Crypto + NN Programing Report

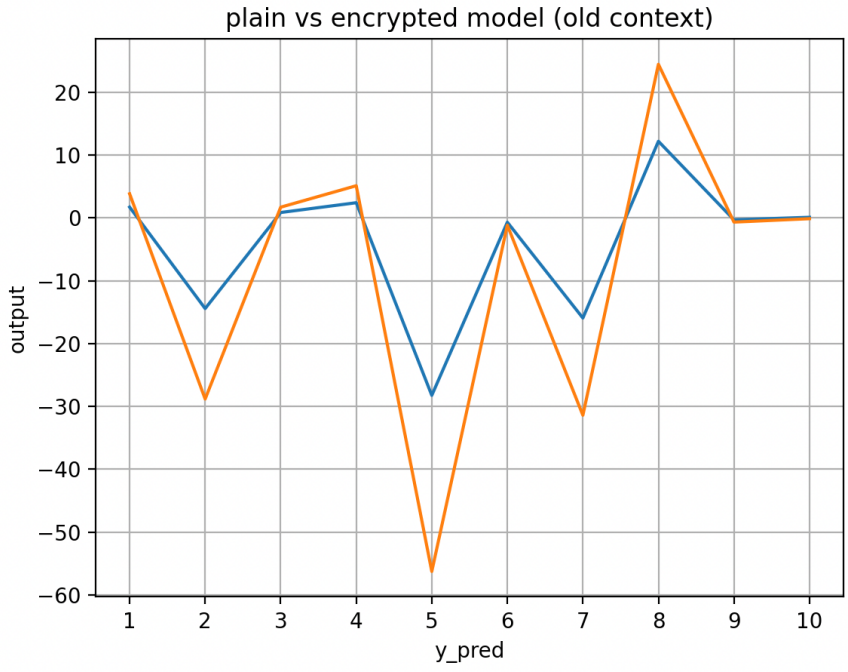
R11922138 蕭彧

Q1:

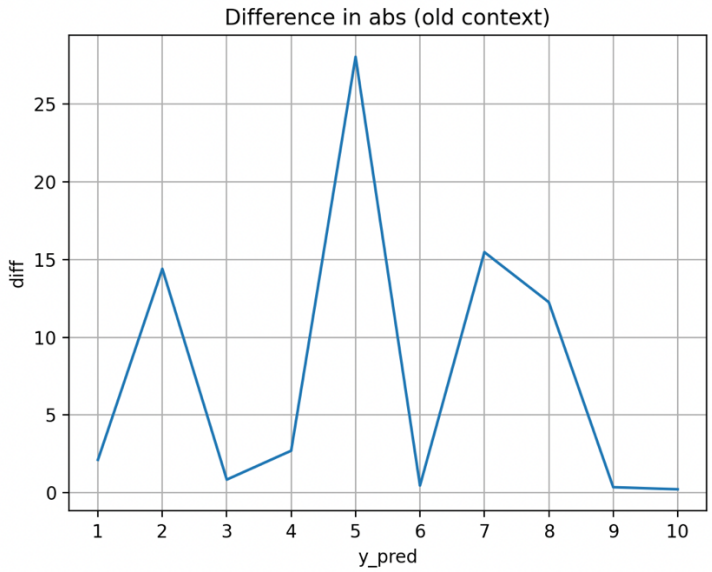
```python
class Encrypted_MLP(nn.Module):
    def __init__(self, input_dim, output_dim, weight):
        super().__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.weight = weight
        self.input_weight = ts.ckks_tensor(context, self.weight['input_fc.weight'])
        self.input_bias = ts.ckks_tensor(context, self.weight['input_fc.bias'])
        self.output_weight = ts.ckks_tensor(context, self.weight['output_fc.weight'])
        self.output_bias = ts.ckks_tensor(context, self.weight['output_fc.bias'])
    def forward(self, x):
        hidden = self.input_weight.mm(x.reshape(-1,1)).transpose() + self.input_bias
        hidden = hidden * hidden # activation function
        output = hidden.mm(self.output_weight.transpose()) + self.output_bias
        return output
```

Above is my encrypted MLP model. We only need to encrypt model weight. Tenseal only support C2P encrypt, while the original equation of the first layer: y= x*A^T + b (x is plain input, A is encrypted weight matrix, b is encrypted bias) is P2C. So, we can't simply encrypted the weight and use nn.linear to get the encrypted output. Therefore, I modify the equation to: y = (A*x^T)^T + b. By this equation, all the operation is rather C2P or C2C.

Result:



Diff:



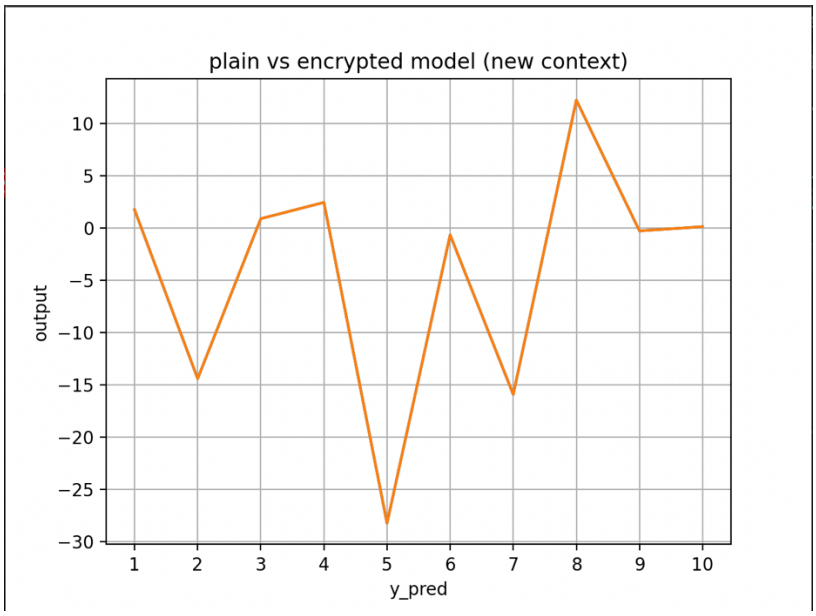
In the first figure we can see the results of two models is very different.
The difference between output of plain model and encrypted model is extremely large. It is about 50% of bias.

Q2:

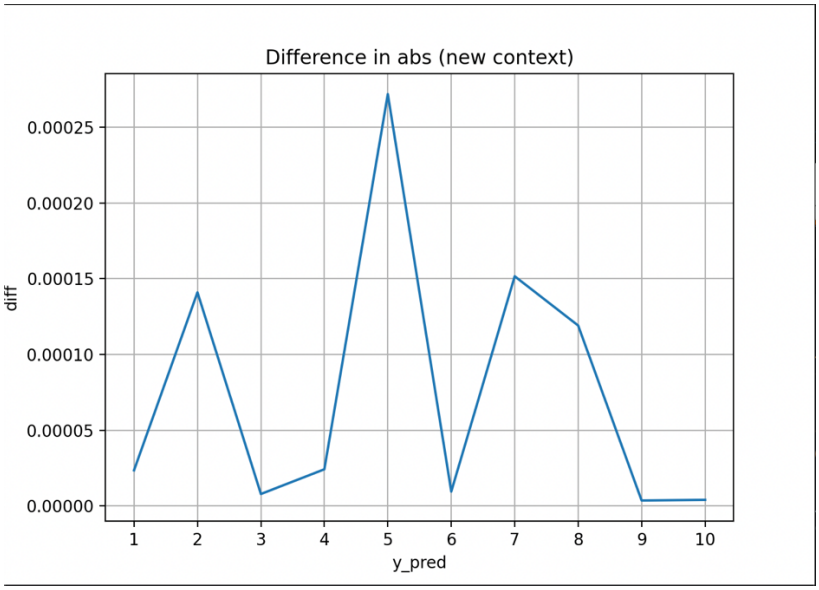
```
context = ts.context(
            ts.SCHEME_TYPE.CKKS,
            poly_modulus_degree=16384,
            coeff_mod_bit_sizes=[60, 40, 40, 40, 60]
          )
context.global_scale = 2**40
```

Above is my new context. I have tested a lot of parameters. The coeff_mod_bit_sizes control the depth of multiplication. The poly_modulus_degree control the number of coefficients in plaintext polynomials. The global_scale control the accuracy of floating points. The key parameter to minimize the diff is global scale. As long as your global scale is big enough, you won't lose a lot of accuracy while computing. I also tune the global_scale to 2^30, but the diff is as large as origin.

Result:



Diff:



The result of two model is so close so that the first figure looks like there's only one line but in fact there are two overlapped lines. In the second figure we can see that the difference is relatively small compared to the old context.