Project Report

On

Multi-objective Precision Optimization of Deep Neural Networks

Submitted by

Divyansh Choudhary
Yasasvi V Peruvemba

Computer Science and Engineering

3rd year

Under the Guidance of

Dr. Kapil Ahuja

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Autumn 2019

Introduction

There are many obvious value propositions and use cases for the implementation of deep neural networks on resource constrained platforms. We aim to implement a method that strikes a middle path that efficiently allocates bit width at the level of layers, and take into consideration constraints on resources.

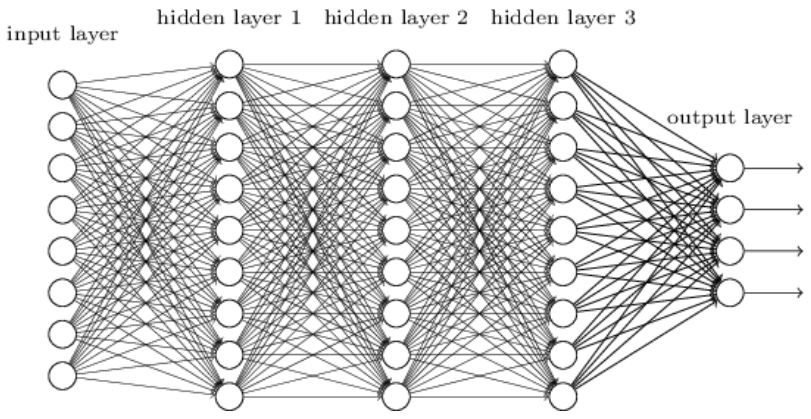
Prerequisites

We have used the MNIST dataset of images to train our self implemented neural network to obtain weights, in order to perform mathematical computations that will help us solve the problem of assigning bit width layer by layer.

The network consists of 5 layers, trained stochastically for 10 epochs, resulting in a final accuracy of ~ 93%

The nodes in each layer of the network are : [784,160,80,40,20,10]

These weights and biases of the network were then stored, to perform the inject and extract algorithm.



Optimization Problem

For pooling layers, max pooling does not affect the output rounding error, i.e. if y = max pool(x) then $\sigma_y = \sigma_x$, since the error of y is a sub-sample of the error in x.

The popular nonlinear activation function ReLU(x) = max{0, x} does not break the linear relationship between s.d. of value passing through it. Consider the output y = ReLU(x). Since the rounding error of exact 0 values when using fixed-point is 0, regardless of the precision we choose, having more zero values after ReLU will scale down the s.d. while keeping the mean at 0. Thus, $\sigma_y = \alpha \cdot \sigma_x$, with some constant α.

We define the problem as follows -

ΔX_K = Error allowed in Layer X

$\sigma X_{K->L}$ = Standard Deviation of Outputs caused by error in Layer K

From [1],

We know that there exists a linear relation between the Error in Layer X and the Standard Deviation from unmanipulated network output.

For all K in the network,

$$\Delta X_K = \lambda_K * \sigma X_{K->L} + \theta_K$$

Hence, we use linear regression to decipher the values of lambda and theta pertaining to each layer.

$\sigma X_{K \text{ -> } L} \Rightarrow \sigma_Ł \sqrt{\xi_K}$, We replace these errors from a layer, by a distribution of overall error tolerance with the help of new variables ξ_K.

Now, we frame the optimization problem as given in [1],

$\Delta X_K = \lambda_K * \sigma_Ł \sqrt{\xi_K} + \theta_K$, For every layer, Number of bits allowed would by inverse logarithm of the error allowed for that layer.

Hence, the optimization problem finally presents itself as -

$$\min \mathbf{F} = \Sigma \rho_K(-\log_2(\Delta X_K)) \quad \text{subject to } \Sigma \xi_K = 1$$

Determining Regression Coefficients

We began by amassing the weights and biases of the trained neural network. We then select 50 images for which we will create a dataset to apply regression to. We record the actual values of the outputs from the last layer of these 50 images. We iterate over every image 20 times, for K^{th} layer, whilst inducing an error of ΔX for each iteration, we then inject a uniform error from the range $(-\Delta X, \Delta X)$ to the input of the K^{th} layer. We then note down the standard deviation of the output from the last layer as σ_K. We have now generated the dataset for each layer, with 1000 entries each (50 images * 20 iterations per image).

Linear Regression pseudo code :

```
SUB Regress(x, y, n, a1, a0, syx, r2)

  sumx = 0: sumxy = 0: st = 0
  sumy = 0: sumx2 = 0: sr = 0
  DOFOR i = 1, n
    sumx = sumx + x_i
    sumy = sumy + y_i
    sumxy = sumxy + x_i*y_i
    sumx2 = sumx2 + x_i*x_i
  END DO
  xm = sumx/n
  ym = sumy/n
  a1 = (n*sumxy - sumx*sumy)/(n*sumx2 - sumx*sumx)
  a0 = ym - a1*xm
  DOFOR i = 1, n
    st = st + (y_i - ym)^2
    sr = sr + (y_i - a1*x_i - a0)^2
  END DO
  syx = (sr/(n - 2))^0.5
  r2 = (st - sr)/st

END Regress
```

Writing a simple Linear regression problem to solve this, we get the following results -

Layer No.	**Lamda**	**Theta**	**Bits Assigned** **(After optimization)**
1	0.612293131692 97480	-0.03371212188 158611	4
2	0.843734682196 4328	-0.04650270480 6702023	3
3	5.033100125970 339	-0.27769976431 01508	1
4	2.465840333955 5597	-0.13602926735 116322	2
5	3.103776965919 5202	-0.17124393186 657508	1

Implementation

Find the implementation of the Neural Network, the Dataset creation, the Linear Regression followed by the SQP octave implementation and BitAssignment in the GitHub repository below -

GitHub Repository

Optimization Technique:

We used Sequential Quadratic Programming (SQP) technique to solve our optimization problem. This is because, our objective function was non-linear. SQP technique finds the local optimum for non-linear constrained optimization problems. It was introduced by R.B. Wilson in the early 1960s, followed by variants by W. Murray and M.C. Biggs, and then popularized and refined by S.P. Han and M.J.D. Powell. SQP involves recursive use of quadratic programming to calculate the iterative improvements to the estimates of the constrained optimum using Lagrange and Kuhn-Tucker multipliers. Here is the pseudo-code:

Algorithm 3.1 Parametric SQP Method.

1: Begin with $(\bar{x}, \bar{y})$, an approximate solution to (1.1) at $t = 0$.
2: Set constants $\gamma, \gamma_1 \in (0, 1), \gamma_2 > 1$.
3: **while** $t < 1$ **do**
4: Calculate $\mathcal{A}_{+,\gamma}(\bar{x}, \bar{y}, t)$ by (3.3).
5: Solve (3.4) for $(\Delta x, \Delta y)$.
6: **if** $\eta(\bar{x} + \Delta x, \bar{y} + \Delta y, t + \Delta t) \leq \eta(\bar{x}, \bar{y}, t + \Delta t)^{1+\gamma}$ **then**
7: Let $\bar{x} = \bar{x} + \Delta x$, $\bar{y} = \bar{y} + \Delta y$, and $t = t + \Delta t$.
8: Let $\Delta t = \min(1 - t, \gamma_2 \Delta t)$.
9: **else**
10: Let $\Delta t = \gamma_1 \Delta t$. Go back to Step 5
11: **end if**
12: **end while**

Algorithm 5.1 Predictor–corrector algorithm for degenerate parametric nonlinear programming.

1: Begin with $(\bar{x}, \bar{y})$, an approximate solution to (1.1) at $t = 0$
2: Set constants γ, $\gamma_1 \in (0, 1)$, $\gamma_2 > 1$.
3: Estimate $\mathcal{A}_\gamma(\bar{x}, \bar{y}, 0)$ by (2.1).
4: Solve (5.2) for each $k \in \{1, \ldots, N_0\}$.
5: **while** $t < 1$ **do**
6: **for** each k **do**
7: Solve (5.4) for each $j \in \{1, \ldots, N_k\}$ to obtain $\{\hat{x}, \hat{y}\}_{kj}$.
8: Discard all $(\hat{x}, \hat{y})_{kj}$ that do not satisfy (5.5).
9: **if** $\{(\hat{x}, \hat{y})_{kj}\} = \emptyset$ **then**
10: Let $(\hat{x}_{kj}, \hat{y}_{kj}) = (\bar{x}, \bar{y}_k)$ for all j.
11: **end if**
12: **for** each (k, j) such that $(\hat{x}, \hat{y})_{kj}$ exists **do**
13: Calculate $\mathcal{A}_\gamma(\hat{x}_{kj}, \hat{y}_{kj}, t + \Delta t)$ by (2.1).
14: Solve (5.6) for $(\Delta x_{kj}, \Delta y_{kj})$.
15: **if** $(\Delta x_{kj}, \Delta y_{kj})$ satisfies (5.7) **then**
16: Estimate $\mathcal{A}_\gamma(\hat{x}_{kj} + \Delta x_{kj}, \hat{y}_{kj} + \Delta y_{kj}, t + \Delta t)$ by (2.1).
17: Solve (5.8) for each $l \in \{1, \ldots, N_{kj}\}$ to obtain $\{\tilde{y}_{kjl}\}$.
18: **end if**
19: **end for**
20: **end for**
21: **if** no satisfactory $(\Delta x_{kj}, \tilde{y}_{kjl})$ is found for all l, k, j **then**
22: Let $\Delta t = \gamma_1 \Delta t$, go back to Step 7.
23: **else**
24: For some k, j, and for all l with a satisfactory $(\Delta x_{kj}, \tilde{y}_{kjl})$,
25: Let $\bar{x} = \hat{x}_{kj} + \Delta x_{kj}$, $N_0 = \hat{N}_{kj}$,
26: $\{\aleph_1, \aleph_2, \ldots, \aleph_{N_0}\} = \{\bar{\aleph}_{kj1}, \bar{\aleph}_{kj2}, \ldots, \bar{\aleph}_{kj\hat{N}_{kj}}\}$
27: $\{\tilde{y}_1, \ldots, \tilde{y}_{N_0}\} = \{\tilde{y}_{kj1}, \ldots, \tilde{y}_{kj\hat{N}_{kj}}\}$, $t = t + \Delta t$.
28: Let $\Delta t = \max(1 - t, \gamma_2 \Delta t)$.
29: **end if**
30: **end while**

We are using Octave's implementation of SQP to solve our non-linear optimization problem and the results of the same are given below.

Solving the Optimization problem :

We optimized our objective function using the SQP ultility provided by Octave. We took (0.30, 0.15, 0.18, 0.20, 0.17) (random values) as our initial guess for Ɛ and We considered a standard deviation of 0.30 as acceptable for the output. SQP took 11 iterations to minimize our objective function to a value of 13.219. The results of our optimization are below -

```
octave:255> function r = g (x)
  r = [x(1) + x(2) + x(3) + x(4) + x(5) - 1];
endfunction

function obj = fn(x)
        lam = [0.61229313169297480 0.8437346821964328 5.033100125970339 2.4658403339555597 3.1037769659195202];
        thet = [-0.03371212188158611 -0.046502704806702023 -0.2776997643101508 -0.13602926735116322 -0.17124393186657508];
        sigma = 0.32;

        obj=0;
        for i = 1:5
                obj = obj - log2(lam(i)*sigma*sqrt(x(i)) + thet(i));
        endfor
endfunction

x0 = [0.30; 0.15; 0.18; 0.20; 0.17];
[x, obj, info, iter, nf, lambda] = sqp (x0, @fn, @g, [], 0, +realmax)

x =

   0.19986
   0.19996
   0.20006
   0.20005
   0.20006

obj =  13.219
info =  104
iter =  11
nf =  24
lambda =

  -5.86733
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
   0.00000
```

As we can see, the minimizer for our problem is:

(0.19986, 0.19996, 0.20006, 0.20005, 0.20006)

When we use this minimization value to find the optimal number of bits, we get the following:

```
1  import numpy as np
2  lam = [0.61229313169297480, 0.8437346821964328, 5.033100125970339, 2.4658403339555597, 3.1037769659195202]
3  thet = [-0.03371212188158611, -0.046502704806702023, -0.2776997643101508, -0.13602926735116322, -0.17124393186657508]
4  sigma = 0.32
5  x = [0.19986, 0.19996, 0.20006, 0.20005, 0.20006]
6  for i in range(0,5):
7      print("Layer {} requires: {} bits".format(i, np.floor(-np.log2(lam[i]*sigma*np.sqrt(x[i]) + thet[i]))))
8

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

➜ ~ python code.py
Layer 0 requires: 4.0 bits
Layer 1 requires: 3.0 bits
Layer 2 requires: 1.0 bits
Layer 3 requires: 2.0 bits
Layer 4 requires: 1.0 bits
```

Here we can see that the max number of bits we require is 4 (for the first layer). To see how good this is, we must consider that the IEEE-754 format keeps 23 bits to represent the fractional part of number and we are getting a reasonable accuracy with only 4 bits. If we take this into account while designing hardware, we can end up saving a huge amount of space.

Impact on Accuracy :

As we know, we initially started with an acceptable tolerance level in the standard deviation in the last layer of the network. It is vital for us to measure the impact the standard deviation has on the accuracy of the classifying network. We found no conclusive relation between standard deviation and the accuracy of the network, but it is trivial to note the dependency between the two. The lesser the standard deviation, the lesser the loss in accuracy.

Now, one of the problems we faced during optimising the objective function is the fact that in order to achieve the best possible distribution of standard deviation among the layers, sometimes due to very small standard deviations, the value of the objective function became imaginary.

We then tried to find the values of standard deviation for which there would occur no such problems (SD = 0.17), which led to distribution of bits as follows:

Layer No.	Bits Allocated
1	2
2	3
3	5
4	5
5	6

This led to us writing another piece of code which compared the accuracy of the network when bit width was manipulated vs. when it wasn't.

Below is the code to simulate assignment of bits to a particular value -

```
val = int((number)*2**(num_bits_allocated))
val = val/2**(num_bits_allocated)
number = np.float32(val)
```

When simulated with these bit assignment for each layer, on picking a random 10000 images from 60000, and using them as the testing set, the manipulated network performed just under 1% margin of error.

```
=== RESTART: C:/Documents #2/ML Examples/DigitRecognitionMNIST/Accuracy.py ===
Accuracy without Bit Assignment of Epoch 1 : 9319 / 10000
Accuracy with Bit Assignment of Epoch 1 : 9250 / 10000
```

As we can see, this particular random set of 10000 images gave a loss of only about 0.7%, which we consider to be quite remarkable. The bit lengths used were, [2,3,5,5,6] respectively, which when compared to the 23 bits IEEE standard provides, is a very good improvement.

Conclusion :

This project has been instrumental in understanding Neural Networks and Error propagation through them. We also gained insight into the application of Optimization techniques (SQP) in problems of great scientific importance. We have optimized the number of fractional bits that need to be allocated to a neural network which would work on image classification (MNIST), which can be fruitful whilst designing minimal hardware solutions (Cost-effective) to such problems. We also tried to emulate the assignment of bits, which in turn let us visualise the impact that changing the bit width of the network had on it's classification accuracy.

References

1. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8714785
2. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7783722
3. https://arxiv.org/pdf/1511.06393.pdf
4. https://link.springer.com/article/10.1007/s10589-014-9696-2