Quantization steps

trace code順序：

1	ristretto.cpp	quantize()
2	quantization.cpp	QuantizeNet() RunForwardBatches()
3	net.cpp	RangeInLayers()
4	quantization.cpp	Quantize2DynamicFixedPoint() EditNetDescriptionDynamicFixedPoint() Quantize2MiniFloat() EditNetDescriptionMiniFloat() Quantize2IntegerPowerOf2Weights() EditNetDescriptionIntegerPowerOf2Weights()
5	solver.cpp	Step()
6	net.hpp	ForwardBackward()
7	net.cpp	Forward(Dtype* loss) ForwardFromTo(int start, int end)
8	layer.hpp	Forward()
9	conv_ristretto_layer.cpp	Forward_cpu()
10	base_ristretto_layer.cpp	QuantizeLayerInputs_cpu() QuantizeWeights_cpu() QuantizeLayerOutputs_cpu() Trim2FixedPoint_cpu() Trim2MiniFloat_cpu() Trim2IntegerPowerOf2_cpu()
11	sgd_solver.cpp	ApplyUpdate()

Dynamic Fixed Point

1. 從command line得到各種資料，用來initialize Quantization
2. 複製model和weight，net_val，使用test data測試得到basline accuracy

3. 複製model和weight，net_test，使用train data來找出con.,ip. layer的index(layer_names)，並算出該layer的in、out、param的最大值(max_in、max_out、mxa_params)
   此處param是weights
4. 對max_in(_out、_params)取log2，可以得到integer length(所需bit數)
5. 從model讀出parameter，並設定成test data
6. 對所有的con. layer的weight找出fl，從16 bit width開始直到accuracy降到error margin之外
   a. 每跑完一種bit width，會把accuracy儲存起來
   b. 每個con. layer會用同一個bit width做測試
7. 對所有的ip. layer的weight找出fl，從16 bit width開始直到accuracy降到error margin之外
   a. 每個ip. layer會用同一個bit width做測試
8. 對所有的con.、ip. layer的in、out找出fl，從16 bit width開始直到accuracy降到error margin之外
   a. 每個layer的activition都用同一個bit width做測試
   b. 6.~8.不會同時Quantize，一次只會對某個layer的某個part做Quantize後算出accuracy
9. 將在error margin以內的accuracy，選出最小bit width
   a. bw_con、bw_fc、bw_out(bw_in = bw_out)
   b. 每個con. layer都是用bw_con、ip. layer用bw_fc、兩者都使用bw_in、bw_out
10. 從model讀出parameter，並設定成test data
11. 把bw_con、bw_fc、bw_out、bw_in放入EditNetDescriptionDynamicFixedPoint()，得到每個layer part都做過Quantize的net
12. 在開始training之後，Forwarding的過程中，再做運算之前，會先把input、output、weight做quantize(Trim2FixedPoint_cpu)才開始該次forward

```cpp
template <typename Dtype>
void BaseRistrettoLayer<Dtype>::Trim2FixedPoint_cpu(Dtype* data, const int cnt,
      const int bit_width, const int rounding, int fl) {
  for (int index = 0; index < cnt; ++index) {
    // Saturate data
    Dtype max_data = (pow(2, bit_width - 1) - 1) * pow(2, -fl);
    Dtype min_data = -pow(2, bit_width - 1) * pow(2, -fl);
    data[index] = std::max(std::min(data[index], max_data), min_data);
    // Round data
    data[index] /= pow(2, -fl);
    switch (rounding) {
    case QuantizationParameter_Rounding_NEAREST:
      data[index] = round(data[index]);
      break;
    case QuantizationParameter_Rounding_STOCHASTIC:
      data[index] = floor(data[index] + RandUniform_cpu());
      break;
    default:
      break;
    }
    data[index] *= pow(2, -fl);
  }
}
```

a. type仍然是32bit，但Data的value已經被改為bw_con、bw_fc、bw_out、bw_in可以表示的範圍

MiniFloat

1. 接續DFP步驟3.
2. 對max_in、max_out做兩次取log2，第一次是得到需要n次方，第二次是得到表達n次方所需的bit width，得exp_in、exp_out。接著取兩者間最大的作為exp_bit。
3. 換下一個layer做，得到新的exp_in、exp_out，若是有更大的值就更新exp_bit
    a. 因為weight通常小於activation，所以不另外計算
    b. 所有layer都用一樣長度的exp_bit**(?)**
4. 從model讀出parameter，並設定成test data
5. 對所有layer的activition，從16 bit width開始測試直到accuracy降到error margin之外，每一次bitwidth的結果都會被記錄下來
    a. input、output、weight都用同一個bit width
6. 在error margin之內選出最小的bit width，作為quantize net的結果

```
template <typename Dtype>
void BaseRistrettoLayer<Dtype>::Trim2MiniFloat_cpu(Dtype* data, const int cnt,
      const int bw_mant, const int bw_exp, const int rounding) {
  for (int index = 0; index < cnt; ++index) {
    int bias_out = pow(2, bw_exp - 1) - 1;
    float_cast d2;
    // This casts the input to single precision
    d2.d = (float)data[index];
    int exponent=d2.parts.exponent - 127 + bias_out;
    double mantisa = d2.parts.mantisa;
    // Special case: input is zero or denormalized number
    if (d2.parts.exponent == 0) {
      data[index] = 0;
      return;
    }
    // Special case: denormalized number as output
    if (exponent < 0) {
      data[index] = 0;
      return;
    }
```

```
    // Special case: denormalized number as output
    if (exponent < 0) {
      data[index] = 0;
      return;
    }
    // Saturation: input float is larger than maximum output float
    int max_exp = pow(2, bw_exp) - 1;
    int max_mant = pow(2, bw_mant) - 1;
    if (exponent > max_exp) {
      exponent = max_exp;
      mantisa = max_mant;
    } else {
      // Convert mantissa from long format to short one. Cut off LSBs.
      double tmp = mantisa / pow(2, 23 - bw_mant);
      switch (rounding) {
      case QuantizationParameter_Rounding_NEAREST:
        mantisa = round(tmp);
        break;
      case QuantizationParameter_Rounding_STOCHASTIC:
        mantisa = floor(tmp + RandUniform_cpu());
        break;
      default:
        break;
      }
    }
    // Assemble result
    data[index] = pow(-1, d2.parts.sign) * ((mantisa + pow(2, bw_mant)) /
        pow(2, bw_mant)) * pow(2, exponent - bias_out);
  }
}
```

INTEGER_POWER_OF_2_WEIGHTS

1. weights直接指定exp_min = -8、exp_max = 1
2. Activations使用dynamix fix point， 用bitwitdh=8

```
template <typename Dtype>
void BaseRistrettoLayer<Dtype>::Trim2IntegerPowerOf2_cpu(Dtype* data,
      const int cnt, const int min_exp, const int max_exp, const int rounding) {
  for (int index = 0; index < cnt; ++index) {
    float exponent = log2f((float)fabs(data[index]));
    int sign = data[index] >= 0 ? 1 : -1;
    switch (rounding) {
    case QuantizationParameter_Rounding_NEAREST:
      exponent = round(exponent);
      break;
    case QuantizationParameter_Rounding_STOCHASTIC:
      exponent = floorf(exponent + RandUniform_cpu());
      break;
    default:
      break;
    }
    exponent = std::max(std::min(exponent, (float)max_exp), (float)min_exp);
    data[index] = sign * pow(2, exponent);
  }
}
```