Deep Learning in Hardware | ECE 498/598 (Fall 2020)
Prof. Naresh Shanbhag | Assigned: 09/03 - Due: 09/18

Homework 1

This homework contains some questions and problems marked with a star. These are intended for graduate students enrolled in the 598NSG section. Undergraduates enrolled in the 498NSU section are welcome to solve them for extra credit. The topics covered in this homework include: some general machine learning, the ADALINE algorithm, quantization, DNN complexity estimation, deep learning in Python, theoretically optimal linear prediction, and online prediction. This homework needs to be submitted through Gradescope before 5pm CT on the due date. No extension will be provided, so make sure to get started as soon as possible.

Problem 1: Learning a Boolean Function using ADALINE and MADALINE

In this problem we will test out the ADALINE and MADALINE algorithms in the context of implementing Boolean functions. Consider the following function:

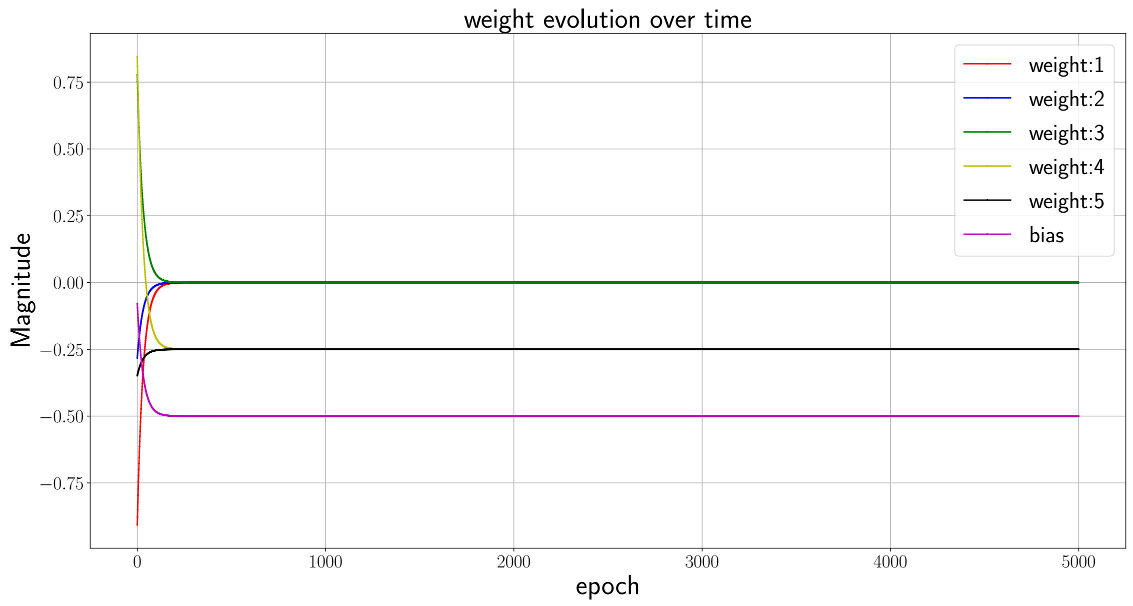
$$f(v, w, x, y, z) = (((v \oplus w) + x) \oplus y) + (y \oplus z)$$

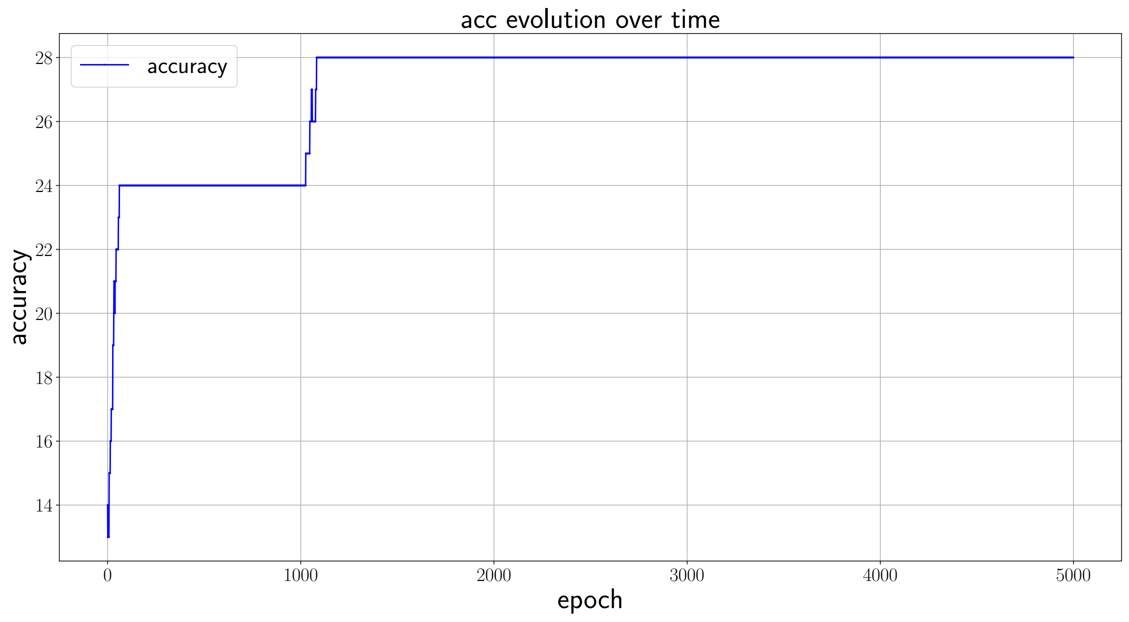
where v, w, x, y, z are binary variables in $\{-1, +1\}$, $\oplus$ is the 'xor' operation, i.e., $a \oplus b = \mathbb{1}_{\{a==b\}} - \mathbb{1}_{\{a==-b\}}$ and $+$ denotes the 'or' operation.

1. Draw the truth table of the function f.
   Solution: This is the truth table:

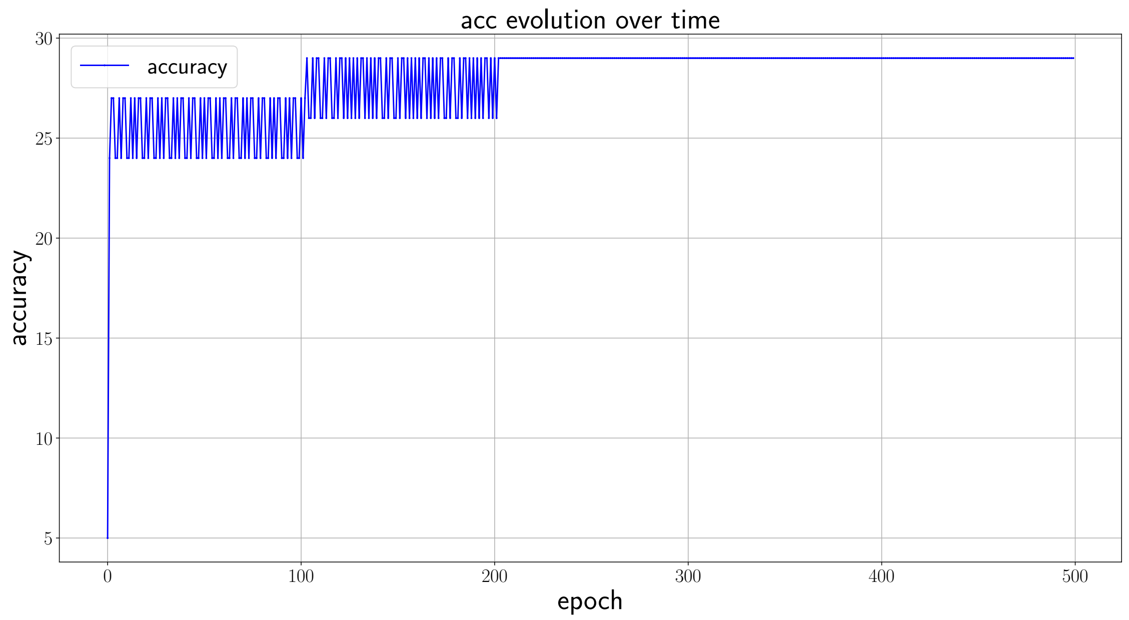
v	w	x	y	z	f
1	1	1	1	1	1
1	1	1	1	-1	-1
1	1	1	-1	1	-1
1	1	1	-1	-1	-1
1	1	-1	1	1	-1
1	1	-1	1	-1	-1
1	1	-1	-1	1	-1
1	1	-1	-1	-1	1
1	-1	1	1	1	-1
1	-1	1	1	-1	-1
1	-1	1	-1	1	-1
1	-1	1	-1	-1	1
1	-1	-1	1	1	-1
1	-1	-1	1	-1	-1
1	-1	-1	-1	1	-1
1	-1	-1	-1	-1	1
-1	1	1	1	1	-1
-1	1	1	1	-1	-1
-1	1	1	-1	1	-1
-1	1	1	-1	-1	1
-1	1	-1	1	1	-1
-1	1	-1	1	-1	-1
-1	1	-1	-1	1	-1
-1	1	-1	-1	-1	1
-1	-1	1	1	1	1
-1	-1	1	1	-1	-1
-1	-1	1	-1	1	-1
-1	-1	1	-1	-1	-1
-1	-1	-1	1	1	-1
-1	-1	-1	1	-1	-1
-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	1

2. Code the ADALINE algorithm to learn this Boolean function. Plot the evolution of the five weights and the bias. Upon convergence, how many of the 32 input combinations result in the correct output?
   Solution: We use the ADALINE Algorithm to learn this function. As shown in the plots below, the ADALINE algorithm reaches an accuracy of approximately 87.5%. It in fact classifies 28/32 combinations correctly.





3. * Code the MADALINE algorithm to learn this Boolean function. Use two layers and five units in the intermediate layer. Report the converged values of all weights and biases. How many of the 32 input combinations result in the correct output?
Solution: There are 29/32 input combinations that are correctly classified upon convergence. Below are the accuracy vs iteration plot and converged weights and biases.



First layer weight matrix:
⟨0.42332679, 0.42332679, 0.42332679, 0.42332679, 0.42332679
0.42332679, 0.42332679, 0.42332679, 0.42332679, 0.42332679
0.42332679, 0.42332679, 0.42332679, 0.42332679, 0.42332679
1.42945482, 1.42945482, 1.42945482, 1.42945482, 1.42945482
1.42945482, 1.42945482, 1.42945482, 1.42945482, 1.42945482⟩
First layer bias vecotr: ⟨2.43548689, 2.43548689, 2.43548689, 2.43548689, 2.43548689⟩
Second layer weight vector: ⟨-0.14694004, -0.14694004, -0.14694004, -0.14694004, -0.14694004⟩

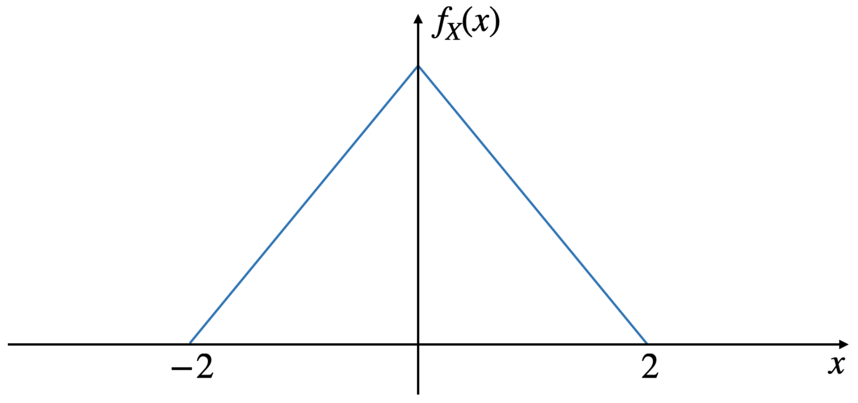


Figure 1: The considered triangular distribution in Problem 2.

Problem 2: Optimal quantization of a triangular distribution

In this problem, we will test out the Lloyd-Max (LM) algorithm on the triangular distribution depicted in Figure 1. Note: the figure is **NOT** drawn to scale.

1. Recall the quantization level update equation in the Lloyd Max iteration:

$$r_q = \frac{\int_{t_q}^{t_{q+1}} x f_X(x)dx}{\int_{t_q}^{t_{q+1}} f_X(x)dx}$$

Derive the closed form expression of this update (i.e., evaluate the above expression).
Solution: In order to solve this question, we need to evaluate two integrals per case. Thus it is important to find out the expressions describing the distribution for the different regions where the input may lie:

$$f_X(x) = \begin{cases} \frac{1}{2} + \frac{x}{4} & \text{if } x \in [-2, 0] \\ \frac{1}{2} - \frac{x}{4} & \text{if } x \in (0, 2] \end{cases}$$

In this problem it is enough to consider cases where the quantization thresholds are in the same region or in two juxtaposed regions. Indeed, from the symmetry of the distribution, we can infer that the optimal quantizer will not have thresholds two regions apart when considering 4 or more levels. We obtain the following:
Case 1: $t_{q+1} \leq 0$,

$$r_q = \frac{\int_{t_q}^{t_{q+1}} x\left(\frac{1}{2} + \frac{x}{4}\right)dx}{\int_{t_q}^{t_{q+1}} \left(\frac{1}{2} + \frac{x}{4}\right)dx} = \frac{\frac{1}{4}\left(t_{q+1}^2 - t_q^2\right) + \frac{1}{12}\left(t_{q+1}^3 - t_q^3\right)}{\frac{1}{2}\left(t_{q+1} - t_q\right) + \frac{1}{8}\left(t_{q+1}^2 - t_q^2\right)}$$

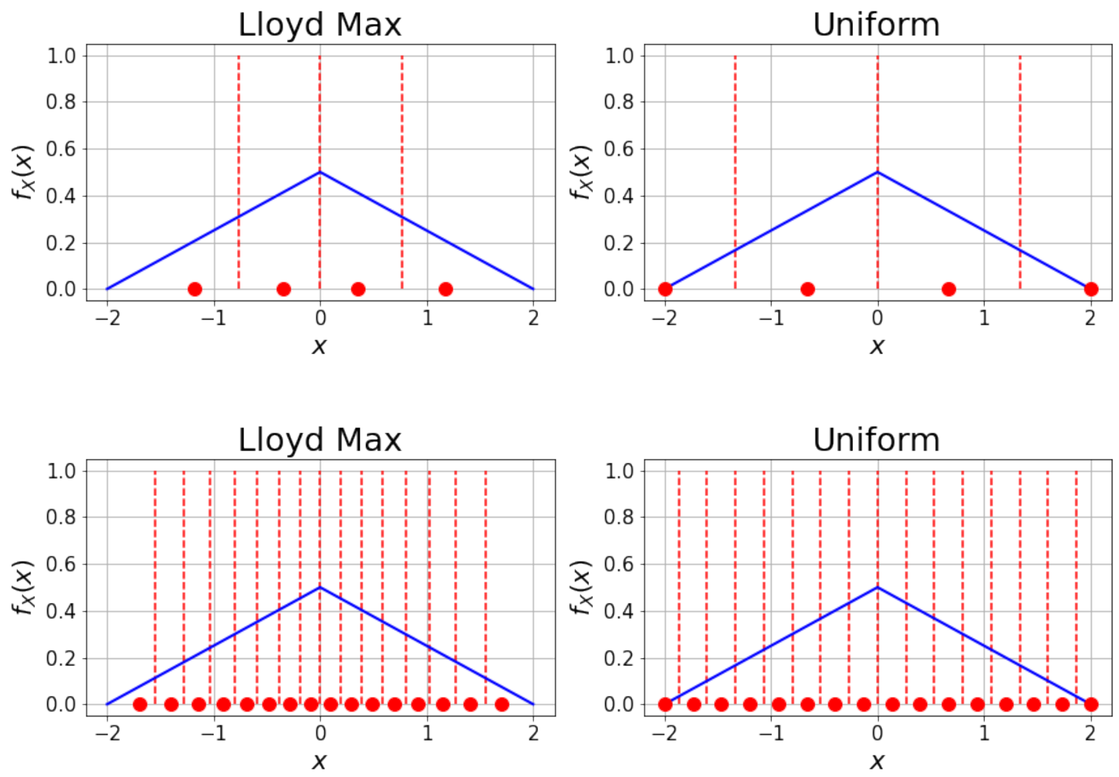
Case 2: $t_q \leq 0$ & $t_{q+1} \geq 0$,

$$r_q = \frac{\int_{t_q}^{0} x\left(\frac{1}{2} + \frac{x}{4}\right)dx + \int_{0}^{t_{q+1}} x(\frac{1}{2} - \frac{x}{4})dx}{\int_{t_q}^{0} \left(\frac{1}{2} + \frac{x}{4}\right)dx + \int_{0}^{t_{q+1}} (\frac{1}{2} - \frac{x}{4})dx} = \frac{\frac{1}{4}\left(t_{q+1}^2 - t_q^2\right) - \frac{1}{12}\left(t_{q+1}^3 + t_q^3\right)}{\frac{1}{2}\left(t_{q+1} - t_q\right) - \frac{1}{8}\left(t_{q+1}^2 + t_q^2\right)}$$

Case 3: $t_q \geq 0$,

$$r_q = \frac{\int_{t_q}^{t_{q+1}} x\left(\frac{1}{2} - \frac{x}{4}\right)dx}{\int_{t_q}^{t_{q+1}} \left(\frac{1}{2} - \frac{x}{4}\right)dx} = \frac{\frac{1}{4}\left(t_{q+1}^2 - t_q^2\right) - \frac{1}{12}\left(t_{q+1}^3 - t_q^3\right)}{\frac{1}{2}\left(t_{q+1} - t_q\right) - \frac{1}{8}\left(t_{q+1}^2 - t_q^2\right)}$$

2. Code the corresponding LM algorithm to determine the optimal quantization levels for this distribution. Plot the quantization levels and thresholds and compare with those of a uniform quantizer. Do this for quantizers with 4 and 16 levels.
Solution: We code the above update equations for the LM algorithm and compare with the uniform quantizer. We obtain the following:



The top plot is for quantizers with 4 levels. The bottom plot is for quantizers with 16 levels.

3. Empirically evaluate and report the SQNR of the LM and uniform quantizers.
Solution: Once the quantization levels are obtained we can empirically estimate the SQNR of the quantizer. Since we need to average over a set of random inputs, we need to find a way to efficiently sample from our triangular distribution, which can be easily done through Python API. We obtain the following values for the SQNRs:
Uniform quantizer - 4 levels: 6.3 dBs
Lloyd-Max quantizer - 4 levels: 10.5 dBs
Uniform quantizer - 16 levels: 20.3 dBs
Lloyd-Max quantizer - 4 levels: 21.81 dBs

4. * Derive an expression for the SQNR of the LM and uniform quantizers, i.e., compute the ratio of the data variance $\mathbb{E}\left[X^2\right]$ and the MSE of the quantizer $\mathbb{E}\left[\left(X-\hat{X}_q\right)^2\right]$. Compare your answer to the empirical SQNR obtained in the previous question.
Solution: The formula for SQNR requires us first to calculate the second moment of X;thus $\mathbb{E}\left[X^2\right]=2/3$.

In order to compute the MSE of the quantizer, we sum MSEs over quantization regions:

$$\begin{aligned}\mathbb{E}\left[\left(X-\hat{X}_q\right)^2\right] &= \sum_{\mathcal{R}_q}\int_{\mathcal{R}_q}(x-r_q)^2 f_X(x)dx \\ &= \sum_{\mathcal{R}_q}\int_{\mathcal{R}_q}x^2 f_X(x)dx - 2\sum_{\mathcal{R}_q} r_q\int_{\mathcal{R}_q}x f_X(x)dx + \sum_{\mathcal{R}_q} r_q^2\int_{\mathcal{R}_q} f_X(x)dx\end{aligned}$$

Note that $\sum_{\mathcal{R}_q}\int_{\mathcal{R}_q}x^2 f_X(x)dx = \mathbb{E}\left[X^2\right] = 2/3$. Furthermore, the quantities, $\int_{\mathcal{R}_q} x f_X(x)dx$ and $\int_{\mathcal{R}_q} f_X(x)dx$ are already computed above as they correspond to the numerators and denominators in the Lloyd-Max update equation, respectively. We just plug these values into the expression of the MSE and obtain the SQNR results. Other than this approach, scipy provides an integral package with which we can evaluate the integral directly.

Uniform quantizer - 4 levels: $\mathbb{E}\left[\left(X-\hat{X}_q\right)^2\right] = 0.156$ so that SQNR $= 6.3$

Lloyd-Max quantizer - 4 levels: $\mathbb{E}\left[\left(X-\hat{X}_q\right)^2\right] = 0.0059$ so that SQNR $= 10.32$

Uniform quantizer - 16 levels: $\mathbb{E}\left[\left(X-\hat{X}_q\right)^2\right] = 0.0059$ so that SQNR $= 20.52$

Lloyd-Max quantizer - 4 levels: $\mathbb{E}\left[\left(X-\hat{X}_q\right)^2\right] = 0.0042$ so that SQNR $= 21.95$ dBs.

5. The quantization we have been discussing so far does not assume clipping. In this problem, we are assuming a signed clipping scheme where $x_q = c$ if $x \geq c$ and $x_q = -c$ if $x \leq -c$ where c is some positive number. Derive an expression for the SQNR as a function of B_x and c. Validate your SQNR expression by plotting the empirical SQNR vs. c for the 4-level and 16-level LM quantizers and c ranges from 0.5 to 1.9. Repeat this operation for the 4-level and 16-level uniform quantizers as well. Use 1000 samples for each empirical SQNR value.

Solution: The signal power, i.e., $E[X^2]$, will stay the same as $2/3$. We need some modifications for the noise term, i.e. $E\left[\left(X-\hat{X}_q\right)^2\right]$. First, rewrite the quantization function as:

$$\hat{X}_q = \begin{cases} Q(x) & \text{if } x \in [-c, c] \\ c & \text{if } x \in (c, 2] \\ -c & \text{if } x \in [-2, -c) \end{cases}$$

where $Q(x)$ is the original quantization function without clipping. Therefore we can divide the contribution of noise power into 2 terms: one due to clipping (clipping noise), another one due to pure quantization (quantization noise). For the clipping noise, we need to evaluate the following integral:

$$N_c = 2\int_c^2 (x-c)^2 f_X(x)dx \tag{1}$$

The factor of 2 is due to symmetry from signed clipping. The evaluation yields the following polynomial in c:

$$N_c = \frac{1}{24}c^4 - \frac{1}{3}c^3 + c^2 - \frac{4}{3}c + \frac{2}{3} \tag{2}$$

We can do a quick check by pluggin in 2 and 0. If all data is clipped away, i.e., the output of the quantizer is always 0. The noise power equals the signal power of 2/3. If $c = 2$, i.e., no clipping noise is introduced, N evaluates to 0 which matches our assumption. For quantization noise N_q, we can use the approximation given in class where N_q is a uniform RV with variance $\Delta^2/12$ and Δ is $\frac{2c}{2^{B_x}}$. Hence,

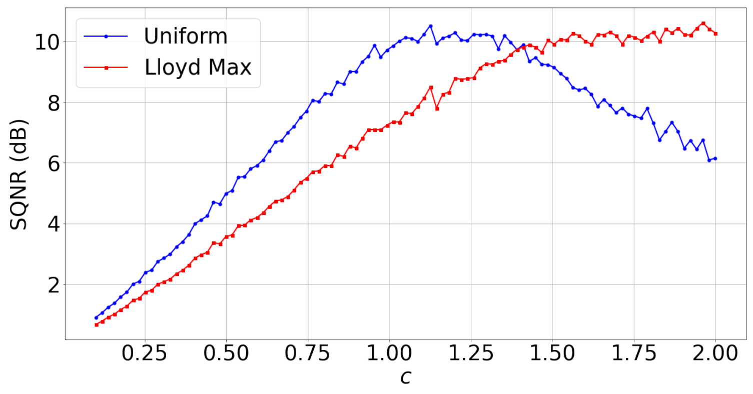
$$N_q = \frac{(\frac{2c}{2^{B_x}})^2}{12} \tag{3}$$

Therefore the SQNR in terms of B_x and c is given by:

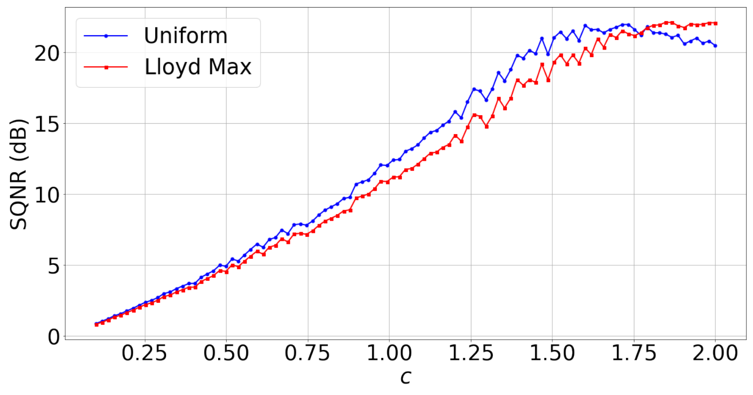
$$\text{SQNR} = 10\log_{10}\frac{2/3}{N_q + N_c}$$

where N_q and N_c are given by (2) and (3). From the SQNR equation, we can expect it to be approximately linear for small values of c, peak somewhere and start to decrease due to increasing step size. Simulation plots are given below.

4-level quantizer:



16-level quantizer:

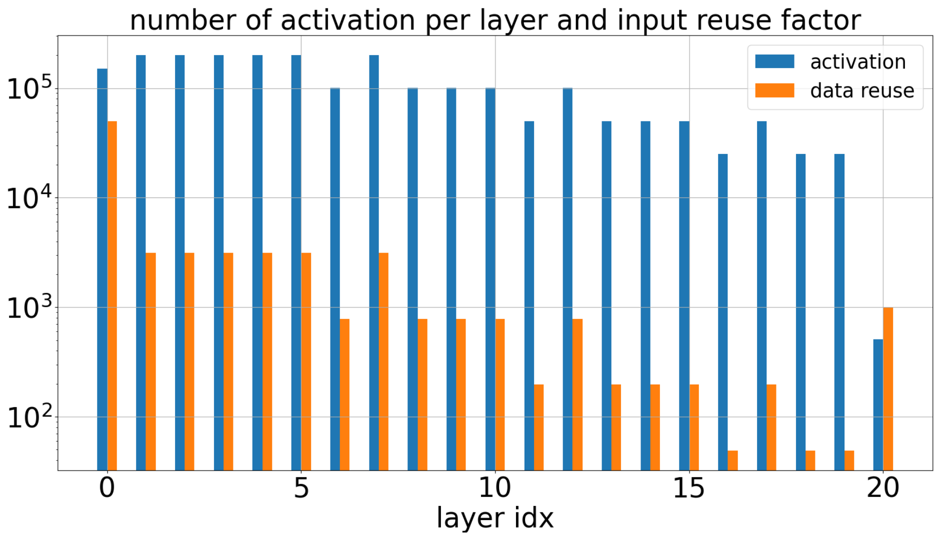


In the case of the uniform quantizer, we can see that the SQNR peaks around 1 when $M = 4$ and 1.75 when $M = 16$. The reason for this peak is the contribution of clipping and quantization noise is shifting. At low c values, the clipping noise is dominant. As c increases, the clipping noise reduces. The quantization noise increases with the increasing c, thereby lowering the entire SQNR.

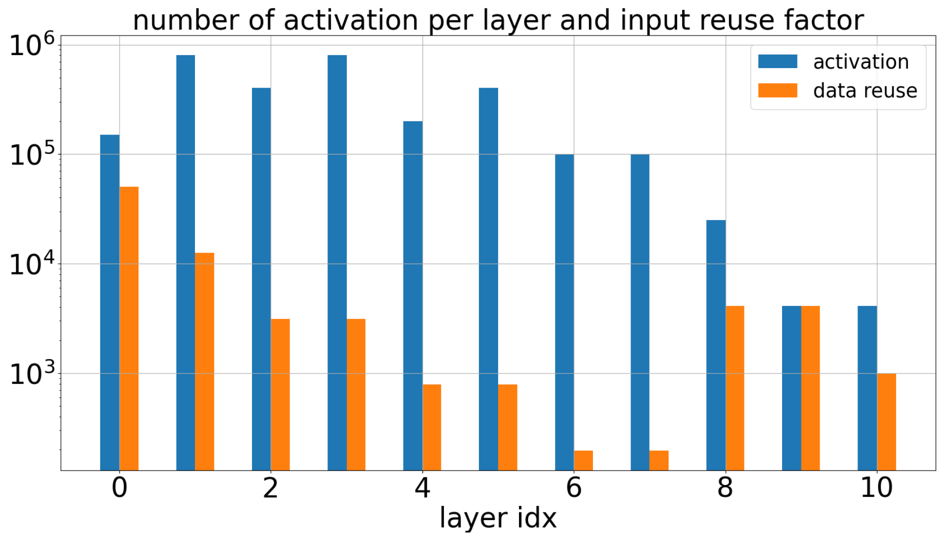
Problem 3: Profiling Deep Net Complexity

We have seen in class that an important first step in hardware implementation of DNNs is an estimation of the hardware complexity. For instance, it is very useful to understand storage and computational costs associated with every DNNs. In this problem, we start by familiarizing ourselves with DNN topologies and how to extract complexity measures from them. You are asked to consider the following two networks: ResNet-18 and VGG-11. Please refer to `https://github.com/pytorch/vision/tree/master/torchvision/models` for the exact topological description of each of these networks, and answer the following questions for both networks. You are encouraged to write python scripts using the PyTorch package to solve these problems.

1. Plot the total number of activations and the data reuse factor per layer as a function of layer index. What is the total number of activations in each network?
Solution: For the entire question, assuming the input size is $3 \times 224 \times 224$. All plots are generated based on this assumption. In order to count the number of activations, we use the fact that for convolutional layers the number of activations is equal to the number of channels multiplied by the feature dimensions and for fully connected layers the number of activations is simply equal to the number of units. Thus we have
ResNet-18: the total number of activations is roughly 2.183M (45.056K for input image size $3 \times 32 \times 32$). The number of activations per layer is given below:



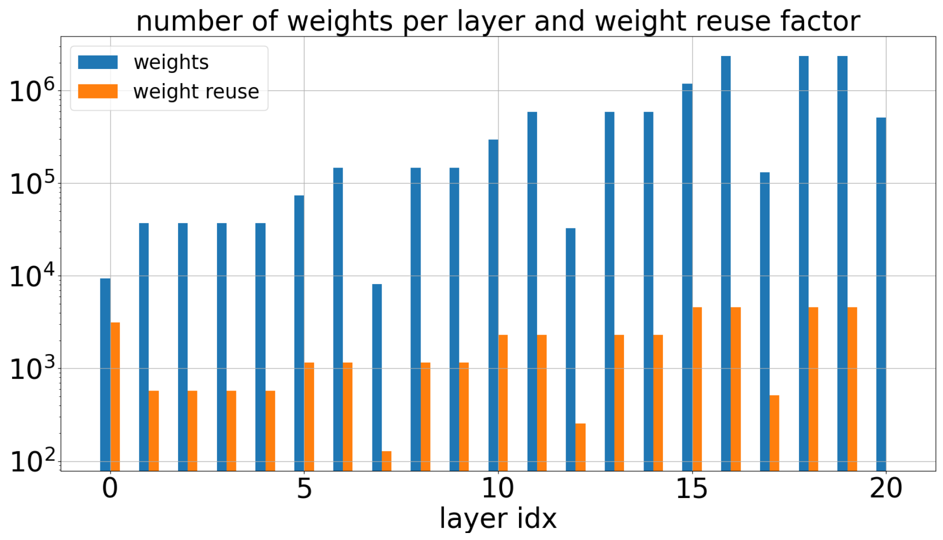
VGG-11: the total number of activations is roughly 2.994M (93.696K for input image size $3 \times 32 \times 32$). The number of activations per layer is given below:



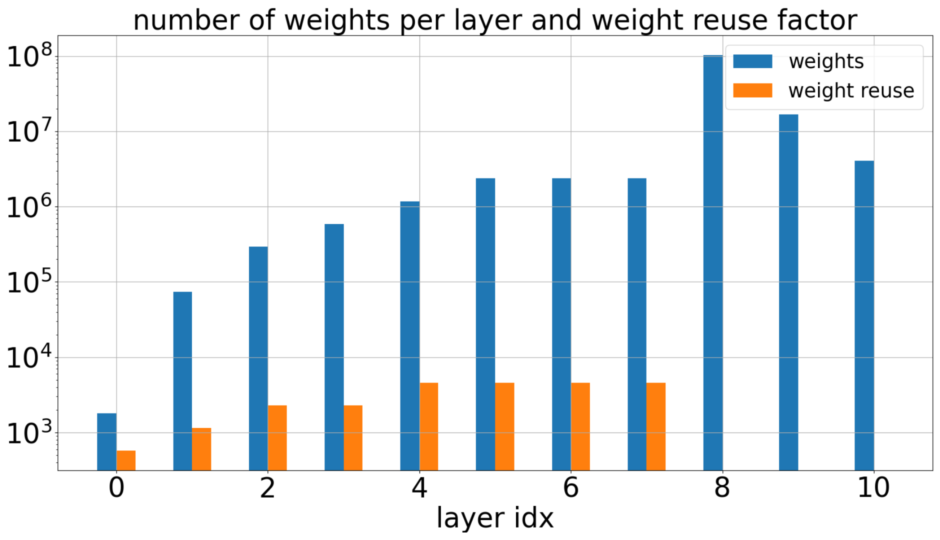
2. Plot the total number of weights and the weight reuse factor per layer as a function of layer index. What is the total number of weights in each network?

   Solution: In order to count the number of weights, we use the fact that for convolutional layers the number of weights equals to the kernel dimensions multiplied by the number of input and output channels, and for fully connected layers the number of weights equals to the product of numbers of input and output units. Thus we have

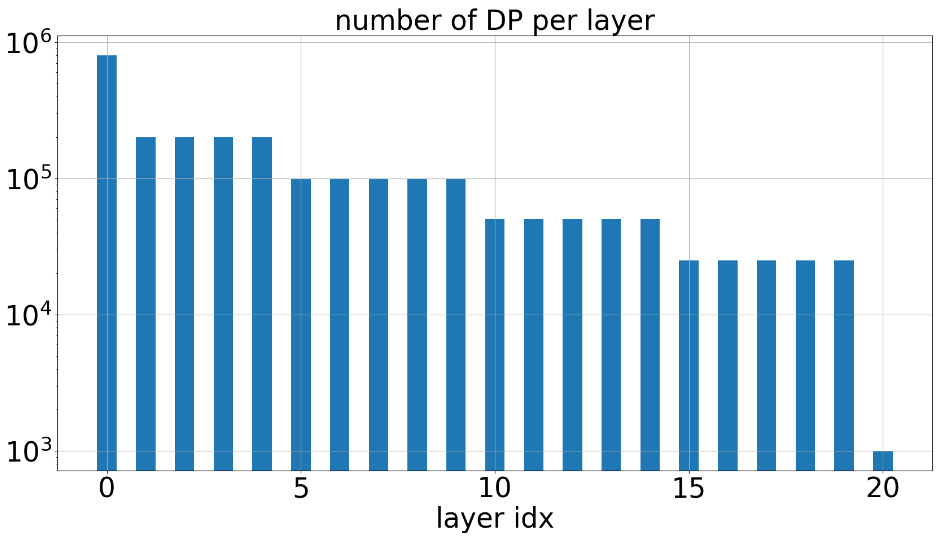
   ResNet-18: the total number of weights is roughly 11.690M. The number of weights and weight reuse are plotted below:



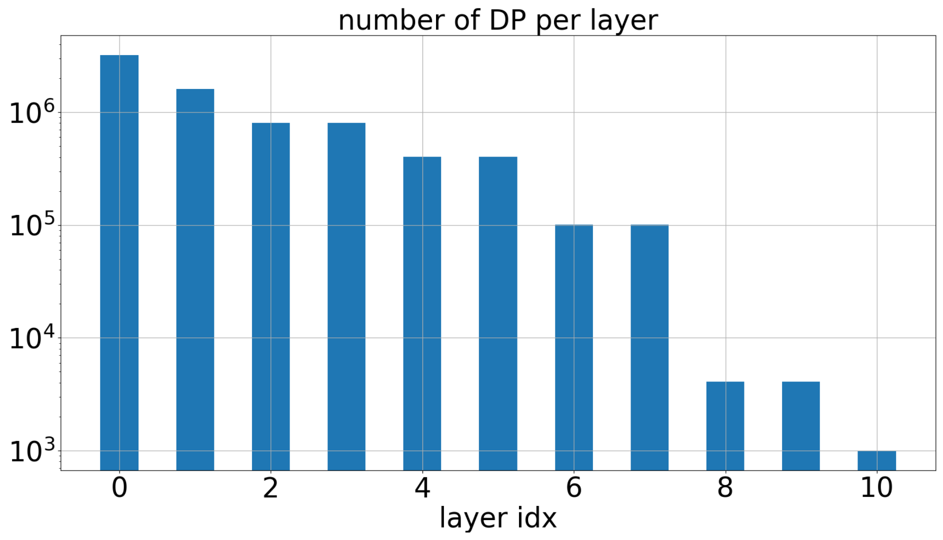
   VGG-11: the total number of weights is roughly 132.863M. The number of weights and weight reuse are plotted below:



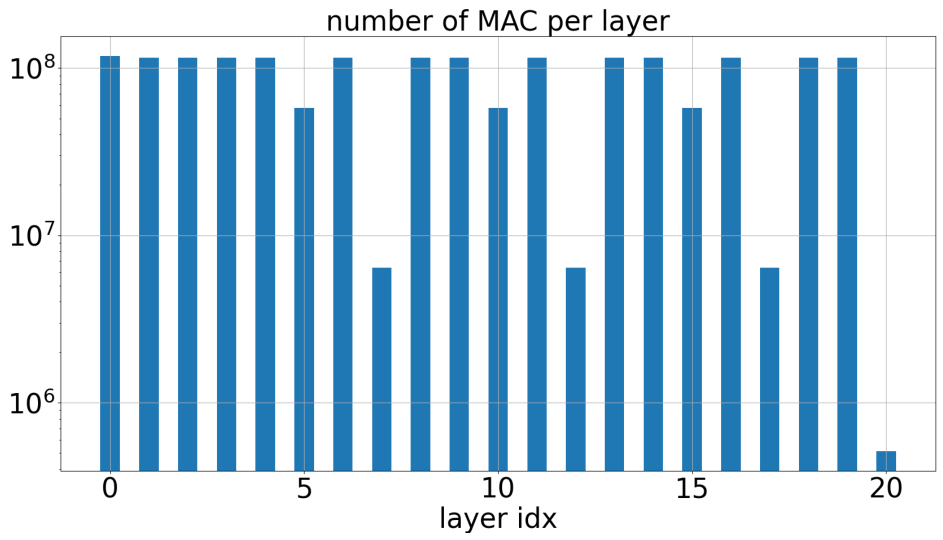
3. Plot the total number of dot products per layer as a function of layer index. How many dot products are needed per inference for each network?
Solution: In convolutional layers, the number of dot products is equal to the output feature dimension multiplied by the number of output channels. In fully connected layers, the number of dot products is equal to the number of output units. Thus we have
ResNet-18: the total number of dot products is roughly 2.485M (51.688K for input size $3 \times 32 \times 32$). The number of



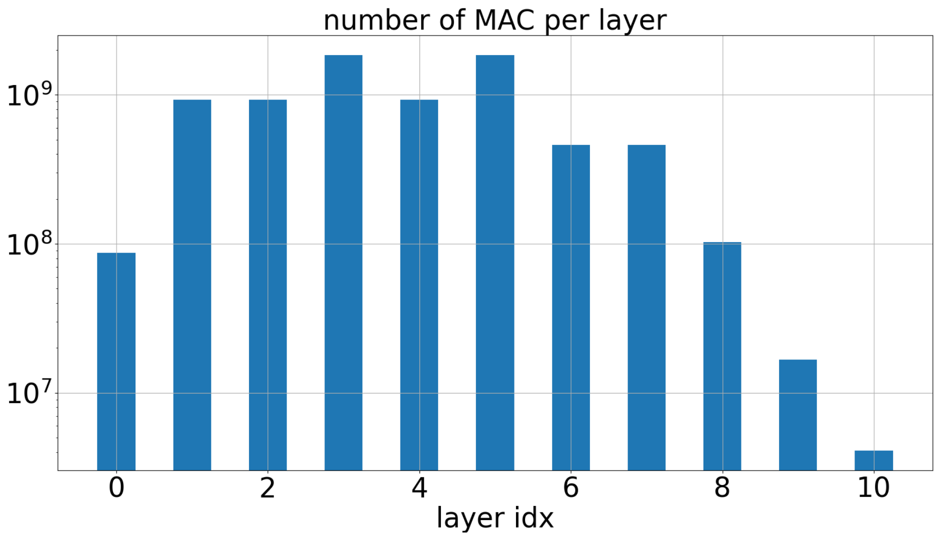
VGG-11: the total number of dot products is roughly 7.435M (160.744K for input size $3 \times 32 \times 32$).



4. Plot the total number of multiply-accumulates (MACs) per layer as a function of layer index. How many MACs are needed per inference for each network?
Solution: At each layer, the number of MACs is equal to the number of dot products multiplied by the corresponding dot product length. In convolutional layers, the dot product length is equal to the number of input channels multiplied by the kernel dimensions, and in fully connected layers, the it is equal tot he number of input units. Thus we have
ResNet-18: the total number of MACs is roughly 1.814G. (37.626M for image size $3\times32\times32$)
The number of MAC per layer is plotted below:



VGG-11: the total number of MACs is roughly 7.609G (276.575M for image size $3\times32\times32$).



5. Assuming 8-bit fixed-point format, determine the total storage requirements in MBs for all activations and weights.
Solution: The storage requirement can be computed by multiplying 8 with the total number of activations and weights. Thus:
ResNet-18: the total storage requirement is roughly 13.873 MB (11.735MB for image size $3 \times 32 \times 32$)
VGG-11: the total storage requirement is roughly 135.857MB (132.957MB for image size $3 \times 32 \times 32$)

6. If each 8-bit MAC consumes 0.5 pJ of energy, how much energy is consumed by the network to generate one decision? i.e., what is the energy/decision E_{dec}.
Solution: By multiplying the total number of MAC with energy per MAC, we get the energy per decision for different networks.
ResNet-18: the energy per decision cost is roughly 0.907 mJ (0.019 mJ for image size $3 \times 32 \times 32$.)
VGG-11: the total storage requirement is roughly 3.81 mJ (0.13 mJ for image size $3 \times 32 \times 32$.)

7. If each 8-bit MAC takes 1 ns to compute, and there are N such MACs operating in parallel with 100% utilization, what is the minimum value of N required to support a frame-rate of 30 frames/s.
Solution:
$$N = \lceil \text{MAC per frame} \times \text{delay per MAC} \times \text{frame rate} \rceil \tag{4}$$
ResNet-18: $N = 55$ (2 if image size $3 \times 32 \times 32$)
VGG-11: $N = 229$ (9 if image size $3 \times 32 \times 32$)

Problem 4: Getting Started with Deep Learning in Python

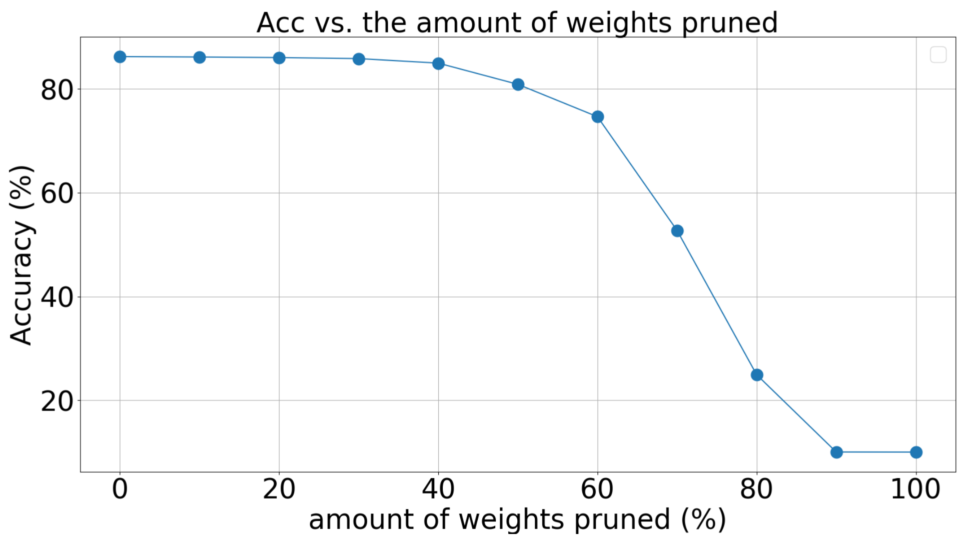
In this problem, we will get started with Deep Learning in Python using the PyTorch framework. We will use the CIFAR-10 dataset which you can download from this link `https://www.cs.toronto.edu/~kriz/cifar.html`. On the course website, you will find a link to downaload all necessary files including a pre-trained model and a script named "inference.py" which you can use to evaluate the model.

1. From the provided code and data, please briefly describe the dataset and the network provided.
   Solution: The dataset used is CIFAR-10. It consists of small RGB images of size 32x32x3. The network used in this example is an AlexNet model.

2. Run the provided code and determine the accuracy of this model on the CIFAR-10 task. Please report this number.
   Solution: Running the code provided, we see that the model achieves an accuracy of 86.24% on the CIFAR-10 dataset.

3. One method to reduce the complexity of a network is pruning which means to zero out some of the weights in the network. There are various strategies to prune a network. In this problem, plot the network's classification accuracy vs. the fraction of pruned weights where pruning is done by removing (zeroing out) weights whose magnitude is less than a specified threshold.

   Solution: We zero out $x\%$ of weights of the given network with the lowest magnitude and re-evaluate the network. The accuracy of the pruned network vs. the amount of weights pruned in percent is given below:



Problem 5: Theoretically Optimal Linear Predictor for Time Series

The data generation model is given by:

$$x_n = 0.1g_n + 0.5g_{n-1} - 0.5g_{n-2} + 0.1g_{n-3} \tag{5}$$

where g_n are i.i.d. $\mathcal{N}(0,1)$ random variables. We wish to predict x_n from its previous samples

x_{n-1}, x_{n-2}, and x_{n-3}, i.e., we wish to find a function $\hat{x}_n = f(x_{n-1}, x_{n-2}, x_{n-3})$ such that the cost function $\mathbb{E}\left[e_n^2\right] = \mathbb{E}\left[(\hat{x}_n - x_n)^2\right]$ is minimized. This cost function is called the mean squared error (MSE). Furthermore, for simplicity, we wish to restrict the predictor f to be a linear function of its arguments as shown below:

$$\hat{x}_n = w_1 x_{n-1} + w_2 x_{n-2} + w_3 x_{n-3} \tag{6}$$

1. Simulate the data generation model (5) and the inference model (6) with $w_1 = 0.4$, $w_2 = -0.1$, and $w_3 = 0.02$ and empirically estimate the MSE using 1000 samples.
Solution: An empirical estimate of the MSE is found by averaging 1000 the squared differences $(\hat{x} - x)^2)$ and the value is approximately 0.82.

2. Obtain the optimal predictor coefficients.
Solution: We want to minimize:

$$\mathbb{E}\left[e_n^2\right] = \mathbb{E}\left[(\hat{x}_n - x_n)^2\right] = \mathbb{E}\left[\left(\mathbf{w}^T\mathbf{x}_n - x_n\right)^2\right]$$

where $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$ and $\mathbf{x}_n = \begin{bmatrix} x_{n-1} \\ x_{n-2} \\ x_{n-3} \end{bmatrix}$. Rearranging the terms above, we have

$$\mathbb{E}\left[e_n^2\right] = \mathbf{w}^T\mathbf{R}\mathbf{w} - 2\mathbf{w}^T\mathbf{p} + \sigma_x^2$$

where $\mathbf{R} = \mathbb{E}\left[\mathbf{x}_n\mathbf{x}_n^T\right]$ is the auto-correlation matrix, $\mathbf{p} = \mathbb{E}\left[x_n\mathbf{x}_n^T\right]$ is the cross-correlation vector, and $\sigma_x^2 = \mathbb{E}\left[x_n^2\right]$ is the input variance. Taking the gradient of $\mathbb{E}\left[e_n^2\right]$ with respect to $\mathbf{w}$ and setting to zero yields:

$$\mathbf{w}^{(opt)} = \mathbf{R}^{-1}\mathbf{p}$$

which is commonly known as the Weiner-Hopf solution. In our particular example, we have

$$\mathbf{R} = \begin{bmatrix} 0.52 & 0.25 & 0 \\ -0.25 & 0.52 & -0.25 \\ 0 & -0.25 & 0.52 \end{bmatrix} \quad \& \quad \mathbf{p} = \begin{bmatrix} -0.25 \\ 0 \\ 0.01 \end{bmatrix}$$

and consequently

$$\mathbf{w}^{(opt)} = \mathbf{R}^{-1}\mathbf{p} = \begin{bmatrix} -0.679 \\ -0.412 \\ -0.179 \end{bmatrix}$$

3. Find the minimum MSE of your optimal predictor.
Solution: Now that we have the closed form expression for the optimal weights, we can plug that into the formula for the MSE:

$$MSE_{min} = \left[\mathbf{w}^{(opt)}\right]^T \mathbf{R}\mathbf{w}^{(opt)} - 2\left[\mathbf{w}^{(opt)}\right]^T \mathbf{p} + \sigma_x^2$$

which can be shown to be equal to:

$$MSE_{min} = \sigma_x^2 - \left[\mathbf{w}^{(opt)}\right]^T \mathbf{p}$$

which in our case evaluates to 0.352. Note if we substitute the weight values in question 1 with the solution of Wiener-Hopf, we will reach the same empirical estimation error.

Problem 6: Computationally Efficient Predictor *

For this problem, please refer to the previous problem and answer the following questions:

1. What makes this solution hard to compute (hint: consider an N-tap predictor where N is large)?
   Solution: There are two aspects of the solution that are computationally unfriendly. Indeed, $\mathbf{w}^{(opt)} = \mathbf{R}^{-1}\mathbf{p}$ requires one matrix inversion and one matrix-vector multiplication. The complexity is $O(n^3)$.

2. Can you suggest a computationally friendly algorithm to obtain your solution?
   Solution: It is possible to compute the optimal weights via iterative methods. The first option is to use gradient descent. Indeed, we have that

$$\nabla_w \left(\mathbb{E}\left[e_n^2\right]\right) = 2\left(\mathbf{R}\mathbf{w} - \mathbf{p}\right)$$

   Thus, one algorithm to compute the optimal weights is to iteratively perform gradient descent, i.e., start with a random vector $\mathbf{w}_0$ and progressively perform the following update:

$$w_{t+1} \leftarrow w_t - \mu\left(\mathbf{R}\mathbf{w}_t - \mathbf{p}\right)$$

   where μ is the learning rate. Note that while this approach suppresses the burden of the matrix inversion, it still requires a matrix-vector multiplication at every iteration and it assumes knowledge of data statistics ($\mathbf{R}$ and $\mathbf{p}$). One other option is to perform stochastic gradient descent (SGD) which is an online learning algorithm. SGD approximates the loss function $\mathbb{E}\left[e_n^2\right]$ by it's instantaneous value e_n^2 at each iteration. We have:

$$\nabla_w \left(e_n^2\right) = 2e_n x_n$$

   and hence the update equation is:

$$w_{n+1} \leftarrow w_n - \mu e_n x_n$$

   The algorithm above is commonly known as the least-mean-square (LMS) algorithm which we will revisit in this course.