ECE 046211 - Technion - Deep Learning

HW2 - Multilayer NNs and Convolutional NNs ¶

Keyboard Shortcuts

- Run current cell: **Ctrl + Enter**
- Run current cell and move to the next: **Shift + Enter**
- Show lines in a code cell: **Esc + L**
- View function documentation: **Shift + Tab** inside the parenthesis or `help(name_of_module)`
- New cell below: **Esc + B**
- Delete cell: **Esc + D, D** (two D's)

Students Information

- Fill in

Name	Campus Email	ID
Student 1	student_1@campus.technion.ac.il	123456789
Student 2	student_2@campus.technion.ac.il	987654321

Submission Guidelines

- Maximal garde: 100.
- Submission only in **pairs**.
  - Please make sure you have registered your group in Moodle (there is a group creation component on the Moodle where you need to create your group and assign members).
- **No handwritten submissions.** You can choose whether to answer in a Markdown cell in this notebook or attach a PDF with your answers.
- SAVE THE NOTEBOOKS WITH THE OUTPUT, CODE CELLS THAT WERE NOT RUN WILL NOT GET ANY POINTS!
- What you have to submit:
  - If you have answered the questions in the notebook, you should submit this file only, with the name: `ece046211_hw2_id1_id2.ipynb` .
  - If you answered the questions in a different file you should submit a `.zip` file with the name `ece046211_hw2_id1_id2.zip` with content:
    - `ece046211_hw2_id1_id2.ipynb` - the code tasks
    - `ece046211_hw2_id1_id2.pdf` - answers to questions.
  - No other file-types (`.py` , `.docx` ...) will be accepted.
- Submission on the course website (Moodle).
- **Latex in Colab** - in some cases, Latex equations may no be rendered. To avoid this, make sure to not use *bullets* in your answers ("* some text here with Latex equations" -> "some text here with Latex equations").

Working Online and Locally

- You can choose your working environment:
  1. `Jupyter Notebook` , **locally** with Anaconda (https://www.anaconda.com/distribution/) or **online** on Google Colab (https://colab.research.google.com/)
     - Colab also supports running code on GPU, so if you don't have one, Colab is the way to go. To enable GPU on Colab, in the menu: `Runtime` → `Change Runtime Type` → `GPU`.
  2. Python IDE such as PyCharm (https://www.jetbrains.com/pycharm/) or Visual Studio Code (https://code.visualstudio.com/).
     - Both allow editing and running Jupyter Notebooks.
- Please refer to `Setting Up the Working Environment.pdf` on the Moodle or our GitHub (https://github.com/taldatech/ee046211-deep-learning (https://github.com/taldatech/ee046211-deep-learning)) to help you get everything installed.
- If you need any technical assistance, please go to our Piazza forum (`hw2` folder) and describe your problem (preferably with images).

Agenda

Part 1 - Theory

- You can choose whether to answser these straight in the notebook (Markdown + Latex) or use another editor (Word, LyX, Latex, Overleaf...) and submit an additional PDF file, **but no handwritten submissions**.
- You can attach additional figures (drawings, graphs,...) in a separate PDF file, just make sure to refer to them in your answers.
- $\LaTeX$ Cheat-Sheet (https://kapeli.com/cheat_sheets/LaTeX_Math_Symbols.docset/Contents/Resources/Documents/index) (to write equations)
  - Another Cheat-Sheet (http://tug.ctan.org/info/latex-refsheet/LaTeX_RefSheet.pdf)

Question 1 -Generalization in A Teacher-Student Setup

Recall from lecture 4 the Risk $\mathcal{R}(w)$:

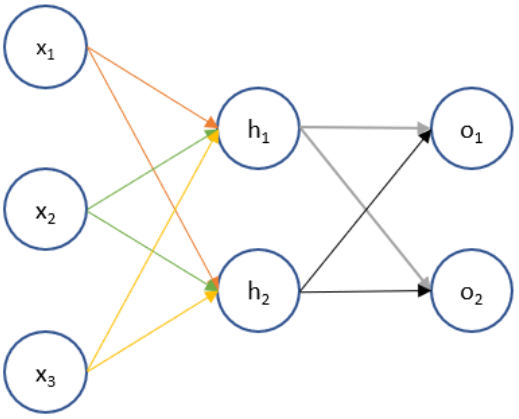
$$\mathcal{R}(w) \triangleq \mathbb{E}_{x^{(0)} \sim \mathcal{N}(0,I)} \left[||w^T x^{(0)} - w_t^T x^{(0)}||^2 \right]$$

Prove:

$$\mathcal{R}(w) = ||w - w_t||^2$$

Question 2 - Backpropagation By Hand

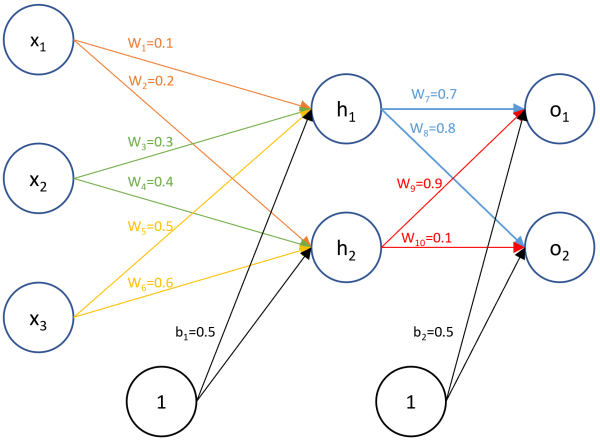
Consider the following network:



We will work with one sample for this example, but it can be extended to mini-batches.

- Input: $x = \begin{bmatrix} 1 \\ 4 \\ 5 \end{bmatrix} \in \mathbb{R}^3$
- Output (target): $t = \begin{bmatrix} 0.1 \\ 0.05 \end{bmatrix} \in \mathbb{R}^2$
- Number of Hidden Layers: 1
- Activation: Sigmoid for both hidden and output layers
- Loss Functions: MSE

We initialize the weights and biases to random values as follows:



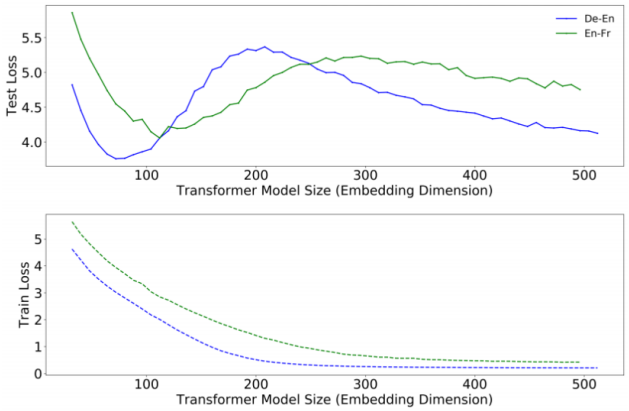
1. Perform one forward pass and calculate the MSE.
2. Perform backpropagation (one backward pass, i.e., calculate the gradients).
3. With a learning rate of $\alpha = 0.01$, what are the new values of the weights after performing the forward pass and backward pass (assume we use SGD)?

Question 3 - Deep Double Descent

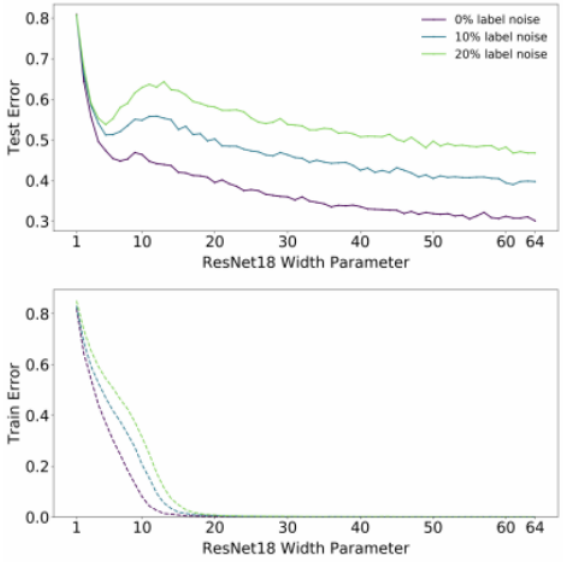
For the following plots:

1. Where is the critical point (the point of transition between the "Classical Regime" and "Modern Regime") of the deep double descent?
2. What type of double descent is shown (**look closely at the graph**)? Explain. There can be more than one correct answer.

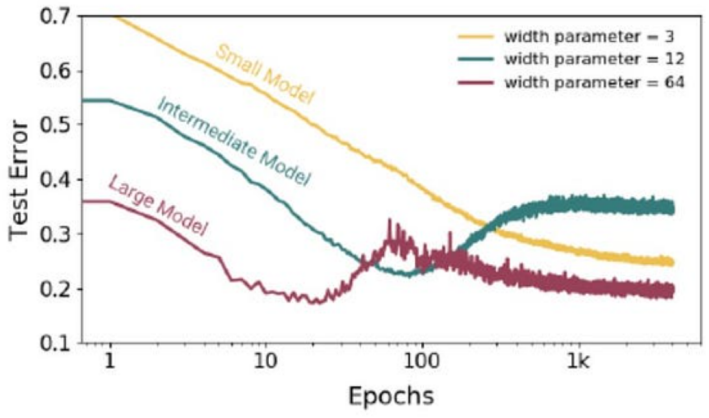
a.



b.



c.



Question 4 - Initialization

Recall that in lecture 5 we were discussing how to calculate the initialization variance, and reached the conclusion that

$$\sigma_l = \frac{1}{\sqrt{\sum_j \mathbb{E}\left[\varphi^2(u_{l-1}[j])\right]}}$$

Show that for ReLU activation ($\varphi(z) = max(0, z)$), the optimal variance satisfies:

$$\sigma_l = \sqrt{\frac{2}{d_{l-1}}}$$

1. Under the assumption that the distribution of W is symmetric ($\rightarrow$ the distribution of u is symmetric).
2. Using the central limit theorem for large width.

Answer each section **separately** and assume the sections are independent.

All the notations are the same as in the lecture slides.

Question 5 - MLP and Invaraince

You have to design an MLP with the following input: DNA sequences of length d. The DNA is a sequence of bases, where each base can be one of 4 options: (C, T, G, A). Thus, the input can be described as the following matrix:

$$X \in \mathcal{R}^{4\times d},$$

where $X[j, i]$ denotes the measured value of base concentration of the j^{th} base at location i.

The network should output a **binary** classification $y \in \{-1, 1\}$ for a specific property we wish to find. The network will be trained on samples $\{X^{(n)}, y^{(n)}\}_{n=1}^{N}$, with a **logistic loss function**.

First, we will examine a network with 1 hidden layerof size $4 \times d$ and a **LeakyReLU** activation ϕ:

$$f_w(X) = \sum_{r=1}^{4}\sum_{k=1}^{d} W_2[r, k]\phi\left(\sum_{j=1}^{4}\sum_{i=1}^{d} W_1[r, k, j, i]X[j, i]\right),$$

where $w = \{W_1, W_2\}$ are the layers of the weight **tensors**. After training is done, the classification will be done with $\text{sign}(f(X))$.

1. Which invariances exist in the network's parameters?
2. Now, we notice the fact that: the *direction* in which the DNA is scanned is arbitrary. Thus, if for two inputs $X, \tilde{X}$:
   $$\forall i, j: \; X[j, i] = \tilde{X}[j, d - i + 1],$$
   then the two inputs are **equivalent** in their meaning. What constraints should we put on the network's parameters to improve the network's classification performance? Explain why using an **invariant hidden layer** is not optimal.
3. After that, we now recall that the DNA bases come in pairs, and thus if for two inputs $X, \tilde{X}$:
   $$\forall i, j: \; X[j, i] = \tilde{X}[(4 - j)\text{mod}4 + 1, i] = \tilde{X}[5 - j, i],$$
   then the two inputs are **equivalent** in their meaning. What constraints should we put on the network's parameters to improve the network's classification performance?
4. We now notice that the measurement process in noisy, each sample $X^{(n)}$ is in arbitrary scale, and thus if for two $X, \tilde{X}$:
   $$\forall i, j: \; X[j, i] = c\tilde{X}[j, i],$$
   for some constant $c > 0$, then the two inputs are **equivalent** in their meaning.

   - (a) For the given network, that **is already trained**, what is the effect of the scale c on the classification result?
   - (b) Can the arbitrary scale hurt the training process? Hint: think what happens to the gradient of each sample.
   - (c) How can use this information to improve the classifier performance?

Question 6 -VGG Architecture

1. The VGG-11 CNN architecture consists of 11 convolution (CONV)/fully-connected (FC) layers (every CONV layer has the same padding and stride, every MAXPOOL layer is 2×2 and has padding of 0 and stride 2). Fill in the table. You need to **consider the bias**.

- CONVM-N: a convolutional layer of size $M \times M \times N$, where M is the kernel size and N is the number of filters. $stride = 1, padding = 1$.
- POOL2: 2×2 Max Pooling with $stride = 2$
  - In case the input of the layer is odd, you should round down. For example, if the output of the layer should be $3.5 \times 3.5 \times 3$, you should round to $3 \times 3 \times 3$ (i.e., ignore the last column of the input image when performing MaxPooling).
- FC-N: a fully connected layer with N neurons.

Layer	Output Dimension	Number of Parameters (Weights)
INPUT	224x224x3	0
CONV3-64	-	-
ReLU	-	-
POOL2	-	-
CONV3-128	-	-
ReLU	-	-
POOL2	-	-
CONV3-256	-	-
ReLU	-	-
CONV3-256	-	-
ReLU	-	-
POOL2	-	-
CONV3-512	-	-
ReLU	-	-
CONV3-512	-	-
ReLU	-	-
POOL2	-	-
CONV3-512	-	-
ReLU	-	-
CONV3-512	-	-
ReLU	-	-
POOL2	-	-
FC-4096	-	-
FC-4096	-	-
FC-1000	-	-
SOFTMAX	-	-

1. What is the total number of parameters? (use a calculator for this one)
2. What percentage of the weights are found in the fully-connected layers?

Part 2 - Code Assignments

- You must write your code in this notebook and save it with the output of all of the code cells.
- Additional text can be added in Markdown cells.
- You can use any other IDE you like (PyCharm, VSCode...) to write/debug your code, but for the submission you must copy it to this notebook, run the code and save the notebook with the output.

Tips

1. Uniformly distributed tensors - `torch.Tensor(dim1, dim2, ...,dimN).uniform_(-1, 1)`
2. Separation to **validation set** in PyTorch - See example here (https://gist.github.com/MattKleinsmith/5226a94bad5dd12ed0b871aed98cb123).

In [ ]:
```
# imports for the practice (you can add more if you need)
import os
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import torchvision
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# %matplotlib notebook
%matplotlib inline

seed = 211
np.random.seed(seed)
torch.manual_seed(seed)
```

Task 1 - The Importance of Activation and Initialization

In this task, we are going to use $x \in \mathcal{R}^{512}$ and simple neural network that outputs $f(x) \in \mathcal{R}^{512}$. The network will have 100 layers with 512 units in each layer.

1. We initialize the weights from a unit normal distribution. Run the following code cell and explain what happens. Add a short piece of code that locates when it happens (hint: use `torch.isnan()`). **Print** the layer number.
2. We can demonstrate that at a given layer, the matrix product of inputs x and weight matrix a that is initialized from a standard normal distribution will, on average, have a standard deviation very close to the square root of the number of input connections. For our example, with 512 dimensions, show that for 10,000 multiplications of a and x, the empirical standard deviation is similar to the square root of the number of input connections. Use the unbiased version:
$$\hat{std} = \sqrt{\frac{\sum_{i=1}^{10000} \frac{1}{N} \sum_{j=1}^{N} y^2}{10000}},$$
where $y = ax$ and N is the number of input connections. **Print** the mean, std and the square root of the number of input connections.
3. For the code from 1, normalize the weight initialization by the square root of the input connections. How does that change the outcome? **Print** the mean and std after the modification.
4. Add a `tanh()` activation after each layer for the code from 1. **Print** the mean and std after the modification. Explain the result.
5. Xavier initialization sets a layer's weights to values chosen from a random uniform distribution that's bounded between
$$\pm\sqrt{\frac{6}{n_i + n_{i+1}}}$$
where n_i is the number of incoming network connections, or "fan-in," to the layer, and n_{i+1} is the number of outgoing network connections from that layer, also known as the "fan-out". Glorot and Bengio believed that Xavier weight initialization would maintain the variance of activations and back-propagated gradients all the way up or down the layers of a network and demonstrated that networks initialized with Xavier achieved substantially quicker convergence and higher accuracy. Implement **Xavier Uniform** as `xavier_init(fan_in, fan_out)` , a function that returns a tensor initialized according to **Xavier Uniform**. Use it on the simple network from 1 with `tanh` activation. **Print** the mean and std after the modification.
6. If you try to replace the `tanh` activation with `relu` activation in section 5, you will see very different results. Xavier strives to acheive activation outputs of each layer to have a mean of 0 and a standard deviation around 1, on average. When using a ReLU activation, a single layer will, on average have standard deviation that's very close to the square root of the number of input connections, **divided by the square root of two** ($\sqrt{\frac{512}{2}}$ in our example). **Kaiming He et. al.** proposed an initialization scheme that's tailored for deep neural nets that use these kinds of asymmetric, non-linear activations. Implement **Kaiming Normal** as `kaiming_init(fan_in, fan_out)` , a function that returns a tensor initialized according to **Kaiming Normal** (use `fan_in` mode). Use it on the simple network from 1 with `relu` activation. **Print** the mean and std after the modification. What happens when you use Xavier with RelU activation?

In [ ]:
```
x = torch.randn(512)
for i in range(100):
    a = torch.randn(512, 512)
    x = a @ x
print(x.mean(), x.std())
```

Your answers here

In [ ]:
```
"""
Your Code Here - Use as many blocks as you need
"""
```

Task 2 - MLP-based Deep Classifer

In this task you are going to design and train your first neural network for classification.

For this task, we will use the "MAGIC Gamma Telescope Data Set" (https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope). Cherenkov gamma telescope observes high energy gamma rays, taking advantage of the radiation emitted by charged particles produced inside the electromagnetic showers initiated by the gammas, and developing in the atmosphere. This Cherenkov radiation (of visible to UV wavelengths) leaks through the atmosphere and gets recorded in the detector, allowing reconstruction of the shower parameters. The available information consists of pulses left by the incoming Cherenkov photons on the photomultiplier tubes, arranged in a plane, the camera.

Depending on the energy of the primary gamma, a total of few hundreds to some 10000 Cherenkov photons get collected, in patterns (called the shower image), allowing to discriminate statistically those caused by primary gammas (**signal**) from the images of hadronic showers initiated by cosmic rays in the upper atmosphere (**background**).

Our data has 10 features and 2 classes (signal and background).

1. Load the MAGIC dataset sored in `magic04.data` and display the first 5 features (just run the cell).
2. Separate the data to train, validation and test, reserve 10% of the data for validation and 20% for test.
3. Perform pre-processing steps of your choice and convert the class label from `str` to `int` (for example, `y_train = np.array([0 if y_train[i] == 'g' else 1 for i in range(len(y_train))]).astype(np.int)`).
4. Train a Logistic Regression model from `sklearn` as a baseline for our neural network (only for this section use both the train and validation sets for training the classifier). **Print the test accuracy**.
5. Convert the `numpy` arrays to `torch` tensors with `TensorDataset` as done in the tutorial.
6. Design a **MLP** to classify the data. Optimize the hyper-parameters of your model using the accuracy on the validation set, and when you are satisfied with the model train it on both the train and validation sets and evaluate it on the test set. **You need to reach at least 85% accuracy on the test set, and 87% for a full grade**.
    - You have a free choice of architecture, optimizer, learning scheduler, initialization, regularization and activations.
    - The loss criterion is binary cross entropy: `nn.BCEWithLogitsLoss()` (performs `sigmoid` for you) or `nn.BCELoss` (you need to apply `sigmoid` on the network output yourself).
    - In a Markdown block, write down the chosen architectures and all the hyper-parameters.
        - Make sure to describe any design choice that you used to improve the performance (e.g. if you used a certain regularization or layer, mention it and describe why you think it helped).
    - **Plot** the loss curves (and any oter statistic you want) as a function of epochs/iterations. **Print** the final performance.
    - **Print** the test accuracy.
7. Change the initialization of the linear layers and re-train the model (with the same optimal hyper-parameters you found). You can pick an initialization of your choosing from : https://pytorch.org/docs/stable/nn.init.html (https://pytorch.org/docs/stable/nn.init.html) . See example below how to use. **Print** the change in accuracy.

In [ ]:
```
# Loading the data
col_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',  'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
feature_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',  'fM3Long', 'fM3Trans', 'fAlpha', 'fDist']
data = pd.read_csv("./magic04.data", names=col_names)
X = data[feature_names]
Y = data['class']
data.head()
```

In [ ]:
```
# separate to train, test
"""
Your Code Here
"""
```

In [ ]:
```
# pre-processing and converting labels to integers
"""
Your Code Here
"""
```

In [ ]:
```
# training a Logistic Regression baseline - complete the code with your variables
logstic_model = LogisticRegression(solver='lbfgs')
y_pred = logstic_model.fit(X_train_prep, y_train_np).predict(X_train_prep)
print("Number of mislabeled points %d out of %d total points."% ((y_train_np != y_pred).sum(), X_train.sha
pe[0]))
print("Logistic Regression Model accuracy =" , logstic_model.score(X_test_prep, y_test_np))
```

In [ ]:
```
# create TensorDataset from numpy arrays
"""
Your Code Here
"""
```

In [ ]:
```
# model, hyoer-paramerters and training
"""
Your Code Here - add as many blocks as you wish
"""
```

In [ ]:
```
# example of weight initialization
import torch.nn as nn
class MyModel(nn.Module):
    def __init__(self, parmaeters):
        super(MyModel, self).__init__()
        # model definitions/blocks
        # ...
        # custom initialization
        self.init_weights()

    def init_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Linear):
                # pick initialzation: https://pytorch.org/docs/stable/nn.init.html
                # examples
                # nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                # nn.init.kaiming_normal_(m.weight, mode='fan_in', nonlinearity='leaky_relu', a=math.sqrt
(5))
                # nn.init.normal_(m.weight, 0, 0.005)
                # don't forget the bias term (m.bias)

    def forward(self, x):
        # ops on x
        # ...
        # output = f(x)
        return output
```

Task 3 - Design a CNN

In this task you are going to design a deep convolutional neural network to classify house number digits from the **The Street View House Numbers (SVHN)** Dataset.

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

- 10 classes, 1 for each digit. Digit '0' has label 0, '1' has label 1,...
- 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data.

1. Load the SVHN dataset with PyTorch using `torchvision.datasets.SVHN(root, split='train', transform=None, target_transform=None, download=True)` , you can read more here: https://pytorch.org/vision/stable/generated/torchvision.datasets.SVHN.html#torchvision.datasets.SVHN (https://pytorch.org/vision/stable/generated/torchvision.datasets.SVHN.html#torchvision.datasets.SVHN). Display 5 images from the train set.
2. Design a Convolutional Neural Network (CNN) to classify digits from the images.
   - Describe the chosen architecture, how many layers? What activations did you choose? What are the filter sizes? Did you use fully-connected layers (if you did, explain their sizes)?
   - What is the input dimension? What is the output dimension?
   - Calculate the number of parameters (weights) in the network. What is the model size in Megabytes? (see the convolution tutorial). **Print** these numbers.
3. Train the classifier (preferably on a GPU - use Colab for this part if you don't have a GPU).
   - Describe the the hyper-parameters of the model (batch size, epochs, learning rate....). How did you tune your model? Did you use a validation set to tune the model?
   - What is the final accuracy on the test set? **Print** it.
     - You need to reach at least 86% accuracy in this section, and 90% for a full grade.
   - **Plot** the loss curves (and any other statistic you calculate) as a function of epochs/iterations.
4. For the trained classifier, what is the accuracy on the test set when each test image is added a small noise $a = (0.05, 0.01, 0.005)$:
$$\text{image} + a \times \mathcal{N}(0, 1)$$
. **Print** the result for each value of a.
5. Retrain the classifier, but this time use data augementation of your choosing. Briefly explain what augmentation you chose and how it works. Did the test accuracy improve? **Print** the result.

   - You can use transformations available in `torchvision.transforms` as shown in the tutorial.
   - You are welcome to use `kornia` (https://kornia.github.io/) for the augmentations (**2 points bonus**, maximal grade is still 100).
   - **Plot** the loss curves (and any other statistic you want) as a function of epochs/iterations.

In [ ]:
```
"""
Your Code Here
"""
```

Credits

- Icons made by Becris (https://www.flaticon.com/authors/becris) from www.flaticon.com (https://www.flaticon.com/)
- Icons from Icons8.com (https://icons8.com/) - https://icons8.com (https://icons8.com)
- Datasets from Kaggle (https://www.kaggle.com/) - https://www.kaggle.com/ (https://www.kaggle.com/)