Intro to machine learning- solution 4-

Aviv Avraham

208295691

Introduction to Machine Learning	**Spring Semester**

Homework 4: May 9, 2023

Due: May 24, 2023

Theory 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*):

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

where $\mathbb{1}(\cdot)$ denotes the indicator function. 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 $y_i = argmax_y \mathbf{w}_y^* \cdot \mathbf{x}_i$ for all i. Show that any minimizer of $f(\mathbf{w}_1, \ldots, \mathbf{w}_K)$ will have zero classification error.

First, we'll show that $f(w_1, \ldots, w_k) \geq 0$ $for\ all\ w_i\ classifiers$.

By showing : $\ell(w_1, \ldots, w_k, x_i, y_i) = \max_{j \in [K]} (w_j \cdot x_i - w_{y_i} \cdot x_i + \mathbb{1}(j \neq y_i) \geq 0$

$for\ all\ i \in [n]$

Notice that for j =y_i we are getting $w_{y_i} \cdot x_i - w_{y_i} \cdot x_i + 0 = 0$

And therefore, the max value is greater than 0, for all i, as we wish.

Second, we'll show that there exists $w_1, \ldots, w_k$ those yields $f(w_1, \ldots, w_k) = 0$.

We know that the data is linearly separable by ${w_1}^*, \ldots, {w_k}^*$.

As mentioned, we pay a loss if $w_{y_i} \cdot x_i - w_j \cdot x_i < 1$, $for\ j \neq y_i$. If that isn't the case for ${w_1}^*, \ldots, {w_k}^*$than it reaches f value of 0, and were done. Otherwise, we know from separability that there exist at least one i that follows: $0 < w_{y_i} \cdot x_i - w_j \cdot x_i < 1, for\ j \neq y_i$. Let's define $\alpha = \min_i {w^*}_{y_i} \cdot x_i - {w^*}_j \cdot x_i\ for\ j \neq y_i$. As we notice $1 > \alpha > 0$.

And define: $w_j' = \frac{1}{\alpha} w^*_j$.

And now well notice that for all $j \neq y_i$:

$$w'_{y_i} \cdot x_i - w'_j \cdot x_i = by\ definition\ above = \frac{1}{\alpha} w^*_{y_i} \cdot x_i - \frac{1}{\alpha} w^*_j \cdot x_i = distributive$$
$$= \frac{1}{\alpha}(w^*_{y_i} \cdot x_i - w^*_j \cdot x_i)$$

And as we previously shown:

$$0 < \alpha \leq (w^*_{y_i} \cdot x_i - w^*_j \cdot x_i)$$

And therefore:

$$1 \leq \frac{1}{\alpha}\left(w^*_{y_i} \cdot x_i - w^*_j \cdot x_i\right)$$

And we aren't suffering loss.

As we have shown f is non negative, and there exist and argument that yields f to 0 value. Therefore, every minimizer of the f function yields 0 value for f.

Lemma: every minimizer of f (equivalent to any argument of f who's yields 0 value) has zero classification error.

Let $w_1, \dots, w_k$ be a minimizer for f. as we have already proof, $f(w_1, \dots, w_k) = 0$.

Let's assume in contradiction that $w_1, \dots, w_k$ has a classification error. In that case, there is exist i for which: $w_j \cdot x_i > w_{y_i} \cdot x_i$ for $j \neq y_i$. And therefore $w_j \cdot x_i - w_{y_i} \cdot x_i + \mathbb{I}(j \neq y_i) = w_j \cdot x_i - w_{y_i} \cdot x_i + 1 > 1$

$$from\ that\ we\ see\ that\ \ell(w_1, \dots, w_k, x_i, y_i) = \max_{j \in [K]}(w_j \cdot x_i - w_{y_i} \cdot x_i + \mathbb{I}(j \neq y_i) \geq 1$$

$$and\ \ f(w_1, \dots, w_k) = \frac{1}{n}\sum_{i=1}^{n} \ell(w_1, \dots, w_k, x_i, y_i) \geq \frac{1}{n} > 0 \quad \ell\ is\ non - negative\ as\ we\ shown$$

In contradiction to the fact that $f(w_1, \dots, w_k) = 0$.

3. **(15 points) Soft SVM with ℓ^2 penalty.** Consider the following problem:

$$\min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{n}\xi_i^2$$
$$\text{s.t. } y_i(\boldsymbol{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, n$$

(a) Show that a constraint of the form $\xi_i \geq 0$ will not change the problem. Meaning, show that these non-negativity constraints can be removed. That is, show that the optimal value of the objective will be the same whether or not these constraints are present.

(b) What is the Lagrangian of this problem?

(c) Minimize the Lagrangian with respect to $\boldsymbol{w}, b, \boldsymbol{\xi}$ by setting the derivative with respect to these variables to 0.

(d) What is the dual problem?

a

we will show that the optimal value of the objective will be the same whether these constraints are presents, by showing that in both cases the optimal value is the same.

For the case of this problem with the constraints we'll call case 1 and for the problem without the constraints we'll call case 2 for simplicity.

First, any solution of case 1 is a proper solution for case 2. Therefore, the optimal value of a solution for case 1 is a solution case 2. Therefore:

$$\textit{The optimal value solution in case } 1 \geq \textit{The optimal value solution in case } 2$$

Now we'll show that any solution for case 2 can be modified to a solution in case 1 with the same value. Let's assume that w, b, ξ is a solution for case 2.

Let's define: w, b, ξ' a solution for case 1 where $\xi'_i = |\xi_i|$.

The constraints are held because if $\xi_i \geq 0$ we didn't change it and otherwise $\xi_i < 0$:

$$y_i(w^T \cdot x_i + b) \geq 1 - \xi_i \geq 1 - |\xi_i| = 1 + \xi_i = 1 - \xi_i'$$

The value of the solution is the same because $\xi_i^2 = \xi_i'^2 = |\xi_i|^2$. Therefore:

$$\textit{The optimal value solution in case } 1 \leq \textit{The optimal value solution in case } 2$$

And we get that the optimal solutions are with the same value.

(b) What is the Lagrangian of this problem?

The Lagrangian of this problem is denoted by:

$$\mathcal{L}(w, b, \xi) = \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{n}\xi_i^2 + \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i(w^T \cdot x_i + b))$$

(c) Minimize the Lagrangian with respect to $\boldsymbol{w}, b, \boldsymbol{\xi}$ by setting the derivative with respect to these variables to 0.

$$\nabla_w \mathcal{L} = \frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$$

$$therefore\ (*): \qquad w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\nabla_b \mathcal{L} = \frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$therefore\ (**): \qquad 0 = \sum_{i=1}^{n} \alpha_i y_i$$

Notice that otherwise we can maximize Lagrangian value to infinity with b.

$$\nabla_{\xi_i} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \xi_i} = C\xi_i - \alpha_i = 0$$

$$therefore\ (***): \qquad \xi_i = \frac{1}{C}\alpha_i$$

Let's substitute those values in the Lagrangian, to get the minimum value:

$$\mathcal{L}\,(w, b, \xi) = \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{n} \xi_i^2 + \sum_{i=1}^{n} \alpha_i\big(1 - \xi_i - y_i(w^T \cdot x_i + b)\big) =$$

$$= \frac{1}{2} w \cdot w + \frac{C}{2}\sum_{i=1}^{n} \xi_i^2 + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i \xi_i - w \cdot \sum_{i=1}^{n} \alpha_i y_i x_i - \sum_{i=1}^{n} b\, \alpha_i y_i = *,**,***$$

$$= -\frac{1}{2}\sum_{i=1}^{n} \alpha_i y_i x_i \cdot \sum_{i=1}^{n} \alpha_i y_i x_i + \frac{C}{2}\sum_{i=1}^{n} \frac{1}{C}\alpha_i{}^2 + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \frac{1}{C}\alpha_i{}^2 =$$

$$-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i \cdot x_j - \frac{1}{2C}\sum_{i=1}^{n} \alpha_i{}^2 + \sum_{i=1}^{n} \alpha_i$$

The minimum is as follows because the Lagrangian is convex in respect to his variables and therefore, has a global minimum value(that can be calculate as we done by derivative).

(d) What is the dual problem?

As we know, the dual problem is given by:

$$g(\alpha) = \min_{w,b,\xi} \mathcal{L}\,(w, b, \xi, \alpha)$$

Using the fact that for a convex function:

$$\max_{\alpha} \min_{w,b,\xi} \mathcal{L}\,(w, b, \xi, \alpha) = \max_{\alpha} g(\alpha) =$$

$$\max_{\alpha} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i \cdot x_j - \frac{1}{2C}\sum_{i=1}^{n} {\alpha_i}^2 + \sum_{i=1}^{n} \alpha_i$$

$$s.t. \qquad 0 = \sum_{i=1}^{n} \alpha_i y_i$$

$$\alpha_i \geq 0 \; for \; all \; i \in [n]$$

4. **(15 points) Gradient of cross-entropy loss over softmax.** Let $\mathbf{y} \in \{0,1\}^d$ be a one-hot vector, i.e. $\mathbf{y}$ has a single entry which is 1 and the rest are 0. Consider the following loss function $\ell_{\mathbf{y}} : \mathbb{R}^d \to \mathbb{R}$:

$$\ell_{\mathbf{y}}(\mathbf{w}) = -\mathbf{y} \cdot \log\left(\text{softmax}(\mathbf{w})\right),$$

where $\text{softmax}(\mathbf{w}) = \frac{e^{\mathbf{w}}}{\sum_{j=1}^{d} e^{w_j}}$, is the softmax function (see slide 7 of lecture #7). In the above notation, the exponent and logarithm function operate elementwise when given a vector as an input. The function $\ell_{\mathbf{y}}$ is known as the *cross-entropy loss*, and you will encounter it in the programming assignment.

Prove that the gradient of $\ell_{\mathbf{y}}$ with respect to $\mathbf{w}$ is given by:

$$\nabla \ell_{\mathbf{y}}(\mathbf{w}) = \text{softmax}(\mathbf{w}) - \mathbf{y}$$

We know that y is a one-hot vector. And the loss function is denoted by:

$$\ell_y(w) = -y \cdot \log\big(softmax(w)\big), where\ softmax(w) = \frac{e^w}{\sum_{j=1}^{d} e^{w_j}}$$

We'll use that information to simplify this loss function, where we set $1 \le c \le d$ to be the one hot entry index of y:

$$\ell_y(w) = -y \cdot \log\big(softmax(w)\big) = \sum_{j=1}^{d} -y_j \log\big(softmax(w)_j\big) = -\log\left(softmax(w)_c\right)$$

Now, we can use the following chain rule: $\frac{\partial \ell_y}{\partial w} = \frac{\partial \ell_y}{\partial\, softmax(w)_c} \frac{\partial\, softmax(w)_c}{\partial w}$ *to calculate the gradient of* ℓ_y *in respect to w.*

$$\nabla \ell_y(w) = \frac{-1}{softmax(w)_c} \frac{\partial\, softmax(w)_c}{\partial w}$$

In order to calculate the second derivative, we'll look at two cases:

Case 1: in respect to the c entry:

$$\frac{\partial\, softmax(w)_c}{\partial w_c} = \frac{e^{w_c} \sum_{j=1}^{d} e^{w_j} - e^{w_c} e^{w_c}}{(\sum_{j=1}^{d} e^{w_j})^2} = \frac{e^{w_c}}{\sum_{j=1}^{d} e^{w_j}} \frac{\sum_{j=1}^{d} e^{w_j} - e^{w_c}}{\sum_{j=1}^{d} e^{w_j}} =$$

$$softmax(w)_c(1 - softmax(w)_c)$$

Case 2: in respect to the j ≠ c entry:

$$\frac{\partial\, softmax(w)_c}{\partial w_j} = \frac{-e^{w_c} e^{w_j}}{(\sum_{j=1}^{d} e^{w_j})^2} = -\frac{e^{w_c}}{\sum_{j=1}^{d} e^{w_j}} \frac{e^{w_j}}{\sum_{j=1}^{d} e^{w_j}} = -softmax(w)_c\, softmax(w)_j$$

Now we can put it all together in order to calculate $\nabla \ell_y(w)$.

$$\nabla \ell_y(w)_j = \frac{-1}{softmax(w)_c} \begin{cases} softmax(w)_c(1 - softmax(w)_c), if\ j = c \\ -softmax(w)_c\, softmax(w)_j, otherwise \end{cases} =$$

$$\begin{cases} softmax(w)_c - 1, if\ j = c \\ softmax(w)_j, otherwise \end{cases} = \begin{cases} softmax(w)_c - y_c, if\ j = c \\ softmax(w)_j - y_j, otherwise \end{cases} = softmax(w) - y$$

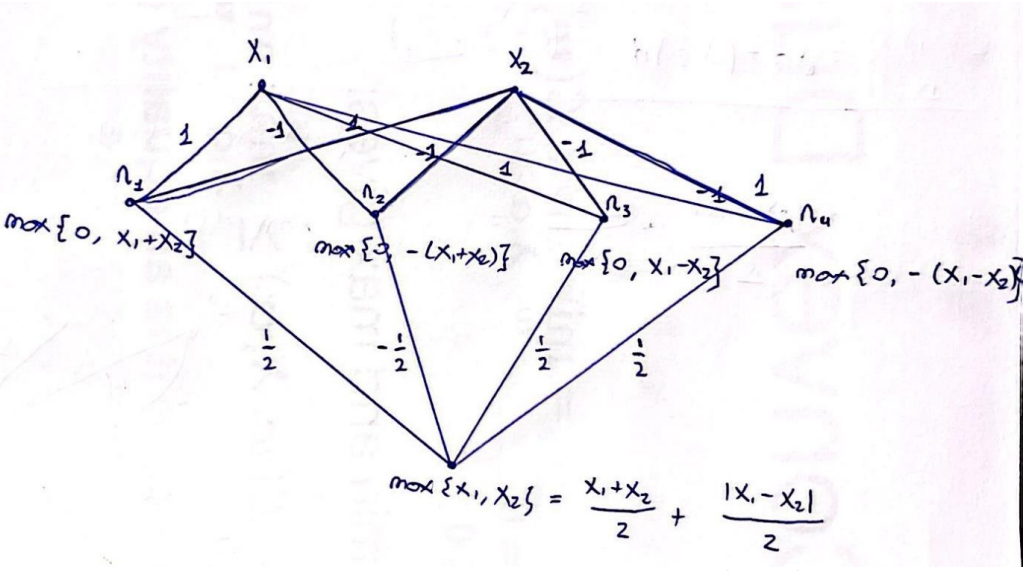
2. **(10 points) Expressivity of ReLU networks.** Consider the ReLU activation function:

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

Show that the maximum function $f(x_1, x_2) = \max\{x_1, x_2\}$ can be implemented using a neural network with one hidden layer and ReLU activations. You can assume that there is no activation after the last layer.

(Hint: It is possible to implement the function using a hidden layer with 4 neurons. You may find the following identity useful: $\max\{x_1, x_2\} = \frac{x_1+x_2}{2} + \frac{|x_1-x_2|}{2}$.)

implementation of $f(x_1, x_2) = \max\{x_1, x_2\}$ using a neural network with one hidden layer with 4 neurons and ReLU activations functions.



Using the hint that provided, we know that $\max\{x_1, x_2\} = \frac{x_1+x_2}{2} + \frac{|x_1-x_2|}{2}$

Notice that $\frac{1}{2}$n1= $\max\{0, x_1 + x_2\}$ + $-\frac{1}{2}$n2= $\max\{0, -(x_1 + x_2)\}$ *equals* $\frac{x_1+x_2}{2}$

And that $\frac{1}{2}$n3= $\max\{0, x_1 - x_2\}$ + $\frac{1}{2}$n4= $\max\{0, -(x_1 - x_2)\}$ *equals* $\frac{|x_1-x_2|}{2}$

Therefore, $\frac{1}{2}$n1 - $\frac{1}{2}$n2 + $\frac{1}{2}$n3 + $\frac{1}{2}$n4 equals $\frac{x_1+x_2}{2} + \frac{|x_1-x_2|}{2}$ that we know (hint) that is $\max\{x_1, x_2\}$ as requested.

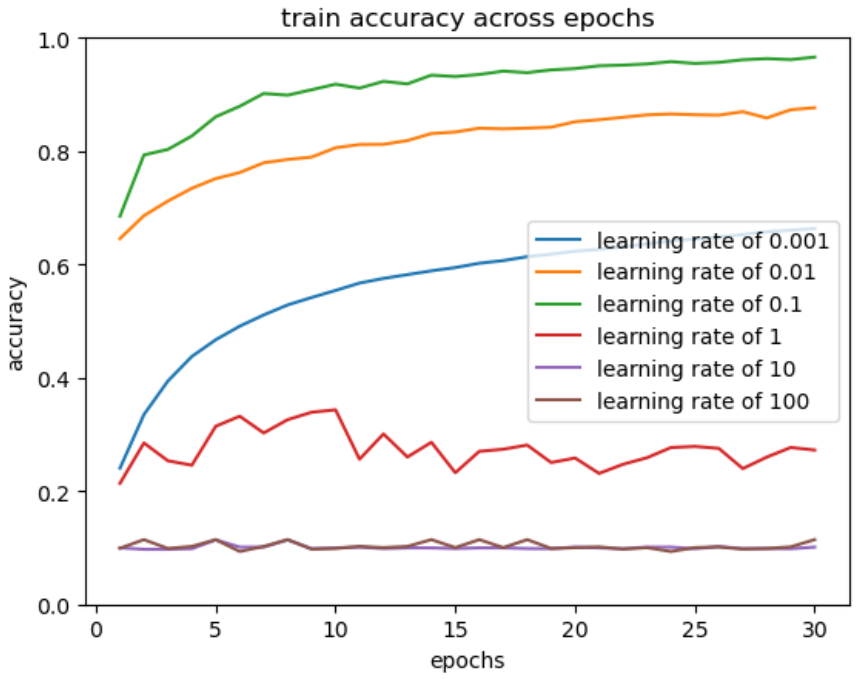
Programming assignment:

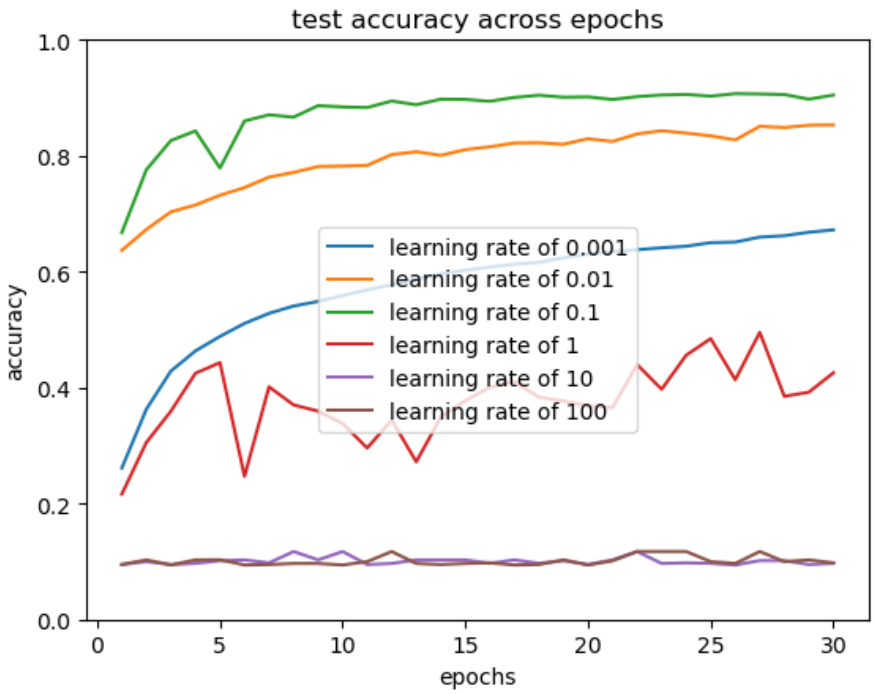
b

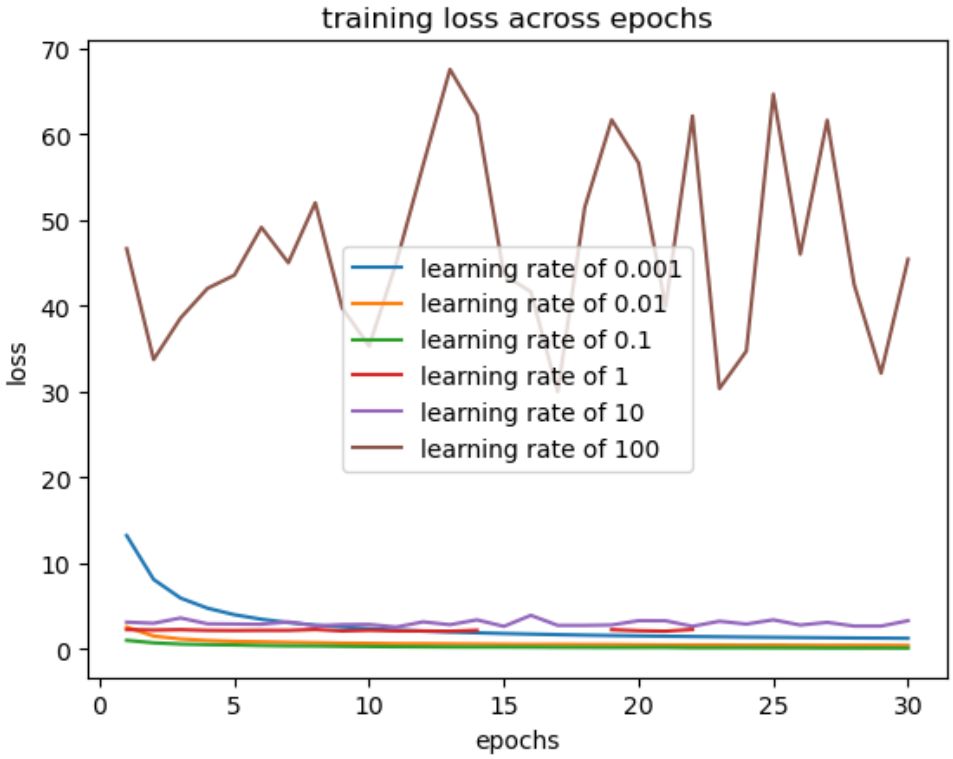
(b) **(10 points)** Train a one-hidden layer neural network as in the example given above (e.g., training set of size 10000, one hidden layer of size 40). For each learning rate in $\{0.001, 0.01, 0.1, 1, 10, 100\}$, plot the *training* accuracy, *training* loss ($\ell(\mathcal{W})$) and *test* accuracy across epochs (3 plots: each contains the curves for all learning rates). For the test accuracy you can use the *one_label_accuracy* function, for the training accuracy use the *one_hot_accuracy* function and for the training loss you can use the *loss* function. All functions are in the *Network* class.

The test accuracy with leaning rate 0.1 in the final epoch should be above 80%.

What happens when the learning rate is too small or too large? Explain the phenomenon.







When the learning rate is too high, {1, 10, 100} we miss the global min when we do our step towards it in the SGD algorithm. Therefore, were staying away from the minimum, and the train and test accuracies are poor. ~0.1 and the loss is high, especially in learning rate of 100, because the results are very different.

On the other hand, when the learning rate is too small, 0.001, the loss, test and train accuracies results are poor because the change is very tiny from each epoch to the other. But in different from the other case, by the end of the epochs, we get good result and small loss.

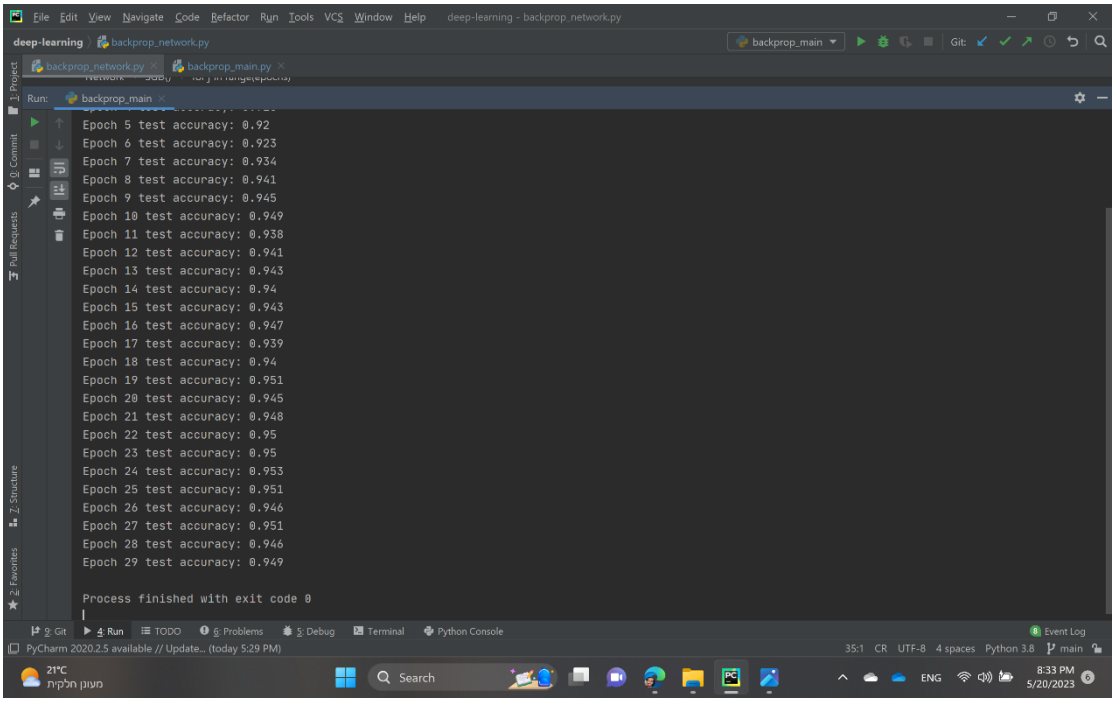
We can infer from that that it always right to take the small learning rate when we don't have any prior knowledge about the model.

(c) **(5 points)** Now train the network on the whole training set and test on the whole test set:

```
>>> training_data, test_data = data.load(train_size=50000,test_size=10000)
>>> net = network.Network([784, 40, 10])
>>> net.SGD(training_data, epochs=30, mini_batch_size=10, learning_rate=0.1,
test_data=test_data)
```

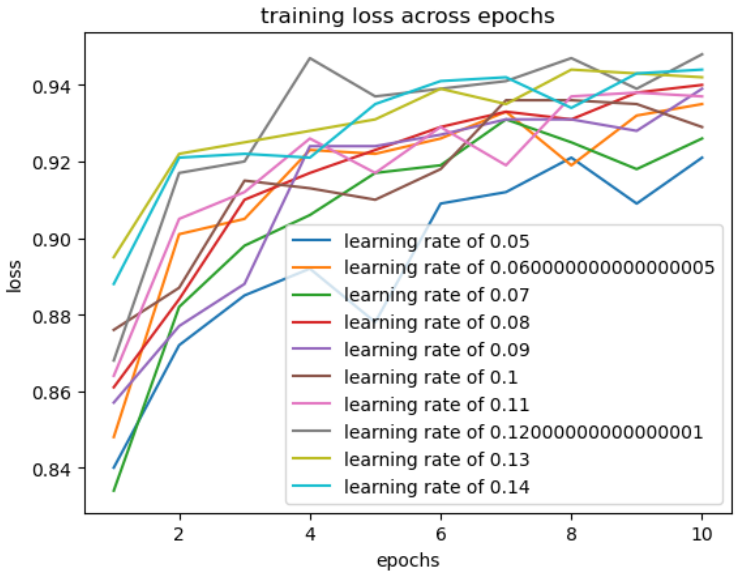
Do **not** calculate the training accuracy and training loss as in the previous section (this is time consuming). What is the test accuracy in the final epoch (should be above 90%)?

As we can tell the best test accuracy achieved with those parameters was 95.3 % test accuracy.

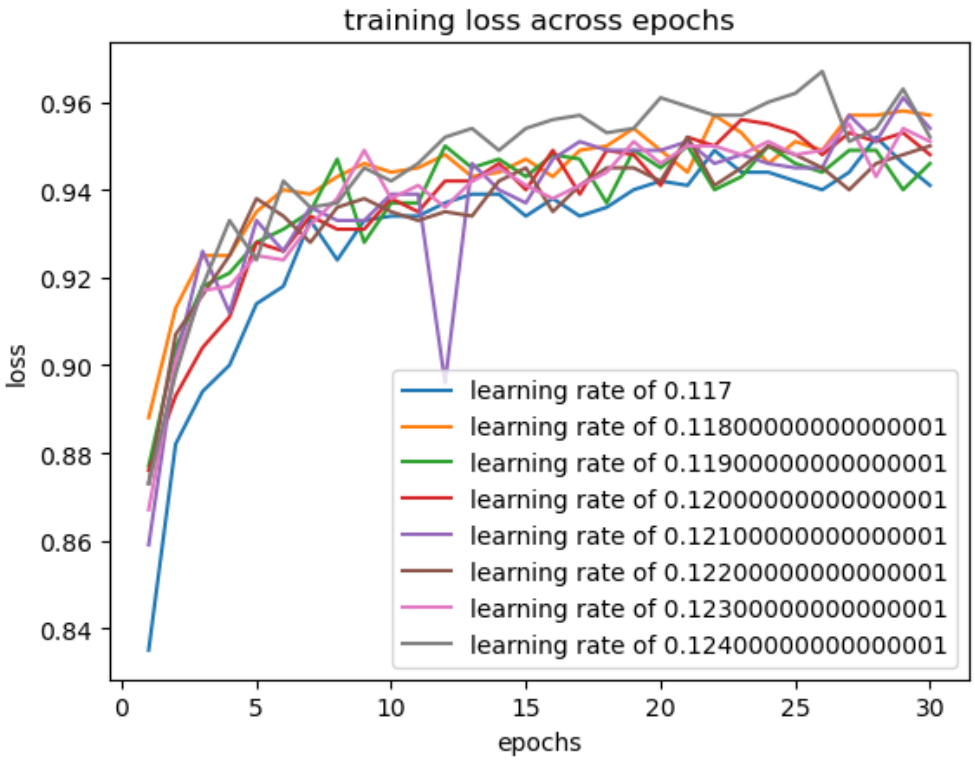


(d) **(10 points bonus)** Explore different structures and parameters and try to improve the test accuracy. Use the whole test set. In order to get full credit you should get accuracy of more than 96%. Any improvement over the previous clauses will give you 5 points. The maximum score for this homework set remains 100.

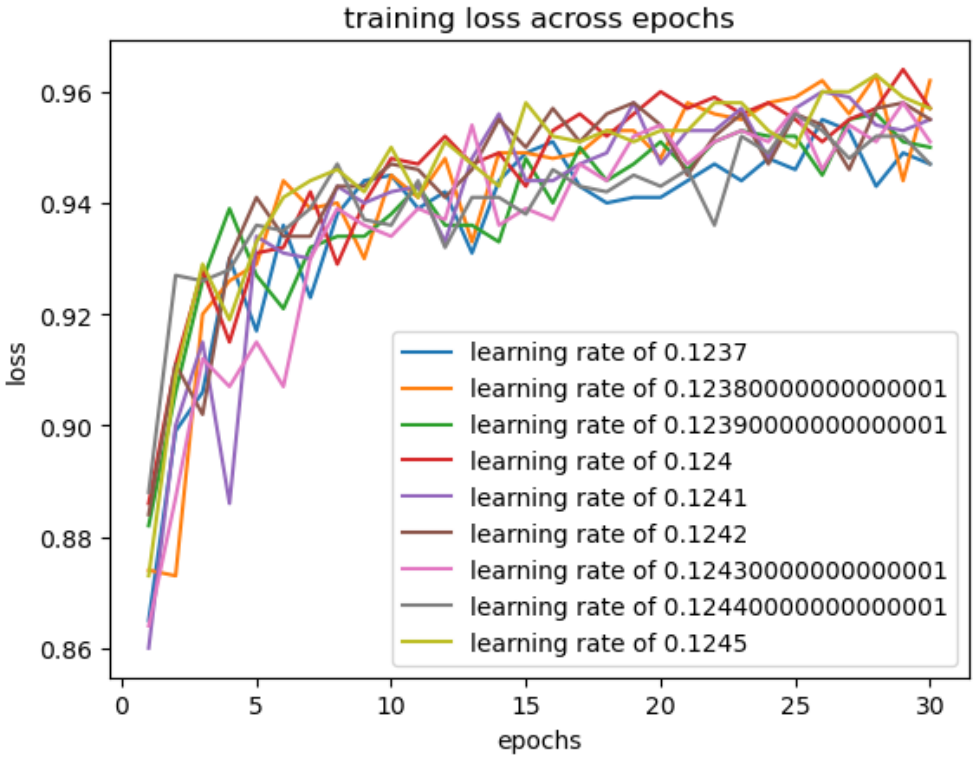
I examined some more learning rates around 0.1:



Then I extended my search around 0.12, which had pick results above 96%.



Then I extended my search around 0.124, which had pick results above 96.5%.



This results reflects that a 0.124 learning rate yields over 96% test accuracy.

(a) **(5 points)** Train the network **on 10% of the training set** (otherwise the training process would take a long time) once with the default parameters given in the code, and then again using the Adam optimizer instead of SGD (see the documentation in `torch.optim.Adam` for more information), with a learning rate of 0.00005. How is the training process affected by this change? Do you achieve better training loss? What about the validation accuracy?

The results of training the network on 10% of the training set with the default parameters are:

```
Epoch [10/10], Step [704/704], Loss: 0.5014
Accuracy of the network on the 5000 validation images: 53.52 %
```

The results of training the network on 10% of the training set with Adam algorithm instead SGD with learning rate of 0.00005 are:

```
Epoch [10/10], Step [704/704], Loss: 0.3353
Accuracy of the network on the 5000 validation images: 81.32 %
```

The training process seemed to improve, since the training loss achieved better results such as the validation accuracy.

(b) **(5 points)** Again using Adam, train the network with a learning rate of 0.005 instead of the default 0.00005. How is the training process affected?

```
Epoch [10/10], Step [704/704], Loss: 2.1716
Accuracy of the network on the 5000 validation images: 14.44 %
```

The learning process deteriorated, as we can see from the results, the loss has multiplied by a factor of 6, and the training accuracy had shrunk by a factor of 6.

(c) **(5 points)** Train the network using Adam and a step size of 0.00005, but change the architecture so that it doesn't include Batch Normalization or Dropout layers (simply comment out the relevant lines in the network's definition). How does the training loss evolve when compared to the one with BatchNorm + Dropout? Do these additions seem to accelerate the learning process?

```
Epoch [10/10], Step [704/704], Loss: 0.0678
Accuracy of the network on the 5000 validation images: 79.16 %
```

The training process seemed to improve it loss on the training set. since the training loss achieved better results such as the validation accuracy.

(d) **(5 points bonus)** Make whatever changes and tweaks in the network structure or other training parameters (except for adding more epochs - which would obviously improve the results) in order to obtain as good a test accuracy as you can get. **Train on the entire training set**. Take this opportunity to be creative, and try to learn more about common methods for training neural networks in practice. In order to get the 5 points bonus, you should get a test accuracy of at least 85%. Clearly state which changes or improvements you made in the training process.

Without any modifications we get:

```
Epoch [10/10], Step [782/782], Loss: 0.3063
Accuracy of the network on the 10000 test images: 81.07 %
```

then I added learning rate schedule instead of a fixed learning rate. The objective was to reduce the learning rate gradually over the course of training to help the model converge better. More specifically, I used scheduler = StepLR(optimizer, step_size=5, gamma=0.1).

Then got:

```
Epoch [10/10], Step [782/782], Loss: 0.3274
Accuracy of the network on the 10000 test images: 81.94 %
```

Clearly that isn't enough. So I enabled data augmentation during training by setting augment=True in the get_train_valid_loader function, plus other minor modifications. This can help the model generalize better by exposing it to a wider range of variations in the training data. Then got:

```
Epoch [10/10], Step [782/782], Loss: 0.5119
Accuracy of the network on the 10000 test images: 81.81 %
```

Then I increased the learning rate to 0.0001 and in the learning rate scheduler I increased the step size to 10 and gamma to 0.2. and Then got:

```
Epoch [10/10], Step [1563/1563], Loss: 0.5281
Accuracy of the network on the 10000 test images: 81.19 %
```