Deep Learning in Hardware
Prof. Naresh Shanbhag

# Homework 1

ECE 498/598 (Fall 2020)
Assigned: 09/03 - Due: 09/18

*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 $f$.

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?

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?

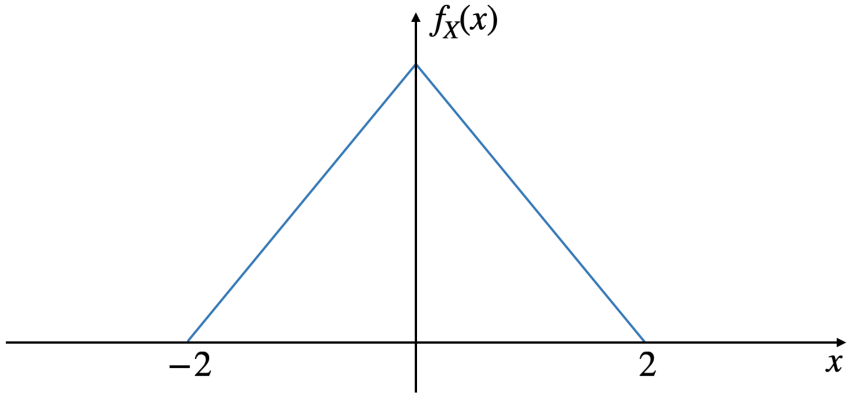

Figure 1: The considered triangular distribution in Problem 2.

## **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).

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.

3. Empirically evaluate and report the SQNR of the LM and uniform quantizers.

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.

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.

## **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?

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?

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?

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?

5. Assuming 8-bit fixed-point format, determine the total storage requirements in MBs for all activations and weights.

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 $E_{\text{dec}}$.

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.

**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.

2. Run the provided code and determine the accuracy of this model on the CIFAR-10 task. Please report this number.

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.

**Problem 5**: Theoretically Optimal Linear Predictor for Time Series

The data generation model is given by:

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

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:

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

1. Simulate the data generation model (1) and the inference model (2) with $w_1 = 0.4$, $w_2 = -0.1$, and $w_3 = 0.02$ and empirically estimate the MSE using 1000 samples.

2. Obtain the optimal predictor coefficients.

3. Find the minimum MSE of your optimal predictor.

**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)?

2. Can you suggest a computationally friendly algorithm to obtain your solution?