EE 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: `ee046211_hw2_id1_id2.ipynb` .
  - If you answered the questionss in a different file you should submit a `.zip` file with the name `ee046211_hw2_id1_id2.zip` with content:
    - `ee046211_hw2_id1_id2.ipynb` - the code tasks
    - `ee046211_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/)](https://www.anaconda.com/distribution/) or **online** on [Google Colab (https://colab.research.google.com/)](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` $\rightarrow$ `Change Runtime Type` $\rightarrow$ `GPU` .
    2. Python IDE such as [PyCharm (https://www.jetbrains.com/pycharm/)](https://www.jetbrains.com/pycharm/) or [Visual Studio Code (https://code.visualstudio.com/)](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)](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)](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)](http://tug.ctan.org/info/latex-refsheet/LaTeX_RefSheet.pdf)

Question 1 -Generalization in A Teacher-Student Setup

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

$$\overline{\mathcal{R}}(w) \triangleq \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma_\epsilon^2 I), w_{true}\sim\mathcal{N}(0,\frac{\sigma_w^2}{d}I)}\left[\mathcal{R}\right],$$

where,

$$\mathcal{R}(w_\mu) = ||w_\mu - w_{true}||^2 = ||(H_\mu^{-1}H - I)w_{true} + H_\mu^{-1}X^T\epsilon||^2$$

Prove:

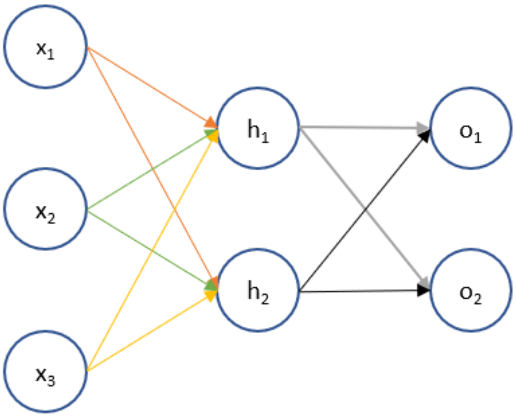
$$\mathcal{R}(w_\mu) = \sum_{i=1}^{d} \frac{(\sigma_w^2/d)\mu^2 + \sigma_\epsilon^2\lambda_i}{(\lambda_i+\mu)^2}$$

Hints:

- $\mathbb{E}\left[\epsilon^T X H_\mu^{-1} H_\mu^{-1} X^T \epsilon\right] = \sum_{i,j}^{N} \mathbb{E}[\epsilon_i\epsilon_j]\left(XH_\mu^{-1}\right)_i\left(H_\mu^{-1}X^T\right)_j$
- $\mathbb{E}[\epsilon_i\epsilon_j] = \sigma_\epsilon^2\delta_{ij}$
- $\sum_{i=1}^{N}\left(XH_\mu^{-1}\right)_i\left(H_\mu^{-1}X^T\right)_i = Tr\left[XH_\mu^{-2}X^T\right]$

Question 2 - Backpropagation By Hand

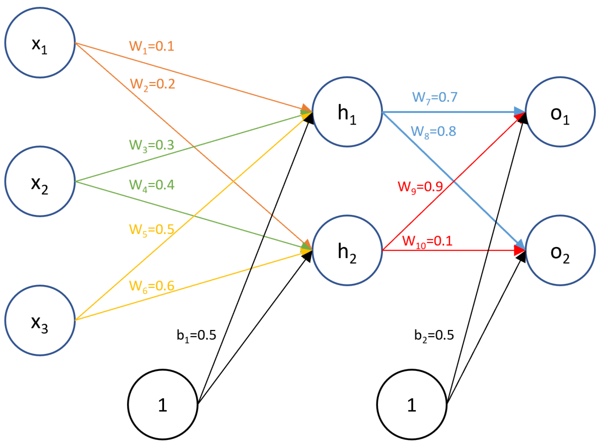
Consider the following network:



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:



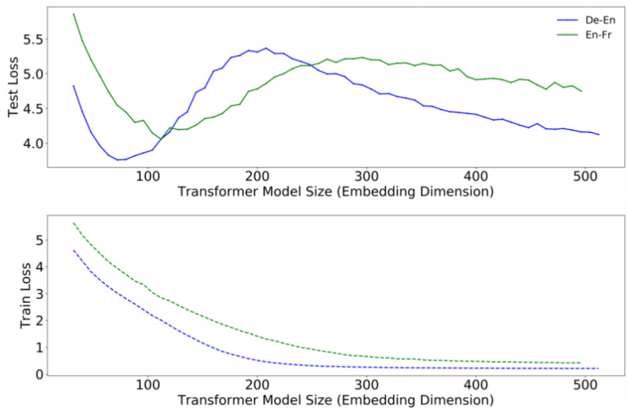
1. Perform one forward pass and calculate the MSE.
2. Perform backpropagation (one backward pass, i.e., calculate the gradients).
3. With a learning rate of $\alpha = 0.01$, what are the new values of the weights after performing the forward pass and backward pass (assume we use SGD)?

Question 3 - Deep Double Descent

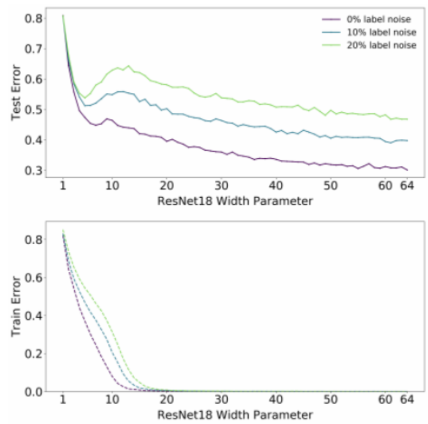
For the following plots:

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? Explain.

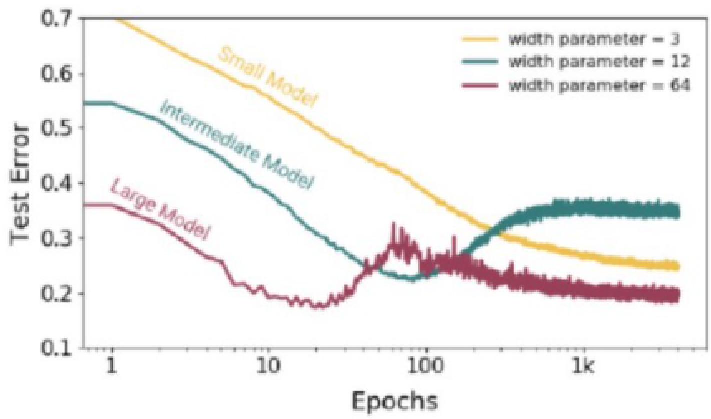
a.



b.



c.



Question 4 - Initialization

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

$$\sigma_l = \frac{1}{\sqrt{d_{l-1}\mathbb{E}_{z\sim\mathcal{N}(0,1)}\left[\varphi^2(z)\right]}}$$

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

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

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

Question 5 -Equivarinace

Recall from lecture 6: A function $f : \mathcal{R}^d \to \mathcal{R}$ is equivariant if $f(\tau \cdot x) = \tau \cdot f(x)$ for all τ.

Let $f_w(x) = \phi(Wx)$ where ϕ is a component-wise non-linearity and $W \in \mathcal{R}^{d\times d}$. Prove that $f_w : \mathcal{R}^d \to \mathcal{R}^d$ is equivariant to transformation family H **if and only if**:

$$\forall \tau \in H, W[i, j] = W[\tau(i), \tau(j)]$$

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 with N neurons, each of size $M \times M \times D$, where D is the number of filters. $stride = 1, padding = 2$
- 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)](https://gist.github.com/MattKleinsmith/5226a94bad5dd12ed0b871aed98cb123).

In [1]:
```python
# 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
import torchvision
import matplotlib.pyplot as plt
# %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 [3]:
```python
x = torch.randn(512)
for i in range(100):
    a = torch.randn(512, 512)
    x = a @ x
print(x.mean(), x.std())
```

```
tensor(nan) tensor(nan)
```

Your answers here

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

Task 2 - FashionMNIST Deep Classifer

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

1. Load the FashionMNIST dataset `torchvision.datasets.FashionMNIST` and display 6 images with their labels from the dataset.
2. Design a **MLP** to classify images from the FashionMNIST dataset. **You need to reach at least 85% accuracy on the test set, and 89% for a full grade**.
   - You have a free choice of architecture, optimizer, learning scheduler, initialization, regularization and activations.
   - In a Markdown block, write down the chosen architectures and all the hyper-parameters.
   - **Plot** the loss curves (and any oter statistic you want) as a function of epochs/iterations.
   - **Print** the test accuracy.

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

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/docs/stable/torchvision/datasets.html#svhn (https://pytorch.org/docs/stable/torchvision/datasets.html#svhn)](https://pytorch.org/docs/stable/torchvision/datasets.html#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. **Print** this number.
   - **Important** - if you used the CNN from the tutorial (`CifarCNN()`), **explain what you changed!**
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 want) 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
$$\text{image} + 0.005 \times \mathcal{N}(0,1)$$
. **Print** the result.
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.
   - **Plot** the loss curves (and any other statistic you want) as a function of epochs/iterations.

In [ ]:

```python
"""
Your Code Here
"""
```

Credits

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