# ONNX 格式介绍

Mingzhu Yan

2026-01-07

# Outline

# 1. Pytorch 手写数字识别

# 1.1 神经网络

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

# 1.2 训练并保存快照

```
def train(epoch):
    network.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()

        output = network(data)

        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % log_interval == 0:
            torch.save(network.state_dict(), "./model.pth")
            torch.save(optimizer.state_dict(), "./optimizer.pth")
```

# 1.3 Pytorch 默认模型格式 .pth


```
{
    "conv1.weight": {
        "shape": [
            10,
            1,
            5,
            5
        ],
        "values_preview": [
            -0.04187455028295517,
            0.33693239092826843,
            0.22457480430603027,
            0.2785506844520569,
            0.24370838701725006
        ]
    },
    "conv1.bias": {
        "shape": [
            10
        ],
        "values_preview": [
            -0.01581696793437004,
            -0.008713895455002785,
            0.19034163653850555,
            0.10842573642730713,
            0.15954284369945526
        ]
    },
    "conv2.weight": {
        "shape": [
            20,
            10,
            5,
            5
        ],
        "values_preview": [
            -0.008292394690215588,
            -0.08871737867593765,
```


model.pth

- 是个压缩包, 里面是序列化后的 python 字典
- 可以将其转换为 json 可读格式

# 1.4 使用 Pytorch 运行 .pth 模型

```
img = Image.open(image_path)
# 转换维度 [1, 1, 28, 28]
img_tensor = transform(img).unsqueeze(0)  # type: ignore

# 执行推理
with torch.no_grad():
    output = network(img_tensor)
    # 获取概率最大的索引
    prediction = output.data.max(1, keepdim=True)[1]
```

- pth 不存储实际执行顺序
- pth 不存储每一个层该如何执行

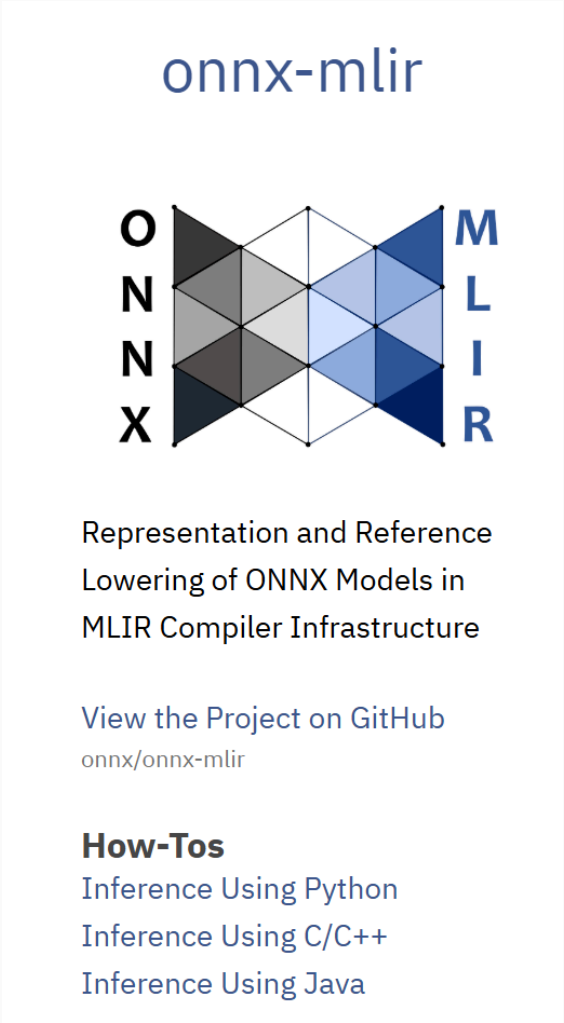
想要执行, 必须使用 pytorch, 并构建相同的网络结构 (minst-demo/pytorch_run.py)

# 2. 导出为 onnx 并使用 onnxruntime 运行

# 2.1 通用的模型格式

- ONNX 不依赖据具体的深度学习框架
- 基于 protobuf, 支持多种编程语言
- 存储执行顺序
- 定义了算子集, 和算子的行为, 需要由具体的 runtime 实现

任何实现了 ONNX 的运行时均可以运行 ONNX 模型


,

# 2.2 pth 转换为 onnx 并用 onnxruntime 运行

```
network = Net()
network.load_state_dict(torch.load("./model.pth", map_location=torch.device("cpu")))
network.eval()

# 2. 创建一个“伪输入”（Dummy Input）
# ONNX 导出需要通过一次“干跑”来追踪计算图
dummy_input = torch.randn(1, 1, 28, 28)

# 3. 导出模型
torch.onnx.export(
    network,  # 要导出的模型
    dummy_input,  # 伪输入 # type: ignore
    "model.onnx",  # 导出文件名
    do_constant_folding=False,  # 是否执行常量折叠优化
    input_names=["input"],  # 输入节点名称
    output_names=["output"],  # 输出节点名称
)
```

# 3. ONNX 格式

# 3.1 protobuf 定义

ONNX 的格式使用 protobuf 定义

https://github.com/onnx/onnx/blob/main/onnx/onnx.proto

# 3.1 protobuf 定义

```
message ModelProto {
  // The version of the IR this model targets. See Version enum above.
  // This field MUST be present.
  optional int64 ir_version = 1;

  // The OperatorSets this model relies on.
  repeated OperatorSetIdProto opset_import = 8;

  // The name of the framework or tool used to generate this model.
  optional string producer_name = 2;

  // The version of the framework or tool used to generate this model.
  optional string producer_version = 3;

  // Domain name of the model.
  // We use reverse domain names as name space indicators. For example:
  // `com.facebook.fair` or `com.microsoft.cognitiveservices`
  optional string domain = 4;

  // The parameterized graph that is evaluated to execute the model.
  optional GraphProto graph = 7;
```

# 4. Python 手动构建 onnx 模型

# 4.1 Python 手动构建 onnx 模型

```
input_tensor_info = helper.make_tensor_value_info(
    name="X", elem_type=TensorProto.FLOAT, shape=[1, 4]
)

output_tensor_info = helper.make_tensor_value_info(
    name="Y", elem_type=TensorProto.FLOAT, shape=[1, 4]
)

node_def = helper.make_node(
    op_type="Softmax", inputs=["X"], outputs=["Y"], name="node_softmax_1"
)

graph_def = helper.make_graph(
    name="test_softmax_graph",
    inputs=[input_tensor_info],
    outputs=[output_tensor_info],
    nodes=[node_def],
    initializer=[],
)

model_def = helper.make_model(
    graph_def,
    producer_name="my_human_hand",
    opset_imports=[helper.make_opsetid(domain="", version=11)],
    ir_version=11,
)

onnx.save(model_def, "softmax.onnx")
```

# 4.2 onnx 对算子的描述

https://onnx.ai/onnx/operators/