Guides

AntChain-MPC是一个隐私保护计算系统。 其中的morse-stf模块是一个基于tensorflow的隐私计算模块，它可以在保护用户私有数据的前提下对多方数据进行联合计算。包括算术运算，逻辑运算，序运算等基础运算，以及基于基础运算功能之上的机器学习。

网络拓扑

网络拓扑请见Tutorials。

安装

安装步骤请见Tutorials。

运行例

下面列出AntChain-MPC-STF支持的一些基于MPC运行的任务示例：

双方私有矩阵乘法

以三方协议运行双方私有矩阵乘法请见Tutorials。下面我们讲述以双方协议运行的方式，该示例收录在`morse-stf/examples/private_matmul_parties2.py`

Local运行

Step 1. 建立一个工作目录 morse-stf ，并将开源代码中conf, examples, dataset目录copy到morse-stf下。

Step 2. 然后配置好`morse-stf\conf\config_parties2.json`文件:

JSON 复制代码

```json
{
    "parties": 2,
    "pre_produce_flag": true,
    "offline_model": false,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }
}
```

其中，`pre_produce_flag` 字段用来指示双方协议运行的两种方式：
当`pre_produce_flag` 设置为 false时，将不进行预处理，而是使用使用纯online模式，即每一次乘法运算（包括标量乘法，向量乘法，矩阵乘法）都在online用同态加密进行计算。
当`pre_produce_flag` 设置为 true时，将使用预处理：其中当`offline_model`设置为 true时，程序运行于offline model下，将不读取用户数据，仅利用同态加密进行一些预处理，并将预处理得到的伪随机数据存储于当前目录的serialize子目录下；当`offline_model`设置为 false时，程序运行于online model下，将读取用户数据，并利用offline model下预处理的伪随机数，运行MPC协议以进行隐私计算。

`hosts`字段需要配置两台机器的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。
`default_fixed_point` 字段表示默认的定点位置。
其余字段在本例中没有用途。

下面分为a, b两条路线:

路线a:

Step 3a. 当`pre_produce_flag=false`时，于`morse-stf/examples`目录下运行`python3 private_matmul_parties2.py`

路线b:
Step 3b. 当`pre_produce_flag=true`时，首先置`morse-stf\conf\config_parties2.json`文件中的`offline_model=true`，于`morse-stf/examples`目录下运行`python3 private_matmul_parties2.py`。此时程序会进行预处理。

Step 4b. 待预处理完毕后，再置`morse-stf\conf\config_parties2.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 private_matmul_parties2.py`。此时程序会进行双方私有矩阵乘法的安全计算，并输出运算结果。

Distributed运行

Step 1. 在参与计算的两台机器（以下称为workerL和workerR）上分别建立工作目录 morse-stf，并将开源代码中conf, examples, dataset目录分别copy到两台机器的morse-stf下。
Step 2. 在workerL, workerR 上分别配置好`morse-stf\conf\config_parties2.json`文件:

JSON 复制代码

```json
{
    "parties": 2,
    "pre_produce_flag": true,
    "offline_model": true,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }
}
```

其中，`morse-stf\conf\config_parties2.json`文件中的hosts字段需要配置两台机器的ip:host，而stf_home_workerL, stf_home_workerR字段需要分别填写两台机器上morse-stf工作目录的绝对路径。

当`pre_produce_flag` 设置为 false时，将不进行预处理，而是使用使用纯online模式，即每一次乘法运算（包括标量乘法，向量乘法，矩阵乘法）都在online用同态加密进行计算。
当`pre_produce_flag` 设置为 true时，将使用预处理：其中当`offline_model`设置为 true时，程序运行于offline model下，将不读取用户数据，仅利用同态加密进行一些预处理，并将预处理得到的伪随机数据存储于当前目录的serialize子目录下；当`offline_model`设置为 false时，程序运行于online model下，将读取用户数据，并利用offline model下预处理的伪随机数，运行MPC协议以进行隐私计算。

`default_fixed_point` 字段表示默认的定点位置。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR 上启动服务。

```
Plain Text 复制代码
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config_parties2.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config_parties2.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改`morse-stf/example/private_matmul_parties2.py` 文件，将
代码中的

```
Plain Text 复制代码
start_local_server(config_file="../conf/config_parties2.json")
```

修改为

```
Plain Text 复制代码
start_clinet(config_file="../conf/config_parties2.json", job_name="workerR")
```

> 下面分为a, b两条路线，无论那种路线，都只需要在workerR上进行:

路线a:
Step 5a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 private_matmul_parties2.py`

路线b:
Step 5b. 当`pre_produce_flag=true` 时，首先置`morse-stf/conf/config_parties2.json` 文件中的`offline_model=true`，于`morse-stf/examples` 目录下运行`python3 private_matmul_parties2.py` 。 此时程序会进行预处理。

Step 6b. 待预处理完毕后，再置`morse-stf/conf/config_parties2.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 private_matmul_parties2.py`。此时程序会进行双方私有矩阵乘法的安全计算，并输出运算结果。

单方特征的LR模型训练预测

三方协议

Local运行

下面给出一个在本机开三端口利用stf进行逻辑回归模型训练\预测的例子，在本例中，L方有特征，R方有label，假设他们已经将数据对齐。

Step1. 建立一个工作目录morse-stf，将morse-stf源代码中 conf，examples，dataset子目录copy到morse-stf工作目录下
Step2. 配置morse-stf/conf/config.json文件

JSON 复制代码

```json
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/embed_op_fea_5w_format_x_train.csv",
            "R": "dataset/embed_op_fea_5w_format_y_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/embed_op_fea_5w_format_x_test.csv",
            "R": "dataset/embed_op_fea_5w_format_y_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}

```

其中，`hosts`字段需要配置本机的三个空闲的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。
`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step3. 于morse-stf/examples目录下运行python3 lr_train_and_predict.py

Distribute运行

下面给出一个利用三台机器（以下称为workerL，workerR和RS）进行逻辑回归模型训练、预测的例子。在本例中，workerL上有特征，workerR上有label，且假设数据已经对齐。RS为Random Servies（伪随机

数服务器)，它只负责产生伪随机数，并发送给workerL和workerR，并不从workerL和workerR上接收数据。

Step 1. 在参与计算的三台机器上分别建立工作目录 morse-stf ，并将开源代码中conf, examples, dataset目录分别copy到三台机器的morse-stf下。
Step 2. 在workerL, workerR, RS上分别配置好`morse-stf\conf\config_ym.json`文件:

JSON 复制代码

```
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_RS": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/embed_op_fea_5w_format_x_train.csv",
            "R": "dataset/embed_op_fea_5w_format_y_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/embed_op_fea_5w_format_x_test.csv",
            "R": "dataset/embed_op_fea_5w_format_y_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}
```

其中，`hosts`字段需要配置三台机器的ip:host，而stf_home_workerL, stf_home_workerR, stf_home_RS字段需要分别填写三台机器上morse-stf工作目录的绝对路径，`ml`字段下的所有路径填写morse-stf下的相对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。

`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR, RS 上启动服务。

```
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config_ym.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config_ym.json
(在RS上上的morse-stf目录下运行)    morse-stf-server --player=RS --config_file=.\conf\config_ym.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改`morse-stf/examples/lr_train_and_predict.py` 文件，将
代码中的

```
start_local_server(config_file="../conf/config_ym.json")
```

修改为

```
start_clinet(config_file="../conf/config_ym.json", job_name="workerR")
```

> 下面分为a, b两条路线，无论那种路线，都只需要在workerR上进行:

路线a:
Step 5a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict.py`

路线b:
Step 5b. 当`pre_produce_flag=true`时，首先置`morse-stf/conf/config_ym.json`文件中的`offline_model=true`，于`morse-stf/examples`目录下运行`python3 lr_train_and_predict.py`。此时程序会进行预处理。

Step 6b. 待预处理完毕后，再置`morse-stf/conf/config_ym.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 lr_train_and_predict.py`。此时程序会进行训练/预测。

双方协议

Local运行

下面给出一个在本机开两个端口利用stf训练逻辑回归模型的例子，在本例中，L方有特征，R方有label，假设他们已经将数据对齐。

Step1. 建立一个工作目录morse-stf，将morse-stf源代码中 conf，examples，dataset，serialize子目录copy到morse-stf工作目录下
Step2. 配置morse-stf/conf/config_ym_parties2.json文件，

JSON 复制代码

```json
{
    "parties": 2,
    "pre_produce_flag": true,
    "offline_model": false,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "sess_worker": "workerR",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/embed_op_fea_5w_format_x_train.csv",
            "R": "dataset/embed_op_fea_5w_format_y_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/embed_op_fea_5w_format_x_test.csv",
            "R": "dataset/embed_op_fea_5w_format_y_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}
```

其中，`pre_produce_flag` 字段用来指示双方协议运行的两种方式：
当`pre_produce_flag` 设置为 false时，将不进行预处理，而是使用使用纯online模式，即每一次乘法运算（包括标量乘法，向量乘法，矩阵乘法）都在online用同态加密进行计算。
当`pre_produce_flag` 设置为 true时，将使用预处理：其中当`offline_model`设置为 true时，程序运行于offline model下，将不读取用户数据，仅利用同态加密进行一些预处理，并将预处理得到的伪随机数据存储于当前目录的serialize子目录下；当`offline_model`设置为 false时，程序运行于online model下，将读取用户数据，并利用offline model下预处理的伪随机数，运行MPC协议以进行隐私计算。

`hosts`字段需要配置两台机器的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 修改 `morse-stf/examples/lr_train_and_predict.py` 文件，将代码中的

```
start_local_server(config_file="../conf/config_ym.json")
```

修改为

```
start_local_server(config_file="../conf/config_ym_parties2.json")
```

> 下面分为a, b两条路线:

路线a:
Step 4a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict.py`

路线b:
Step 4b. 当`pre_produce_flag=true` 时，首先置`morse-stf/conf/config_ym_parties2.json`文件中的`offline_model=true`，于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict.py` 。 此时程序会进行预处理。

Step 5b. 待预处理完毕后，再置`morse-stf/conf/config_ym_parties2.json`文件中的`offline_model=false`，于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict.py` 。 此时程序会进行基于隐私计算的LR模型训练/预测。

Distribute运行

下面给出一个利用两台机器（以下称为workerL，workerR）进行逻辑回归模型训练、预测的例子。在本例中，workerL上有特征，workerR上有label，且假设数据已经对齐。

Step 1. 在参与计算的三台机器上分别建立工作目录 morse-stf ，并将开源代码中conf, examples, dataset, serialize目录分别copy到三台机器的morse-stf下。

Step 2. 在workerL, workerR 上分别配置好`morse-stf\conf\config_ym_parties2.json`文件:

JSON 复制代码

```json
{
    "parties": 2,
    "pre_produce_flag": true,
    "offline_model": false,
    "hosts": {
        "workerL": "xxx.xxx.xxx.000:8886",
        "workerR": "xxx.xxx.xxx.001:8886"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "sess_worker": "workerR",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/embed_op_fea_5w_format_x_train.csv",
            "R": "dataset/embed_op_fea_5w_format_y_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/embed_op_fea_5w_format_x_test.csv",
            "R": "dataset/embed_op_fea_5w_format_y_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}

```

其中，`hosts`字段需要配置两台机器的ip:host，而stf_home_workerL, stf_home_workerR字段需要分别填写两台机器上morse-stf工作目录的绝对路径，`ml`字段下的所有路径填写morse-stf下的相对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR 上启动服务。

```
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config_ym_parties2.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config_ym_parties2.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改 `morse-stf/examples/lr_train_and_predict.py` 文件，将
代码中的

```
start_local_server(config_file="../conf/config_ym.json")
```

修改为

```
start_clinet(config_file="../conf/config_ym_parties2.json", job_name="workerR")
```

> 下面分为a, b两条路线，无论那种路线，都只需要在workerR上进行:

路线a:
Step 5a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict.py`

路线b:
Step 5b. 当`pre_produce_flag=true` 时，首先置`morse-stf/conf/config_ym_parties2.json`文件中的`offline_model=true`，于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict.py` 。 此时程序会进行预处理。

Step 6b. 待预处理完毕后，再置`morse-stf/conf/config_ym_parties2.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 lr_train_and_predict.py`。此时程序会进行训练/预测.

双方特征的LR模型训练预测

三方协议

Local运行

下面给出一个在本机开三端口利用stf进行逻辑回归模型训练\预测的例子，在本例中，L方有一部分特征，R方有另一部分特征及label，假设他们已经将数据对齐。

Step1. 建立一个工作目录morse-stf，将morse-stf源代码中 conf，examples，dataset子目录copy到morse-stf工作目录下
Step2. 配置morse-stf/conf/config.json文件

JSON 复制代码

```
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}

```

其中，`hosts`字段需要配置本机的三个空闲的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。
`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step3. 于morse-stf/examples目录下运行python3 lr_train_and_predict2.py

Distribute运行

下面给出一个利用三台机器（以下称为workerL，workerR和RS）进行逻辑回归模型训练、预测的例子。在本例中，workerL上有特征，workerR上有label，且假设数据已经对齐。RS为Random Servies（伪随机

数服务器)，它只负责产生伪随机数，并发送给workerL和workerR，并不从workerL和workerR上接收数据。

Step 1. 在参与计算的三台机器上分别建立工作目录 morse-stf ，并将开源代码中conf, examples, dataset目录分别copy到三台机器的morse-stf下。
Step 2. 在workerL, workerR, RS上分别配置好`morse-stf\conf\config.json`文件:

JSON 复制代码

```
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_RS": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}
```

其中，`hosts`字段需要配置三台机器的ip:host，而stf_home_workerL, stf_home_workerR, stf_home_RS字段需要分别填写三台机器上morse-stf工作目录的绝对路径，`ml`字段下的所有路径填写morse-stf下的相对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。

`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR, RS 上启动服务。

```
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config.json
(在RS上上的morse-stf目录下运行)    morse-stf-server --player=RS --config_file=.\conf\config.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改`morse-stf/examples/lr_train_and_predict2.py` 文件，将
代码中的

```
start_local_server(config_file="../conf/config.json")
```

修改为

```
start_clinet(config_file="../conf/config.json", job_name="workerR")
```

> 下面分为a, b两条路线，无论那种路线，都只需要在workerR上进行:

路线a:
Step 5a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict2.py`

路线b:
Step 5b. 当`pre_produce_flag=true`时，首先置`morse-stf/conf/config.json`文件中的`offline_model=true`，于`morse-stf/examples`目录下运行`python3 lr_train_and_predict2.py`。此时程序会进行预处理。

Step 6b. 待预处理完毕后，再置`morse-stf/conf/config.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 lr_train_and_predict2.py`。此时程序会进行训练/预测。

双方协议

Local运行

下面给出一个在本机开两个端口利用stf训练逻辑回归模型的例子，在本例中，L方有特征，R方有label，假设他们已经将数据对齐。

Step1. 建立一个工作目录morse-stf，将morse-stf源代码中 conf，examples，dataset，serialize子目录copy到morse-stf工作目录下
Step2. 配置morse-stf/conf/config_parties2.json文件，

JSON 复制代码

```json
{
    "parties": 2,
    "pre_produce_flag": true,
    "offline_model": false,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "sess_worker": "workerR",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}
```

其中，`pre_produce_flag` 字段用来指示双方协议运行的两种方式：
当`pre_produce_flag` 设置为 false时，将不进行预处理，而是使用使用纯online模式，即每一次乘法运算（包括标量乘法，向量乘法，矩阵乘法）都在online用同态加密进行计算。
当`pre_produce_flag` 设置为 true时，将使用预处理：其中当`offline_model`设置为 true时，程序运行于offline model下，将不读取用户数据，仅利用同态加密进行一些预处理，并将预处理得到的伪随机数据存储于当前目录的serialize子目录下；当`offline_model`设置为 false时，程序运行于online model下，将读取用户数据，并利用offline model下预处理的伪随机数，运行MPC协议以进行隐私计算。

`hosts`字段需要配置两台机器的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 修改 `morse-stf/examples/lr_train_and_predict2.py` 文件，将代码中的

```
start_local_server(config_file="../conf/config.json")
```

修改为

```
start_local_server(config_file="../conf/config_parties2.json")
```

> 下面分为a, b两条路线:

路线a:
Step 4a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict2.py`

路线b:
Step 4b. 当`pre_produce_flag=true` 时，首先置`morse-stf/conf/config_parties2.json` 文件中的`offline_model=true`，于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict2.py` 。 此时程序会进行预处理。

Step 5b. 待预处理完毕后，再置`morse-stf/conf/config_parties2.json`文件中的`offline_model=false`，于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict2.py` 。 此时程序会进行基于隐私计算的LR模型训练/预测。

Distribute运行

下面给出一个利用两台机器（以下称为workerL，workerR）进行逻辑回归模型训练、预测的例子。在本例中，workerL上有特征，workerR上有label, 且假设数据已经对齐。

Step 1. 在参与计算的三台机器上分别建立工作目录 morse-stf ，并将开源代码中conf, examples, dataset, serialize目录分别copy到三台机器的morse-stf下。

Step 2. 在workerL, workerR 上分别配置好`morse-stf\conf\config_parties2.json`文件:

JSON 复制代码

```json
{
    "parties": 2,
    "pre_produce_flag": true,
    "offline_model": false,
    "hosts": {
        "workerL": "xxx.xxx.xxx.000:8886",
        "workerR": "xxx.xxx.xxx.001:8886"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "sess_worker": "workerR",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}

```

其中，`hosts`字段需要配置两台机器的ip:host，而stf_home_workerL, stf_home_workerR字段需要分别填写两台机器上morse-stf工作目录的绝对路径，`ml`字段下的所有路径填写morse-stf下的相对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR 上启动服务。

```
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config_parties2.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config_parties2.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改 `morse-stf/examples/lr_train_and_predict2.py` 文件，将
代码中的

```
start_local_server(config_file="../conf/config.json")
```

修改为

```
start_clinet(config_file="../conf/config_parties2.json", job_name="workerR")
```

> 下面分为a, b两条路线，无论那种路线，都只需要在workerR上进行:

路线a:
Step 5a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict2.py`

路线b:
Step 5b. 当`pre_produce_flag=true` 时，首先置`morse-stf/conf/config_parties2.json` 文件中的`offline_model=true`，于`morse-stf/examples` 目录下运行`python3 lr_train_and_predict.py` 。 此时程序会进行预处理。

Step 6b. 待预处理完毕后，再置`morse-stf/conf/config_parties2.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 lr_train_and_predict.py`。此时程序会进行双方私有矩阵乘法的安全计算，并输出运算结果。

单方特征的DNN训练预测

三方协议

Local运行

下面给出一个在本机开三端口利用stf进行逻辑回归模型训练\预测的例子，在本例中，L方有特征，R方有label，假设他们已经将数据对齐。

Step1. 建立一个工作目录morse-stf，将morse-stf源代码中 conf，examples，dataset子目录copy到morse-stf工作目录下
Step2. 配置morse-stf/conf/config.json文件

JSON 复制代码

```json
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/embed_op_fea_5w_format_x_train.csv",
            "R": "dataset/embed_op_fea_5w_format_y_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/embed_op_fea_5w_format_x_test.csv",
            "R": "dataset/embed_op_fea_5w_format_y_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}

```

其中，`hosts`字段需要配置本机的三个空闲的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。
`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step3. 于morse-stf/examples目录下运行python3 DNN_train_and_predict.py

Distribute运行

下面给出一个利用三台机器（以下称为workerL，workerR和RS）进行逻辑回归模型训练、预测的例子。在本例中，workerL上有特征，workerR上有label，且假设数据已经对齐。RS为Random Servies（伪随机

数服务器)，它只负责产生伪随机数，并发送给workerL和workerR，并不从workerL和workerR上接收数据。

Step 1. 在参与计算的三台机器上分别建立工作目录 morse-stf ，并将开源代码中conf, examples, dataset目录分别copy到三台机器的morse-stf下。
Step 2. 在workerL, workerR, RS上分别配置好`morse-stf\conf\config_ym.json`文件:

JSON 复制代码

```
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_RS": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/embed_op_fea_5w_format_x_train.csv",
            "R": "dataset/embed_op_fea_5w_format_y_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/embed_op_fea_5w_format_x_test.csv",
            "R": "dataset/embed_op_fea_5w_format_y_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}
```

其中，`hosts`字段需要配置三台机器的ip:host，而stf_home_workerL, stf_home_workerR, stf_home_RS字段需要分别填写三台机器上morse-stf工作目录的绝对路径，`ml`字段下的所有路径填写morse-stf下的相对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。

`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR, RS 上启动服务。

```
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config_ym.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config_ym.json
(在RS上上的morse-stf目录下运行)    morse-stf-server --player=RS --config_file=.\conf\config_ym.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改`morse-stf/examples/DNN_train_and_predict.py` 文件，将
代码中的

```
start_local_server(config_file="../conf/config_ym.json")
```

修改为

```
start_clinet(config_file="../conf/config_ym.json", job_name="workerR")
```

> 下面分为a, b两条路线，无论那种路线，都只需要在workerR上进行:

路线a:
Step 5a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 DNN_train_and_predict.py`

路线b:
Step 5b. 当`pre_produce_flag=true`时，首先置`morse-stf/conf/config_ym.json`文件中的`offline_model=true`，于`morse-stf/examples`目录下运行`python3 DNN_train_and_predict.py`。此时程序会进行预处理。

Step 6b. 待预处理完毕后，再置`morse-stf/conf/config_ym.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 DNN_train_and_predict.py`。此时程序会进行训练/预测。

双方协议

双方协议暂未支持DNN。

双方特征的DNN训练预测

三方协议

Local运行

下面给出一个在本机开三端口利用stf进行逻辑回归模型训练\预测的例子，在本例中，L方有一部分特征，R方有另一部分特征及label，假设他们已经将数据对齐。

Step1. 建立一个工作目录morse-stf，将morse-stf源代码中 conf，examples，dataset子目录copy到morse-stf工作目录下
Step2. 配置morse-stf/conf/config.json文件

JSON 复制代码

```
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}

```

其中，`hosts`字段需要配置本机的三个空闲的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。

`default_fixed_point` 字段表示默认的定点位置。

`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。

`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。

其余字段在本例中没有用途。

Step3. 于morse-stf/examples目录下运行python3 DNN_train_and_predict2.py

Distribute运行

下面给出一个利用三台机器（以下称为workerL，workerR和RS）进行逻辑回归模型训练、预测的例子。在本例中，workerL上有特征，workerR上有label，且假设数据已经对齐。RS为Random Servies（伪随机

数服务器)，它只负责产生伪随机数，并发送给workerL和workerR，并不从workerL和workerR上接收数据。

Step 1. 在参与计算的三台机器上分别建立工作目录 morse-stf ，并将开源代码中conf, examples, dataset目录分别copy到三台机器的morse-stf下。
Step 2. 在workerL, workerR, RS上分别配置好`morse-stf\conf\config.json`文件:

JSON 复制代码

```
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_RS": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}
```

其中，`hosts`字段需要配置三台机器的ip:host，而stf_home_workerL, stf_home_workerR, stf_home_RS字段需要分别填写三台机器上morse-stf工作目录的绝对路径，`ml`字段下的所有路径填写morse-stf下的相对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。

`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下应该填写训练集和测试集在 stf_home 下的相对路径。
`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR, RS 上启动服务。

```
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config.json
(在RS上上的morse-stf目录下运行)    morse-stf-server --player=RS --config_file=.\conf\config.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改`morse-stf/examples/DNN_train_and_predict2.py` 文件，将
代码中的

```
start_local_server(config_file="../conf/config.json")
```

修改为

```
start_clinet(config_file="../conf/config.json", job_name="workerR")
```

> 下面分为a, b两条路线，无论那种路线，都只需要在workerR上进行:

路线a:
Step 5a. 当`pre_produce_flag=false` 时， 于`morse-stf/examples` 目录下运行`python3 DNN_train_and_predict2.py`

路线b:
Step 5b. 当`pre_produce_flag=true`时，首先置`morse-stf/conf/config.json`文件中的`offline_model=true`，于`morse-stf/examples`目录下运行`python3 DNN_train_and_predict2.py`。此时程序会进行预处理。

Step 6b. 待预处理完毕后，再置`morse-stf/conf/config.json`文件中的`offline_model=false`，于`morse-stf/examples`目录下运行`python3 DNN_train_and_predict2.py`。此时程序会进行训练/预测。

双方协议

双方协议下暂未提供对DNN的支持

单方特征的CNN训练预测

三方协议

Local运行

下面给出一个在本机开三端口利用stf进行CNN训练\预测的例子. 本例的数据集为MNIST数据集.
在本例中，L方有特征(图片)，R方有label，假设他们已经将数据对齐。

Step1. 建立一个工作目录morse-stf，将morse-stf源代码中 conf，examples，dataset子目录copy到morse-stf工作目录下
Step2. 配置morse-stf/conf/config.json文件

JSON 复制代码

```json
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}

```

其中，`hosts`字段需要配置本机的三个空闲的ip:host，而`stf_home`字段需要填写morse-stf工作目录的绝对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。

`compress_flag` 标志指示在传输时是否采用压缩。采用压缩可以减少通信开销，但是会增大计算开销。

`default_fixed_point` 字段表示默认的定点位置。

`ml` 字段下的`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。

`protocols` 字段可允许用户对实现某些功能所使用的MPC协议进行选择。可详见【协议选择】章节。

其余字段在本例中没有用途。

Step3. 于morse-stf/examples目录下运行python3 run_networkB.py

Distribute运行

下面给出一个利用三台机器（以下称为workerL，workerR和RS）进行逻辑回归模型训练、预测的例子。在本例中，workerL上有特征，workerR上有label，且假设数据已经对齐。RS为Random Servies（伪随机数服务器），它只负责产生伪随机数，并发送给workerL和workerR，并不从workerL和workerR上接收数据。

Step 1. 在参与计算的三台机器上分别建立工作目录 morse-stf ，并将开源代码中conf，examples，dataset目录分别copy到三台机器的morse-stf下。
Step 2. 在workerL，workerR，RS上分别配置好`morse-stf\conf\config.json`文件：

JSON 复制代码

```json
{
    "parties": 3,
    "hosts": {
        "workerL": "0.0.0.0:8886",
        "workerR": "0.0.0.0:8887",
        "RS": "0.0.0.0:8888"
    },
    "stf_home_workerL": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_workerR": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "stf_home_RS": "/Users/qizhi.zqz/projects/morse-stf/morse-stf",
    "prf_flag": true,
    "compress_flag": true,
    "default_fixed_point": 14,
    "ml": {
        "dataset_train": {
            "L": "dataset/xindai_xx_train.csv",
            "R": "dataset/xindai_xy_train.csv"
        },
        "dataset_predict": {
            "L": "dataset/xindai_xx_test.csv",
            "R": "dataset/xindai_xy_test.csv"
        },
        "predict_to_file": "output/predict"
    },
    "protocols":
    {
        "drelu": "log"
    }

}
```

其中，`hosts`字段需要配置三台机器的ip:host，而stf_home_workerL，stf_home_workerR，stf_home_RS字段需要分别填写三台机器上morse-stf工作目录的绝对路径，`ml`字段下的所有路径填写morse-stf下的相对路径。

`prf_flag` 字段表示在伪随机数生成时是否使用prf. 使用prf可以减少通信开销，但是会增大计算开销。
`compress_flag` 标志指示在传输时是否采用压缩。采用压缩可以减少通信开销，但是会增大计算开销。
`default_fixed_point` 字段表示默认的定点位置。
`ml` 字段下的`predict_to_file` 应填写用来保存预测结果的文件在stf_home 下的相对路径。
`protocols` 字段可允许用户对实现某些功能所使用的MPC协议进行选择。可详见【协议选择】章节。
其余字段在本例中没有用途。

Step 3. 在workerL, workerR, RS 上启动服务。

```
(在workerL上的morse-stf目录下运行) morse-stf-server --player=workerL  --config_file=.\conf\config.json
(在workerR上的morse-stf目录下运行) morse-stf-server --player=workerR --config_file=.\conf\config.json
(在RS上上的morse-stf目录下运行)    morse-stf-server --player=RS --config_file=.\conf\config.json
```

> Step 4及以下步骤，仅需要在一台机器上执行，本文档中以在workerR上执行为例进行说明：

Step 4.
修改`morse-stf/examples/run_networkB.py` 文件，将
代码中的

```
start_local_server(config_file="../conf/config.json")
```

修改为

```
start_clinet(config_file="../conf/config.json", job_name="workerR")
```

Step 5. 于`morse-stf/examples` 目录下运行`python3 run_networkB.py`

> 用户可以用完全相同的方法运行run_networkC.py 和 run_networkD.py. 他们与run_networkB.py的区别仅在于神经网络的结构。另外，run_networkA.py是以全连接神经网络来进行MNIST数据集的训练和预测，其运行方法也与run_networkB.py一致。

双方协议

双方协议暂不支持CNN的训练/预测。

数据类型

MORSE-STF的数据类型是按照隐私性进行划分的。总的来讲，分为三种：PrivateTensor, SharedTensor, SharedPair.

其中PrivateTensor表示单方私有数据，由一个tf.Tensor of dtype=int64型的inner_value, 一个int型的module, 一个int型的fixedpoint, 和一个owner组成。当module is None时，它表示一个元素为定点数的Tensor，其实际表示的值为 inner_value * pow(2, -fixedpoint)；当module is not None时，它表示一个module阶循环群上的Tensor.

SharedTensor表示一个数据分片，可以理解为PrivateTensor忽略了fixedpoint和owner. 当module is None时，

SharedPair表示一个由和分片形式存储于L, R双方的数据。一个SharedPair包含一个SharedTensor 型的xL， 一个SharedTensor型的xR, 一个ownerL, 一个ownerR, 以及一个fixedpoint. 并且要求ownerL!=ownerR and xL.module==xR.module. 当module is None时，它表示一个元素为定点数的Tensor，其实际表示的值为 (xL.inner_value+xR.inner_value mod pow(2, 64)) * pow(2, -fixedpoint)；当module is not None时，它表示一个module阶循环群上的Tensor, 其实际表示的值为(xL.inner_value+xR.inner_value mod module)

数据载入

MORSE-STF中，只有PrivateTensor有数据载入功能。我们为PrivateTensor提供了如下几个API:
load_from_numpy()
load_from_tf_tensor()

load_first_line_from_file()
load_from_file()
load_from_file_withid()
其中，前3个得到const PrivateTensor, 后两个得到的是非const PrivateTensor.

下面以load_from_file()为例讲述数据载入方法：

```JSON
x_train = PrivateTensor(owner='L')
format_x = [["a"], [0.1], [0.1], [0.1], [0.1], [0.1]]
x_train.load_from_file(path=path,
                       record_defaults=format_x, batch_size=batch_size,
repeat=repeat, skip_col_num=1)
```

其中，path 为数据在磁盘上的绝对路径，format_x为数据格式，batch_size为batch size，repeat为读取数据重复次数，skip_col_num为跳过前面的几列。

模型构建

单方特征逻辑回归模型

对于单方特征的逻辑回归模型，只需要

```Python
from stensorflow.ml.logistic_regression import LogisticRegression
model = LogisticRegression(num_features=featureNum,
learning_rate=learning_rate)
```

即可构建模型。其中featureNum为特征数，learning_rate为学习率。

双方特征逻辑回归模型

对于双方逻辑回归模型，只需要

```Python
from stensorflow.ml.logistic_regression2 import LogisticRegression2
model = LogisticRegression2(learning_rate=learning_rate,
num_features_L=featureNumL, num_features_R=featureNumR)
```

其中，featureNumL为workerL方的特征数，featureNumR为workerR方的特征数，learning_rate为学习率。

单方特征全连接神经网络

只需要

```Python
from stensorflow.ml.nn.networks.DNN import DNN
model = DNN(feature=x_train, label=y_train, dense_dims=dense_dims)
model.compile()
```

即可构建单方特征的全连接神经网络，其中x_train为特征的PrivateTensor，y_train为label的PrivateTensor，dense_dims是一个int型的list,代表神经网络各层神经元的数量（从第0层，及特征层开始）。例如，对于一个用于解决5分类问题的含一个隐层的全连接神经网络，输入为256维的特征，隐层神经元为128个，输出为一个5维的概率值，则dense_dims=[256,128,5].

双方特征全连接神经网络

只需要

```Python
from stensorflow.ml.nn.networks.DNN import DNN
model = DNN(feature=xL_train, label=y_train, dense_dims=dense_dims,
feature_another=xR_train)
model.compile()
```

即可构建双方特征的全连接神经网络，其中xL_train为L方特征的PrivateTensor，xR_train为R方特征的PrivateTensor，y_train为label的PrivateTensor，dense_dims是一个int型的list,代表神经网络各层神经元的数量（从第0层，及特征层开始）。例如，对于一个用于解决5分类问题的含一个隐层的全连接神经网络，输入为256维的特征，隐层神经元为128个，输出为一个5维的概率值，则dense_dims=[256,128,5].

单方特征卷积神经网络

只需要

```Python
from stensorflow.ml.nn.networks.NETWORKB import NETWORKB
model = NETWORKB(feature=x_train, label=y_train,
loss="CrossEntropyLossWithSoftmax")
model.compile()
```

即可构建单方特征的卷积神经网络，其中x_train为特征的PrivateTensor，y_train为label的PrivateTensor。NetworkB的网络结构如下描述：

Python 复制代码

```
# layers for NETWORKB
Conv2D(16, (5, 5), activation='relu', use_bias=False),
AvgPool2D(2, 2),
Conv2D(16, (5, 5), activation='relu', use_bias=False),
AvgPool2D(2, 2),
Flatten(),
Dense(100, activation='relu'),
Dense(10, name="Dense"),
Activation('softmax')
```

与之类似的，还有NETWORKA, NETWORKC, 和NETWORKD（其中NETWORKA是全连接的神经网络，不是卷积神经网络，但由于使用场景，定义方法都与卷积神经网络类似，所以在这里一并介绍）。他们的网络结构如下：

Python 复制代码

```
# layers for NETWORKA
Dense(128, input_dim=28*28),
Activation('relu'),
Dense(128),
Activation('relu'),
Dense(10),
Activation('softmax')

# layers for NETWORKC
Conv2D(20, (5, 5), activation='relu', use_bias=False),
AvgPool2D(2, 2),
Conv2D(50, (5, 5), activation='relu', use_bias=False),
AvgPool2D(2, 2),
Flatten(),
Dense(500, activation='relu'),
Dense(10, name="Dense"),
Activation('softmax')

# layers for NETWORKD
Conv2D(5, (5, 5), activation='relu', use_bias=False),
AvgPool2D(2, 2),
Flatten(),
Dense(100, activation='relu'),
Dense(10, name="Dense"),
Activation('softmax')
```

模型训练

在模型训练之前，要进行Session和Variable的初始化，通常需要加入如下代码：

```python
sess = tf.compat.v1.Session(StfConfig.target)
init_op = tf.compat.v1.initialize_all_variables()
sess.run(init_op)
```

初始化完毕后，即可进行模型训练

单方特征逻辑回归的模型训练

只需要

```python
model.fit(sess=sess, x=x_train, y=y_train, num_batches=train_batch_num)
```

即可实现单方特征逻辑回归模型的训练。其中sess为tensorflow的Session对象，x_train为特征的PrivateTensor,
y_train为label的PrivateTensor, train_batch_num为int,代表要训练多少个batch.

双方特征逻辑回归的模型训练

只需要

```python
model.fit(sess, x_L=xL_train, x_R=xR_train, y=y_train,
num_batches=train_batch_num)
```

即可实现单方特征逻辑回归模型的训练。其中sess为tensorflow的Session对象，xL_train为workerL方特征的PrivateTensor, xR_train为workerR方特征的PrivateTensor, y_train为label的PrivateTensor, train_batch_num为int,代表要训练多少个batch.

神经网络的模型训练

无论单方特征的全连接神经网络，或双方特征的全连接神将网络，还是卷积神经网络，训练的语法是相同的，只需要

```python
model.train_sgd(learning_rate=learning_rate, batch_num=train_batch_num,
l2_regularization=l2_regularization, sess=sess)
```

即可实现神经网络的训练。其中sess为tensorflow的Session对象，learning_rate代表学习率，l2_regularization表示所使用的的l2正则化系数，batch_num为int,代表要训练多少个batch、.

模型预测

单方特征逻辑回归模型预测

只需要

```python
model.predict(id, x_test, pred_batch_num, sess, predict_file=None)
```

即可实现单方特征逻辑回归模型的预测。其中sess为tensorflow的Session对象，x_test为特征的PrivateTensor, id为id列的PrivateTensor, pred_batch_num为int,代表要预测多少个batch, predict_file为预测结果写入的文件名. 如果predict_file为None, 将会将预测结果写入StfConfig.predict_to_file，后者可在config.json文件中配置。

双方特征逻辑回归模型预测

只需要

```python
model.predict(id, xL_test, xR_test, pred_batch_num, sess,  predict_file=None)
```

即可实现单方特征逻辑回归模型的预测。其中sess为tensorflow的Session对象，xL_test为workerL方特征的PrivateTensor, xR_test为workerR方特征的PrivateTensor, id为id列的PrivateTensor, pred_batch_num为int,代表要预测多少个batch，predict_file为预测结果写入的文件名. 如果predict_file为None, 将会将预测结果写入StfConfig.predict_to_file，后者可在config.json文件中配置。

单方特征全连接神经网络的预测

只需要

```python
model.predict_to_file(sess=sess, x=x_test,
predict_file_name=StfConfig.predict_to_file,
                      batch_num=pred_batch_num, idx=id)
```

即可实现单方特征逻辑回归模型的预测。其中sess为tensorflow的Session对象，x_test为特征的PrivateTensor，id为id列的PrivateTensor，pred_batch_num为int,代表要预测多少个batch，StfConfig.predict_to_file为预测结果写入的文件名， 可在config.json文件中配置。

双方特征全连接神经网络的预测

只需要

```python
model.predict_to_file(sess=sess, x=xL_test, x_another=xR_test,
                      predict_file_name=StfConfig.predict_to_file,
                      batch_num=pred_batch_num, idx=id)
```

即可实现单方特征逻辑回归模型的预测。其中sess为tensorflow的Session对象，xL_test为workerL方特征的PrivateTensor，xR_test为workerR方特征的PrivateTensor，id为id列的PrivateTensor，pred_batch_num为int,代表要预测多少个batch， StfConfig.predict_to_file为预测结果写入的文件名，可在config.json文件中配置。

单方特征卷积神经网络的预测

只需要

```python
model.predict_to_file(sess, x_test,
predict_file_name=StfConfig.predict_to_file,
                      pred_batch_num=pred_batch_num,
                      with_sigmoid=False)
```

即可实现单方特征卷积神经网络的预测。其中sess为tensorflow的Session对象，xL_test为workerL方特征的PrivateTensor，xR_test为workerR方特征的PrivateTensor，pred_batch_num为int,代表要预测多少个batch， StfConfig.predict_to_file为预测结果写入的文件名，可在config.json文件中配置。

协议选择

MORSE-STF实际上是一个基于混合协议族的多方安全计算系统，允许用户为某功能设置不同的MPC。目前DReLU支持三种协议"const"，"log"，"linear"，用户只需要在config.json的"protocols"字段进行配置。

```yaml
    "protocols":
    {
        "drelu": "log"    // DReLU协议选择，可选 "const", "log", "linear"
    }
```

示例：不同网络状态下，对于不同的数据集，通过设置不同的DReLU协议来获得更高的训练速度

dataset	Network	delay	train time		
			$\Pi_{\text{DReLU}}^{\text{linear}}$	$\Pi_{\text{DReLU}}^{\text{log}}$	$\Pi_{\text{DReLU}}^{\text{const}}$
xindai10	32,32 *	5ms	67s	46s	208s
xindai10	32,32	10ms	113s	52s	208s
xindai10	32,32	30ms	306s	94s	224s
xindai10	32,32	60ms	606s	165s	252s
xindai291	7,7	5ms	226s	93s	204s
xindai291	7,7	10ms	445s	133s	205s
xindai291	7,7	30ms	1307s	308s	253s
xindai291	7,7	60ms	2549s	612s	492s
xindai291	32,32	5ms	234s	123S	928s
xindai291	32,32	10ms	450s	134s	928s
xindai291	32,32	30ms	1332s	296s	979s
xindai291	32,32	60ms	2657s	640s	1284s
Mnist	network A	60ms	7996s	790s	2527s
Mnist	network B	60ms	19125s	26109s	-
Mnist	network C	60ms	23766s	33764s	-
Mnist	network D	60ms	4620s	3230s	25913s

* It means that the full-connected network with two hidden layer, the size of hidden layers are 32, 32. The time is for training 1 epoch.
- In this case, memory out occur.