Truncation Experiment Results

Serena Zhang, 7/29/2021

Introduction

Truncate the number of digits after the decimal point and to test how that affects number recognition precision with the small subsets of the MNIST data set downloaded from http://pjreddie.com/projects/mnist-in-csv/.

Test 1

Methodology

To do truncation and get the desired number of digits after the decimal point, I used the following function where the value numDigits should be no more than 18 (explained later).

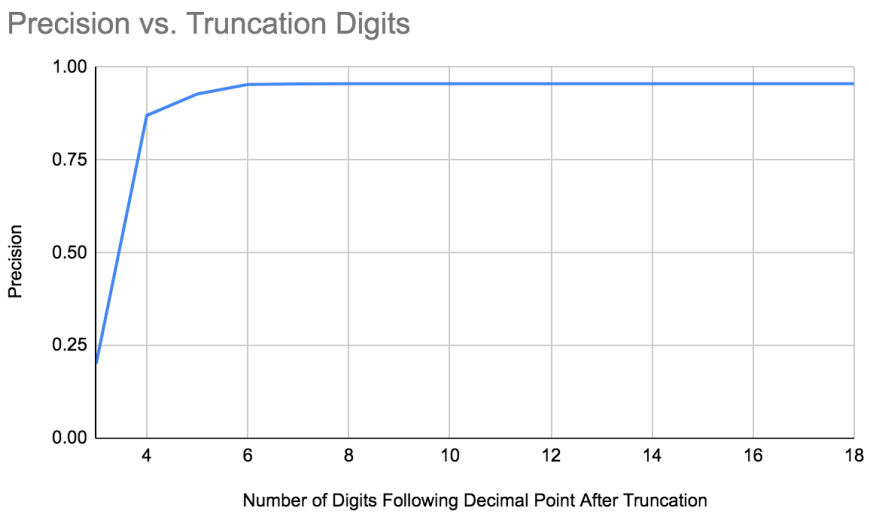
```
func truncate(m *mat.Dense) {
    rows, columns := m.Dims()
    tenthPower := math.Pow(10, float64(numDigits))
    for r:=0; r<rows; r++ {
        for c:=0; c<columns; c++ {
            m.Set(r, c, float64(int(m.At(r, c) * tenthPower)) / tenthPower)
        }
    }
}
```

I truncated the floating numbers in net.outputWeights and net.hiddenWeights in the “train” section as well as in hiddenInputs, hiddenOutputs, finalInputs, and finalOutputs in the “predict” section.

Results

The following charts show the precisions of when the number of digits after the decimal point range from 3 to 18.

digits	precision
3	0.2011
4	0.8703
5	0.9279
6	0.9538
7	0.9553
8	0.9555
9	0.9555
10	0.9555
11	0.9555
12	0.9555
13	0.9555
14	0.9555
15	0.9555
16	0.9555
17	0.9555
18	0.9555



Caveat: when calculating `int(m.At(r, c))`, because the max_int64 is 9,223,372,036,854,775,807, if numDigits > 18, that expression will overflow to a negative value. Thus, I only truncate up to 18 digits after the decimal point.

Test 2

Methodology

I wrote the following function that truncates a given number of bits from the floating number.

```
func truncateMantissa(a float64) float64{
        isNegative := a < 0
        if(isNegative){
                a = -a
        }
        i := uint64(math.Float64bits(a))
        mantissa := (i & 0xfffffffffffff)+0x10000000000000
        originalExponent := i>>52
        if originalExponent==0 {
                return 0
        }
        exponent := int64(originalExponent - 1023)
        if exponent>=0 {
                if exponent<11 {
                        mantissa = mantissa<<exponent
                } else {
                        fmt.Println("value too big")
                        return 0
                }
        } else {
                mantissa = mantissa>>(-exponent)
        }
        // truncate the given number of bits (numBits)
        var bitMask uint64 = getBitMask(numBits)
        mantissa = mantissa & bitMask
        if exponent>=0 {
                mantissa = mantissa>>exponent
        } else {
                mantissa = mantissa<<(-exponent)
        }
        truncated := originalExponent<<52 + mantissa & 0xfffffffffffff
        if isNegative {
                truncated = 1<<63 + truncated
        }
        return math.Float64frombits(truncated)
}
```

I truncated the floating numbers in net.outputWeights and net.hiddenWeights in the “train” section as well as in hiddenInputs, hiddenOutputs, finalInputs, and finalOutputs in the “predict” section.

The code snippet is as the following:

```
func (net *Network) Train(inputData []float64, targetData []float64) {
        ...
        truncateMatrix(net.outputWeights)
        truncateMatrix(net.hiddenWeights)
}

func (net Network) Predict(inputData []float64) mat.Matrix {
        // feedforward
        inputs := mat.NewDense(len(inputData), 1, inputData)
        hiddenInputs := dot(net.hiddenWeights, inputs)
        truncateMatrix((hiddenInputs).(*mat.Dense))
        hiddenOutputs := apply(sigmoid, hiddenInputs)
        truncateMatrix((hiddenOutputs).(*mat.Dense))
        finalInputs := dot(net.outputWeights, hiddenOutputs)
        truncateMatrix((finalInputs).(*mat.Dense))
        finalOutputs := apply(sigmoid, finalInputs)
        truncateMatrix((finalOutputs).(*mat.Dense))
        return finalOutputs
}
```

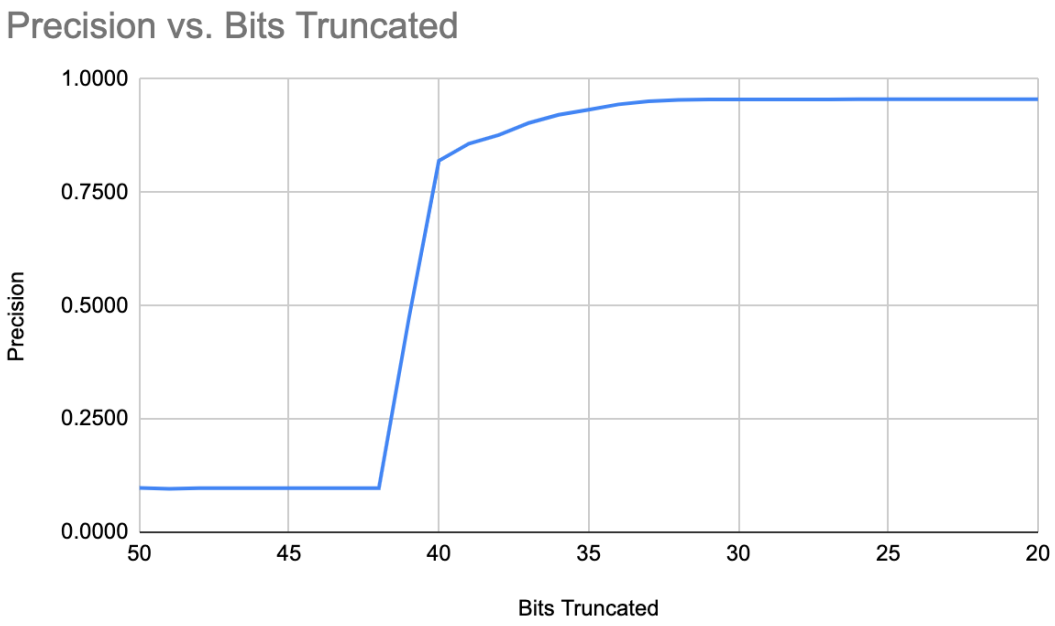
Results

I performed 2 tests with the ladder having additional truncation in the helper functions.

Test 2.1

Truncation in functions Train and Predict

Bits Truncated	Precision
20	0.9555
21	0.9555
22	0.9555
23	0.9555
24	0.9555
25	0.9555
26	0.9555
27	0.9554
28	0.9554
29	0.9552
30	0.9554
31	0.9550
32	0.9540
33	0.9514
34	0.9444
35	0.9324
36	0.9213
37	0.9033
38	0.8768
39	0.8577
40	0.8197
41	0.4720
42	0.0974
43	0.0974
44	0.0974
45	0.0974
46	0.0974
47	0.0974
48	0.0974
49	0.0958
50	0.0980



Test 2.2

Truncation in functions Train, Predict, as well as dot, apply, scales, multiply, add, subtract, max_weights, min_weights

The additional code snippet is as the following:

```
func dot(m, n mat.Matrix) mat.Matrix{
        ...
        truncateMatrix(o)
        return o
}
func apply(fn func(i, j int, v float64) float64, m mat.Matrix) mat.Matrix
func scale(s float64, m mat.Matrix) mat.Matrix
func multiply(m, n mat.Matrix) mat.Matrix
func add(m, n mat.Matrix) mat.Matrix
func subtract(m, n mat.Matrix) mat.Matrix
func max_weights(m, n mat.Matrix) *mat.Dense
func min_weights(m, n mat.Matrix) *mat.Dense
```

Bits Truncated	Precision
20	0.9555
21	0.9555
22	0.9555
23	0.9555
24	0.9555
25	0.9555
26	0.9555
27	0.9554
28	0.9553
29	0.9554
30	0.9554
31	0.9557
32	0.9543
33	0.9521
34	0.9474
35	0.9366
36	0.9254
37	0.9126
38	0.8917
39	0.8596
40	0.8381
41	0.6995
42	0.0974
43	0.0974
44	0.0974
45	0.0974
46	0.0974
47	0.0974
48	0.0974
49	0.0974
50	0.0974