Evaluation of computational complexity for basic operators in L1 Transformer inference

ETH Zürich, NextRAN-AI

Marco Bertuletti mbertuletti@iis.ee.ethz.ch

Matmul

Pseudocode:

```
for (uint32_t il = 0; il < loop; il++) {
  for (uint32_t im = 0; im < m; im++) {
    for (uint32_t in = 0; in < n; in++) {
      for (uint32_t ip = 0; ip < p; ip++) {
        Y[il][im][ip] += X1[il][im][in] * X2[il][in][ip];
      }
    }
  }
}
```

Input Data	X1[loop][m][n]
Input Data	X2[loop][n][p]
Output Data	Y[loop][m][p]
FLOPS	loop*m*n*p
Trainable Param.s	loop*n*p
Memory Footprint	loop*(m*n + n*p + m*p)

Tensor addition

Pseudocode:

```
for (uint32_t il = 0; il < loop; il++) {
  for (uint32_t im = 0; im < m; im++) {
    for (uint32_t in = 0; in < n; in++) {
      Y[il][im][in] = X1[il][im][in] + X2[il][im][in];
    }
  }
}
```

Input Data	X1[loop][m][n]
Input Data	X2[loop][m][n]
Output Data	Y[loop][m][n]
FLOPS	loop*m*n
Trainable Param.s	0
Memory Footprint	loop*(2*m*n)

Constant multiplication tensor

Pseudocode:

```
for (uint32_t il = 0; il < loop; il++) {
  for (uint32_t im = 0; im < m; im++) {
      for (uint32_t in = 0; in < n; in++) {
      Y[il][im][in] = X[il][im][in] * C;
      }
  }
}
```

Input Data	X[loop][m][n]
Output Data	Y[loop][m][n]
FLOPS	loop*m*n
Trainable Param.s	1
Memory Footprint	loop*m*n

Energy normalization

```
for (uint32_t il = 0; il < loop; il++) {
  float sum = 0.0;
  for (uint32_t im = 0; im < m; im++) {
      sum += X[il][im]*X[il][im];
  }
  for (uint32_t im = 0; im < m; im++) {
      Y[il][im] = X[il][im] / sqrt( sum + epsilon );
  }
}
```

Input Data	X[loop][m]
Output Data	Y[loop][m]
FLOPS	loop*(m + 3*m)
Trainable Param.s	loop
Memory Footprint	loop*m

Layer normalization

```
for (uint32_t il = 0; il < loop; il++) {
  float mean = 0.0;
  float var = 0.0;
  for (uint32_t im = 0; im < m; im++) {
    mean += X[il][im];
  }
  mean = mean / m;
  for (uint32_t im = 0; im < m; im++) {
    var += (X[il][im] - mean)^2;
  }
  var = var / m;
  for (uint32_t im = 0; im < m; im++) {
    Y[il][im] = ((X[il][im]-mean)*gamma / sqrt(var + epsilon)) + beta;
  }
}
```

Input Data	X[loop][m]
Output Data	Y[loop][m]
FLOPS	loop*(m + 1 + 2*m + 1 + 6*m)
Trainable Param.s	3*loop
Memory Footprint	loop*m

Softmax

```
for (uint32_t il = 0; il < loop; il++) {
  float sum = 0.0;
  for (uint32_t im = 0; im < m; im++) {
      sum += exp(X[il][im]);
  }
  for (uint32_t im = 0; im < m; im++) {
      Y[il][im] = exp(X[il][im]) / sum;
  }
}
```

Input Data	X[loop][m]
Output Data	Y[loop][m]
FLOPS	loop*(2*m + 2*m)
Trainable Param.s	0
Memory Footprint	loop*m

Gelu

Pseudocode:

```
for (uint32_t il = 0; il < loop; il++) {
  for (uint32_t im = 0; im < m; im++) {
      Y[loop][m] = 0.5*(1+tanh(const1*(X[il][im]+const2*X[il][im]^3));
  }
}
```

Input Data	X[loop][m]
Output Data	Y[loop][m]
FLOPS	5
Trainable Param.s	0
Memory Footprint	loop*m

Conv1D

Pseudocode:

```
for (uint32_t l = 0; l < loop; l++) {
  for (uint32_t i = 0; i < Cout; i++) {
    for (uint32_t j = 0; j < Win; j++) {

      for (uint32_t k = 0; k < Cin; k++) {
        for (uint32_t f = 0; f < Wf; f++) {
           Y[l][i][j] += X[l][k][j - Wf/2 + f] * F[i][k][f];
        }
      }

    }
  }
}
```

Input Data	X[loop][Cin][Win]
Input Data	F[Cout][Cin][Wf]
Output Data	Y[loop][Cout][Wout]
FLOPS	(Wf*Cin)*Win*Cout*loop
Trainable Param.s	Cout*Cin*Wf
Memory Footprint	Cout*Cin*Wf + loop*(Cin*Win + Cout*Wout)