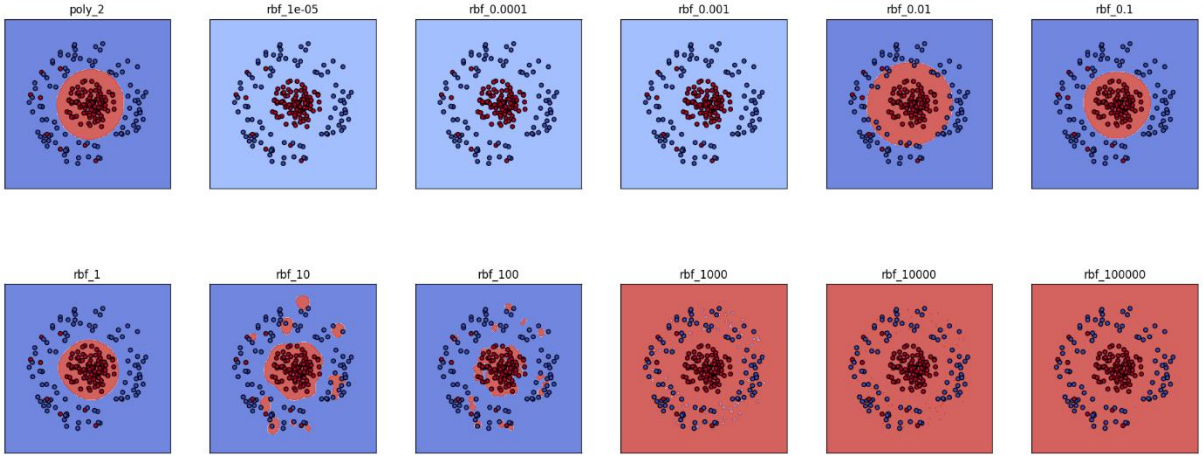
c.



The fitting to the training data:



The fitting with validation data:



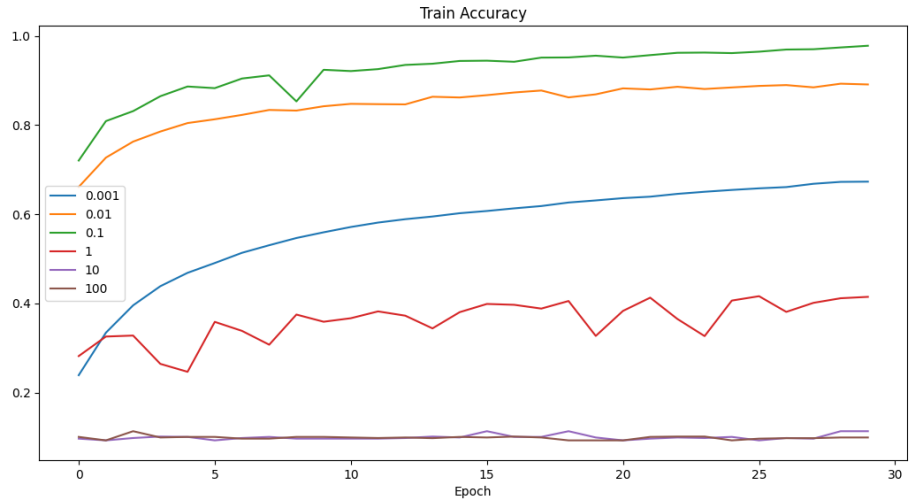
Achieved the following scores:

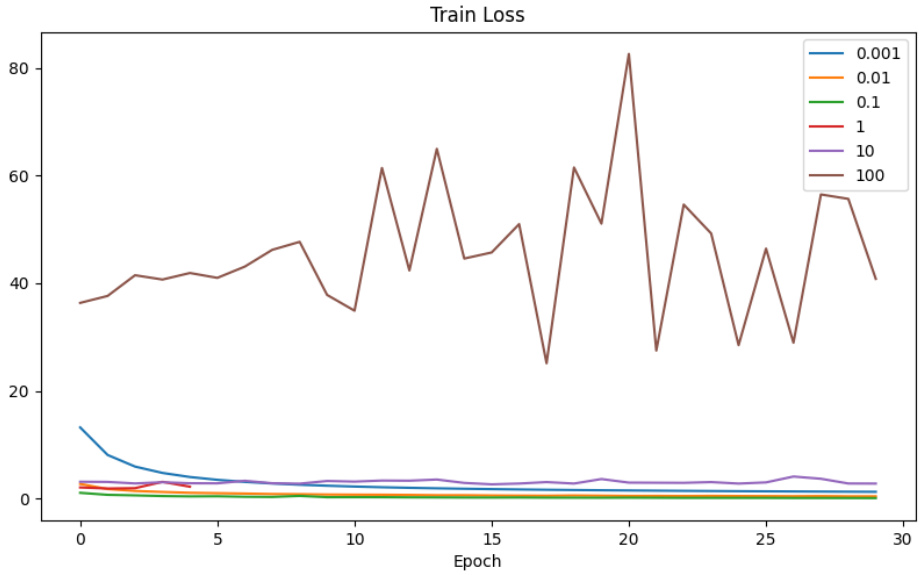
```
Training Scores
    SVC(C=10, coef0=0, degree=2, kernel='poly')          : 93.50%
    SVC(C=10, coef0=0, degree=0, gamma=1e-05)            : 56.50%
    SVC(C=10, coef0=0, degree=0, gamma=0.0001)           : 56.50%
    SVC(C=10, coef0=0, degree=0, gamma=0.001)            : 56.50%
    SVC(C=10, coef0=0, degree=0, gamma=0.01)             : 83.00%
    SVC(C=10, coef0=0, degree=0, gamma=0.1)              : 93.50%
    SVC(C=10, coef0=0, degree=0, gamma=1.0)              : 93.50%
    SVC(C=10, coef0=0, degree=0, gamma=10.0)             : 97.50%
    SVC(C=10, coef0=0, degree=0, gamma=100.0)            : 100.00%
    SVC(C=10, coef0=0, degree=0, gamma=1000.0)           : 100.00%
    SVC(C=10, coef0=0, degree=0, gamma=10000.0)          : 100.00%
    SVC(C=10, coef0=0, degree=0, gamma=100000.0)         : 100.00%
Validation Scores
    SVC(C=10, coef0=0, degree=2, kernel='poly')          : 96.00%
    SVC(C=10, coef0=0, degree=0, gamma=1e-05)            : 54.00%
    SVC(C=10, coef0=0, degree=0, gamma=0.0001)           : 54.00%
    SVC(C=10, coef0=0, degree=0, gamma=0.001)            : 54.00%
    SVC(C=10, coef0=0, degree=0, gamma=0.01)             : 87.50%
    SVC(C=10, coef0=0, degree=0, gamma=0.1)              : 96.00%
    SVC(C=10, coef0=0, degree=0, gamma=1.0)              : 96.00%
    SVC(C=10, coef0=0, degree=0, gamma=10.0)             : 91.50%
    SVC(C=10, coef0=0, degree=0, gamma=100.0)            : 84.50%
    SVC(C=10, coef0=0, degree=0, gamma=1000.0)           : 56.00%
    SVC(C=10, coef0=0, degree=0, gamma=10000.0)          : 54.00%
    SVC(C=10, coef0=0, degree=0, gamma=100000.0)         : 54.00%
```

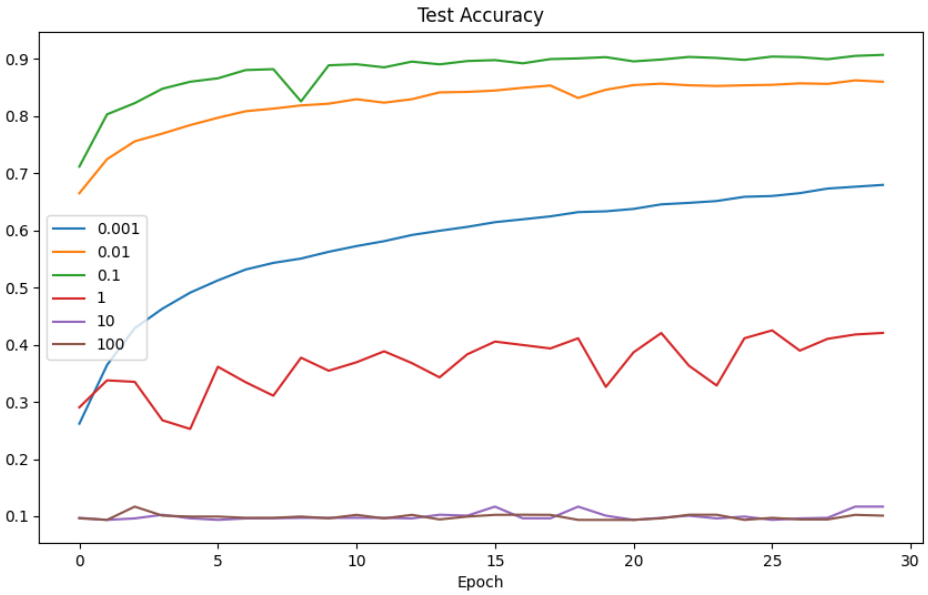
We can see that on the noisy examples RBF tends to overfit training data. When $\gamma \in (0.1\,, 1)$ we get the best results, but it is not better the poly2. On training data high γs *achieved 100% fitting due to overfitting.*

2.

b. When the Learning-Rate is too big the step we are doing in gradient decent is too big such that we pass over the point which minimizes loss, and we can't reach it (like playing golf an always hitting the ball too strongly so we always go to the other direction of the hole). When the Learning-Rate is too small do get closer and closer to the desired minimum-loss point but we are getting there slow, so we need to take more steps (so for a limited number of steps we will not reach desired accuracy)







c.

```
Intro to ML Ex4 2022 Q2: Neural Networks
    Running Section C:
        Learning Rate=0.1
            Initial Test Accuracy: 0.0703
            Epoch 0 Test Accuracy: 0.8626
            Epoch 1 Test Accuracy: 0.8857
            Epoch 2 Test Accuracy: 0.8965
            Epoch 3 Test Accuracy: 0.9099
            Epoch 4 Test Accuracy: 0.9148
            Epoch 5 Test Accuracy: 0.9228
            Epoch 6 Test Accuracy: 0.9298
            Epoch 7 Test Accuracy: 0.9295
            Epoch 8 Test Accuracy: 0.9152
            Epoch 9 Test Accuracy: 0.9366
            Epoch 10 Test Accuracy: 0.9381
            Epoch 11 Test Accuracy: 0.9359
            Epoch 12 Test Accuracy: 0.9323
            Epoch 13 Test Accuracy: 0.9418
            Epoch 14 Test Accuracy: 0.9453
            Epoch 15 Test Accuracy: 0.9456
            Epoch 16 Test Accuracy: 0.9404
            Epoch 17 Test Accuracy: 0.9444
            Epoch 18 Test Accuracy: 0.9452
            Epoch 19 Test Accuracy: 0.9451
            Epoch 20 Test Accuracy: 0.9449
            Epoch 21 Test Accuracy: 0.9458
            Epoch 22 Test Accuracy: 0.9437
            Epoch 23 Test Accuracy: 0.9485
            Epoch 24 Test Accuracy: 0.9484
            Epoch 25 Test Accuracy: 0.9438
            Epoch 26 Test Accuracy: 0.9468
            Epoch 27 Test Accuracy: 0.9419
            Epoch 28 Test Accuracy: 0.9492
            Epoch 29 Test Accuracy: 0.9453
```

d. In this section we tried to optimize Test Accuracy of the network
As a beginning we tested the influence of the mini-batch size (using the best-known learning rate: 0.1)
Later we used the best mini-batch size to scan various learning rate around 0.1:
Since I didn't get any improvement I launched a run that tested all the permutations but it was no good enough either, so I gave up.

```
Scanning batch_sizes=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
        learning_rate=0.1,mini_batch_size=1
            Final Test Accuracy: 0.6204
        learning_rate=0.1,mini_batch_size=2
            Final Test Accuracy: 0.8984
        learning_rate=0.1,mini_batch_size=3
            Final Test Accuracy: 0.9214
        learning_rate=0.1,mini_batch_size=4
            Final Test Accuracy: 0.918
        learning_rate=0.1,mini_batch_size=5
            Final Test Accuracy: 0.9076
        learning_rate=0.1,mini_batch_size=6
            Final Test Accuracy: 0.9048
        learning_rate=0.1,mini_batch_size=7
            Final Test Accuracy: 0.9104
        learning_rate=0.1,mini_batch_size=8
            Final Test Accuracy: 0.907
        learning_rate=0.1,mini_batch_size=9
            Final Test Accuracy: 0.9004
        learning_rate=0.1,mini_batch_size=10
            Final Test Accuracy: 0.9064
        learning_rate=0.1,mini_batch_size=11
            Final Test Accuracy: 0.8994
        learning_rate=0.1,mini_batch_size=12
            Final Test Accuracy: 0.9062
        learning_rate=0.1,mini_batch_size=13
            Final Test Accuracy: 0.9
        learning_rate=0.1,mini_batch_size=14
            Final Test Accuracy: 0.8856
        learning_rate=0.1,mini_batch_size=15
            Final Test Accuracy: 0.8986
        learning_rate=0.1,mini_batch_size=16
            Final Test Accuracy: 0.8984
        learning_rate=0.1,mini_batch_size=17
            Final Test Accuracy: 0.8952
        learning_rate=0.1,mini_batch_size=18
            Final Test Accuracy: 0.9004
        learning_rate=0.1,mini_batch_size=19
            Final Test Accuracy: 0.9016
        learning_rate=0.1,mini_batch_size=20
            Final Test Accuracy: 0.8806
        learning_rate=0.1,mini_batch_size=21
            Final Test Accuracy: 0.889
        learning_rate=0.1,mini_batch_size=22
            Final Test Accuracy: 0.8906
        learning_rate=0.1,mini_batch_size=23
            Final Test Accuracy: 0.8864
        learning_rate=0.1,mini_batch_size=24
            Final Test Accuracy: 0.894
```

```
Scanning rates[0.24 ,0.23 ,0.22 ,0.21 ,0.2 ,0.19 ,0.18 ,0.17 ,0.16 ,0.15 ,0.14 ,0.13 ,0.12 ,0.11 ,0.1 ,0.09 ,0.08 ,0.07 ,0.06 ,0.05 ,0.04 ,0.03 ,0.02 ,0.01]=
        learning_rate=0.01,mini_batch_size=3
            Final Test Accuracy: 0.886
        learning_rate=0.02,mini_batch_size=3
            Final Test Accuracy: 0.8932
        learning_rate=0.03,mini_batch_size=3
            Final Test Accuracy: 0.8982
        learning_rate=0.04,mini_batch_size=3
            Final Test Accuracy: 0.9092
        learning_rate=0.05,mini_batch_size=3
            Final Test Accuracy: 0.9024
        learning_rate=0.06,mini_batch_size=3
            Final Test Accuracy: 0.9104
        learning_rate=0.07,mini_batch_size=3
            Final Test Accuracy: 0.9114
        learning_rate=0.08,mini_batch_size=3
            Final Test Accuracy: 0.9136
        learning_rate=0.09,mini_batch_size=3
            Final Test Accuracy: 0.9168
        learning_rate=0.1,mini_batch_size=3
            Final Test Accuracy: 0.908
        learning_rate=0.11,mini_batch_size=3
            Final Test Accuracy: 0.9144
        learning_rate=0.12,mini_batch_size=3
            Final Test Accuracy: 0.9164
        learning_rate=0.13,mini_batch_size=3
            Final Test Accuracy: 0.9142
        learning_rate=0.14,mini_batch_size=3
            Final Test Accuracy: 0.9118
        learning_rate=0.15,mini_batch_size=3
            Final Test Accuracy: 0.9048
        learning_rate=0.16,mini_batch_size=3
            Final Test Accuracy: 0.9102
        learning_rate=0.17,mini_batch_size=3
            Final Test Accuracy: 0.9088
        learning_rate=0.18,mini_batch_size=3
            Final Test Accuracy: 0.8986
        learning_rate=0.19,mini_batch_size=3
            Final Test Accuracy: 0.8878
        learning_rate=0.2,mini_batch_size=3
            Final Test Accuracy: 0.877
        learning_rate=0.21,mini_batch_size=3
            Final Test Accuracy: 0.884
        learning_rate=0.22,mini_batch_size=3
            Final Test Accuracy: 0.8636
        learning_rate=0.23,mini_batch_size=3
            Final Test Accuracy: 0.845
        learning_rate=0.24,mini_batch_size=3
            Final Test Accuracy: 0.7548
Best Accuracy is 92.14% when self.batch_size=3, self.learning_rate=0
```