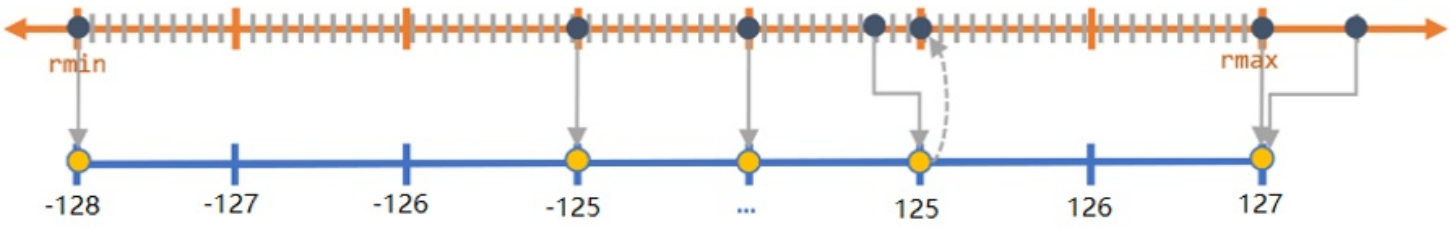
Quantization



Foreword

The purpose of this series is to describe in detail all aspects of the current mobile Int8, from the bottom layer of the Int8 assembly layer implementation principle and assembly performance optimization means, to the middle layer of the mobile framework supporting code implementation (standards take NCNN as an example), and the upper PC end quantitative methods (various papers thinking) corresponding to summarize and implementation, and the last model end method re-train, strategies and indicators introduction.

In addition, the use of PowerPerf (a debugging tool developed specifically for ARM CPU application performance optimization) to quantify the hardware parameters of the convolution kernel (assembly operator) is also the focus of this article, aiming to extract a common set The assembly code tuning methodology allows PowerPerf to be applied to all mobile CPU performance optimization scenarios.

Quantitative background

Although the size of the model is constantly compressed, the calculation amount is usually one or two hundred MFLOPS. This calculation is still a little difficult for the current (low-end) mobile CPU power, so the model side did The biggest effort, the mobile side should not be outdone to work hard!

Usually, the mobile-accelerated solution is divided into CPU and GPU. Currently, the mali GPU performance of the arm is poor on the low-end models, so the basic configuration is still the CPU solution, and the GPUs of the high-end machines are mostly Qualcomm. However, its overall performance is stronger than that of the CPU. Therefore, different solutions have different advantages on different positioning platforms, and each can be adapted according to a specific scenario.

To optimize on the CPU side, we can consider from top to bottom:

At the top is the algorithm layer , such as the use of winograd to mathematically reduce the number of multiplications (only valid in large channel sizes).
The framework implementation layer can implement policies such as memory pool and multi-threading;
The bottom layer is the hardware-related details: hardware architecture features, pipeline, cache, memory data rearrangement, NEON assembly optimization, and so on. . .
I have already discussed a part of the bottom layer, and I will continue to supplement it with Powerperf. I will discuss the quantization method first.

Lesson 2 - 9:
There are two optimizations on the Pre-Trained model explained.
1. Different Precisions (Quantization).

2. Fusing layers into fewer Operations.

Let me give some information about Quantization process,

Why is quantification useful?
Because CNN is not sensitive to noise.

Why use quantification?
The model is too big, for example, alexnet is 200MB, and the storage pressure is high, it must be lowered and cooled down;
The weights range of each layer is basically determined, and the fluctuation is not large, suitable for quantitative compression;
In addition, it not only reduces the number of acquisitions but also **reduces the amount of calculations.**

Why not train low precision models directly?
Because training needs backpropagation and gradient descent, int8 is very bad. For example, our learning rate is generally a few tenths and a few. How do you play an int8?

Secondly, everyone's ecology is a floating point model, so direct conversion is more effective!

Overview of quantitative methods:
There are many quantitative methods. For details, please see the following link. There are many papers in it, you can take a look at it slowly, and you have time to sort out these papers separately and summarize them.

Quantitative method summary link:
https://github.com/Ewenwan/MVision/tree/master/CNN/Deep_Compression/quantization

Principles of INT8 Quantization Algorithm:
INT8 quantization principle:
The simplest implementation at present is **NVIDIA's tensorRT solution**, which is directly quantified without retraining and is simple to implement;

The second is **Google's program, which is slightly more complicated and requires retrain;**

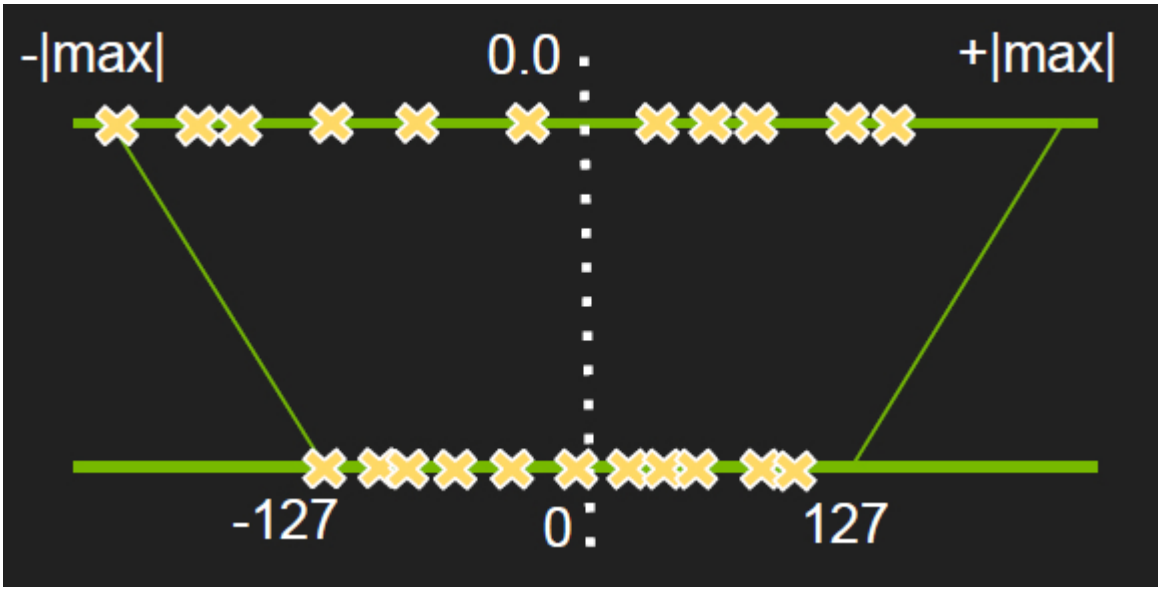
The requirement of retrain is that your weight and activation value (the actual measurement has little effect on the final accuracy) must be evenly distributed, that is, the variance should not be too large. The second is whether you can control the output of each layer within a certain range. This is very helpful when we do int8 quantization.

NVIDIA's solution is public, but it is not open source, which means that you can only use his set of tools (tensorRT) to quantify and deploy. Of course, it is normal, we also want to use his quantitative calibration part to get The calibration parameters are then used directly by the mobile terminal. It is currently found that these intermediate parameters are not available, and there is no source code. The python excuses in the installation package are also called so files; the ppt links they give are as follows:
http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf

Our goal is to convert the original float 32bit convolution operation (multiply and add instruction) into an int8 convolution operation, so that the calculation becomes the **original 1/4.** but the memory is not reduced, because

we are In the kernel, float32 is changed to int8 for calculation.

The roughest, most essential principle is this picture:



It's very simple! It is to circle the activation value range of a layer of your layer, and then **use the maximum value of the absolute value as the threshold** (so when the positive and negative distribution is uneven, part of it is vacant, that is, part of the value range is wasted; There is a small pit here. If my activation address is all positive and there is no negative value, how do you map it?), and **then directly map this range to the range of plus or minus 128.** The formula is as follows:

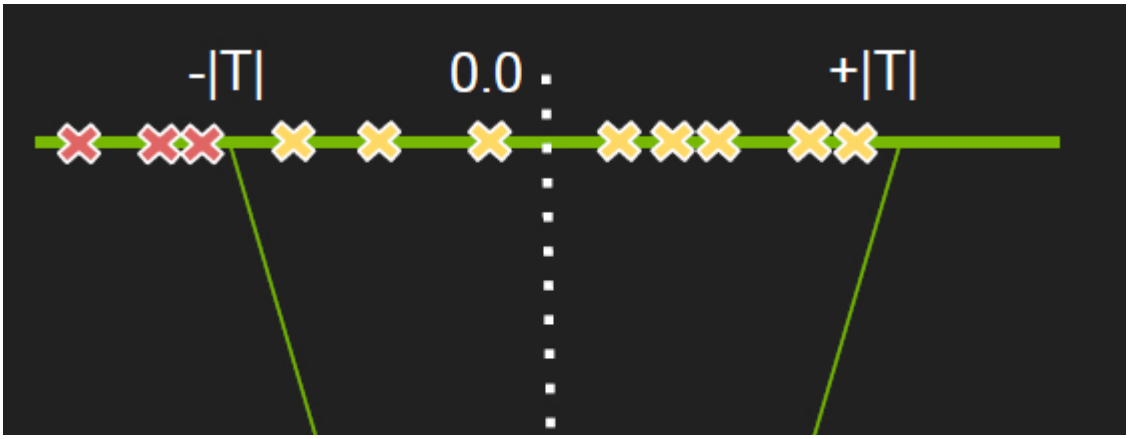
FP32 Tensor (T) = scale_factor(sf) * 8-bit Tensor(t) + FP32_bias (b)

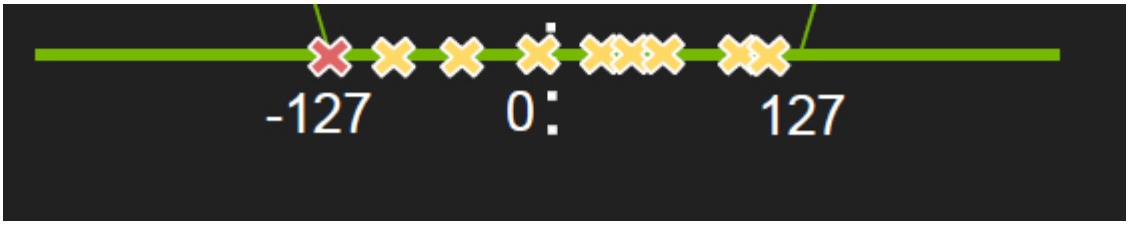
It is known through experiments (NVIDIA said that it has not been verified ~) , the effect of the bias value removed on the accuracy is not very large, so we directly remove:

T = sf * t

The above is a **simple max-max mapping**, which is for **even distribution**. Obviously, as long as the data distribution is not very uniform, the precision loss is very obvious, so in many cases it is:

Let's take a rough look at this picture:





Why is quantification to guarantee the original information?
This reason is like the difference between a high-definition picture and a low-resolution picture. As long as your goal is to generally identify the information in the picture, a low-resolution picture is allowed.

Do you see the online video added to the mosaic will not affect your judgment? No, you will make extra details in the brain. Only when the full-screen mosaic will affect your viewing experience, so this code, no, quantification is actually a degree of problem, one can you The extent of acceptance.

Why does the maximum mapping result in a serious loss of precision ?
You see the distribution of values, because the positive and negative distribution is very uneven, if you map according to the symmetrical maximum value (intended to retain the original information as much as possible), then there is an area on the +max side that is wasted, that is, scale After int8, the dynamic range of int8 is even smaller.
The example of the limit is that after the quantization, the dynamic range of the original int8 is only 1 bit left (that is, the positive samples are not, and the negative ones are all piled up near a small value). This is the full screen mosaic mentioned above~ in this case. . . That also indicates the original information of Mao!

Why is the saturation intercept on the right ok?
Because the problem of unsaturated interception is that when the data distribution is extremely uneven, there are many dynamic ranges that are wasted, that is to say, the mosaic is very large! And saturation interception is to make up for this problem.

When your data distribution is very uneven, as the left side of the figure is more than the right side, then I cut off the original information before mapping, and then form a symmetrical and well-distributed truncation information, and then map this information to int8, then There will be no dynamic range resources being wasted, that is to say, the mosaic is more delicate~ You can estimate the brain to make up the details.

As shown above, first find a threshold T, then all below the minimum threshold are saturated mapped to -127, the three red points on the left side of the above figure are treated as such.

(This is also a very natural idea. Right? Remove the unrelated high-frequency details to get the performance benefits! Network image compression technology is not so complete! PCA principal component, Fourier decomposition ideas are not all This way! Grab the main contradiction of things, ignore the details, and improve the overall performance! Just like the regularization optimization in machine learning is not the case, avoiding you too much into the details to produce over-fitting! In fact, isn't our life the same? Anything has to be ruthless, what about the horns?!! Sometimes giving up some things, first of all, your life will definitely be a lot easier, and secondly, you will get a more stable life happiness value. (generalization performance)!)

Our problem is converted to how to find the optimal threshold T to minimise the loss of precision?

First of all, Naturally, you will want to describe it with a model! Then we have to model first. **Build a model to**

First of all, Naturally, you will want to describe it with a model! Then we have to model first, **Build a model to evaluate the loss of precision before and after quantization, and then minimise the loss.**

The so-called model is a function in the high number! For example, the ellipse formula is a description of an elliptical model, and the parameters inside are the values required for our specific task.

The next layer is how do we design this model function?
I can't think of how to describe this optimization problem in mathematics! After all, my mathematics application ability is still at the level of the application questions in the exam! In the end, it is the foundation and understanding of mathematics!

However, this can not stop my yearning for mathematics and pursuit!

When you experience how mathematically describes a complex thing, you will experience her wonders~ I can't help but admire, wonderful!

Let's learn the NVIDIA ideas!

NVIDIA chose **KL-divergence , which is actually relative entropy.**
Why choose relative entropy? And nothing else? Because the **relative entropy expresses the degree of difference between the two distributions,** it is the degree of difference between the two distributions before and after quantification in our situation. **The smallest difference is the best.** So the problem is converted to the minimum value of relative entropy.

This is the expression of the amount of information entropy:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

That is, the expectation of log(1/p), that is, the expectation of the reciprocal of the probability of occurrence of the event ;

What is Information Entropy:
The concept of entropy was first introduced by statistical thermodynamics.

Information entropy is proposed by Shannon, the father of information theory. **It is used for the uncertainty measure of random variables, first on the formula of information entropy.**

Information is used to reduce random uncertainty (ie, the reduction in uncertainty).
We can use log (1/P) to measure uncertainty. P is the probability that something will happen. The greater the probability, the smaller the uncertainty.

It can be seen that the formula of information entropy is actually the expectation of log (1/P), which is the expectation of uncertainty. It represents the uncertainty of a system. The larger the information entropy, the greater the uncertainty.

The entropy of information entropy in the joint generalized probability distribution

When X, Y are independent of each other, H(X, Y) = H(X) + H(Y)

When X and Y are not independent, I(X, Y) = H(X) + H(Y) - H(X, Y) can be used to measure the correlation of the two distributions. This definition is rarely used.

Example explanation of information entropy:

The example is familiar, I think it is quite good.

For example, in a betting horse game, there are 4 horses { A, B, C, D}, and the winning odds are { 1/2, 1/4, 1/8, 1/8 }, respectively, which horse wins as random The variable X belongs to { A, B, C, D } .

Suppose we need to determine the value of the random variable X with as few binary problems as possible.
For example, question 1: Did A win? Question 2: Has B won? Question 3: Has C won?
Finally, we can determine the value by using up to 3 binary questions.
If X = A, then you need to ask 1 time (question 1: Is it A?), the probability is 1/2
If X = B, then you need to ask 2 times (question 1: Is it A? Question 2: Is it B?), the probability is 1/4
If X = C, then you need to ask 3 times (question 1, question 2, question 3), the probability is 1/8

If X = D, then you need to ask 3 times (question 1, question 2, question 3), the probability is 1/8

Then the number of binary problems to determine the value of X is
Going back to the definition of information entropy, we will find that through the previous information entropy formula, we magically got:
In a binary computer, a bit of 0 or 1 actually represents an answer to a binary question. That is to say, in the computer, we encode which event to win the horse, and the average code length required is 1.75 bits.
Obviously, in order to minimize the code length, we give the probability of occurrence of
$p(x)$
large events, assign a short code length
$l(x)$
. This issue is discussed in depth and the concept of Huffman coding can be derived.

Huffman coding is the idea of using this large probability event to assign short codes, and it can be proved that this coding method is optimal. We can prove the above phenomenon:

- In order to obtain the information entropy is

$H(X)$

- a random variable of

X

- a sample, on average, even throwing a coin (or dibasic problem)

$H(X)$

- times (PORTFOLIO guess the race issue)
- Information entropy is a critical value of data compression (in the case of the reference code length part)

Therefore, the information entropy H(X) can be seen as the expected value of the code length required to encode the samples in X.

3 Cross entropy and KL divergence

The previous section stated that the information entropy H(X) can be seen as the expected value of the encoding length required to encode the samples in X.

Here we can understand the understanding of cross entropy. Now there are two distributions, the real distribution p and the non-real distribution q, and our sample comes from the real distribution p.

The expectation of the code length required to encode the sample according to the true distribution p is

, this is the **information entropy H(p) described** above **.**

The expectation of the length of the code required to encode the sample according to the unrealistic distribution q

is the so-called **cross entropy H(p,q)**

Here we derive the **KL divergence D(p||q)** = H(p,q) - H(p) =

, also called **relative entropy** , which represents the difference between the two distributions. The larger the difference, the larger the relative entropy.

In machine learning, we use the non-real distribution q to predict the true distribution p, because the real distribution p is fixed, and D(p||q) = H(p,q) - H(p) is fixed in H(p), That is to say, the larger the cross entropy H(p, q), the larger the relative entropy D(p||q), the greater the difference between the two distributions.

So cross entropy is used to make the loss function. This is the reason that measures the difference between the real distribution and the predicted distribution.

Entropy discussion:
https://www.zhihu.com/question/22178202

https://www.zhihu.com/question/41252833

The greater the entropy , the **less likely the event is to occur** , that **is, the impossibility is large** , so the amount of information contained in it is greater. For example, the recent World Cup Messi out, this is a low probability event in everyone’ s eyes, so this When things happened, there were a lot of people who said that they wanted to go to the rooftop! Is there no insider in the end? Thoughtful. . . . Um~ You still say that the amount of information about this matter is not big? ! ! !

The explanation in the information theory is that the expectation of the coding length required to encode the sample according to the real distribution p is as follows, which is the **information entropy** H(p).

$$\sum_i p(i) * log\frac{1}{p(i)}$$

The expectation of the code length required to encode the sample according to the unrealistic distribution q is as follows, which is called the **cross entropy** H(p,q)

$$\sum_i p(i) * log\frac{1}{q(i)}$$

Here we derive the KL divergence D(p||q) = H(p,q) - H(p) = the formula below, also called **relative entropy** , which represents the difference between the two distributions. The greater the difference, the more the relative entropy Big.

$$\sum_i p(i) * log\frac{p(i)}{q(i)}$$

The above formula is expanded like this:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot (\log p(x_i) - \log q(x_i))$$

The beginning of entropy is like this! Then in the coding theory, the bottom of the log is changed to 2, and the number of bits is out! In other words, how many bits (expected value) I need to describe the amount of information in a piece of information!

From the perspective of coding, the relative entropy, that is, what is KL-divergence? And why use KL-divergence?

Suppose we have a series of symbols that know the probability of their appearance. If I want to optimally encode these symbols, I will use T bits to represent them. T is the optimal bit number for the original information. We call this code A;

Now we have the same set of symbols, but the probability of their appearance has changed. If I still use A code to encode this set of matches, then the bit length T' of the code is suboptimal, which is greater than the original T value. of.

(Assuming I have a series of symbols, I know the probability of them happening. If I want to optimally encode these symbols, I will use "T" to indicate. Note that T is the optimal number of bits. Let's put this The segment code is called "A". Now, I have the same set of symbols but the probability of their occurrence has changed. Now, the symbol has a new probability. If I use code A to encode the symbol, the number of bits encoded will be Suboptimal, greater than T.)

KL divergence is to accurately measure the difference between this optimal and suboptimal (due to the choice of the wrong code). Here F32 is the original optimal encoding, int8 is the suboptimal encoding, we use KL divergence to describe the difference between the two encodings.

Relative entropy indicates how many bits you need to encode when using suboptimal encoding, that is, the bit difference between the optimal encoding and the optimal encoding.
The cross entropy indicates how many bits are exactly needed when you use suboptimal coding.
Therefore, the bits required for optimal coding = cross entropy - relative entropy.
Of course, the author who knows the above is much more popular. You can go and see it~

Let's write another code to verify the principle:

The process of the following code is to generate two random distributions, the value is between 1 and 11, take 10 values, and then calculate the KL value between the two distributions. When KL is lower than 0.1, we look at its true distribution. Come and feel it.

```
def get_KL():
# 随机生成两个离散型分布
x  = [np.random.uniform(1, 11) for i in range(10)]
px = x / np.sum(x)
y  = [np.random.uniform(1, 11) for i in range(10)]
py = y / np.sum(y)

KL = 0.0
for i in range(10):
    KL += px[i] * np.log(px[i] / py[i])
```