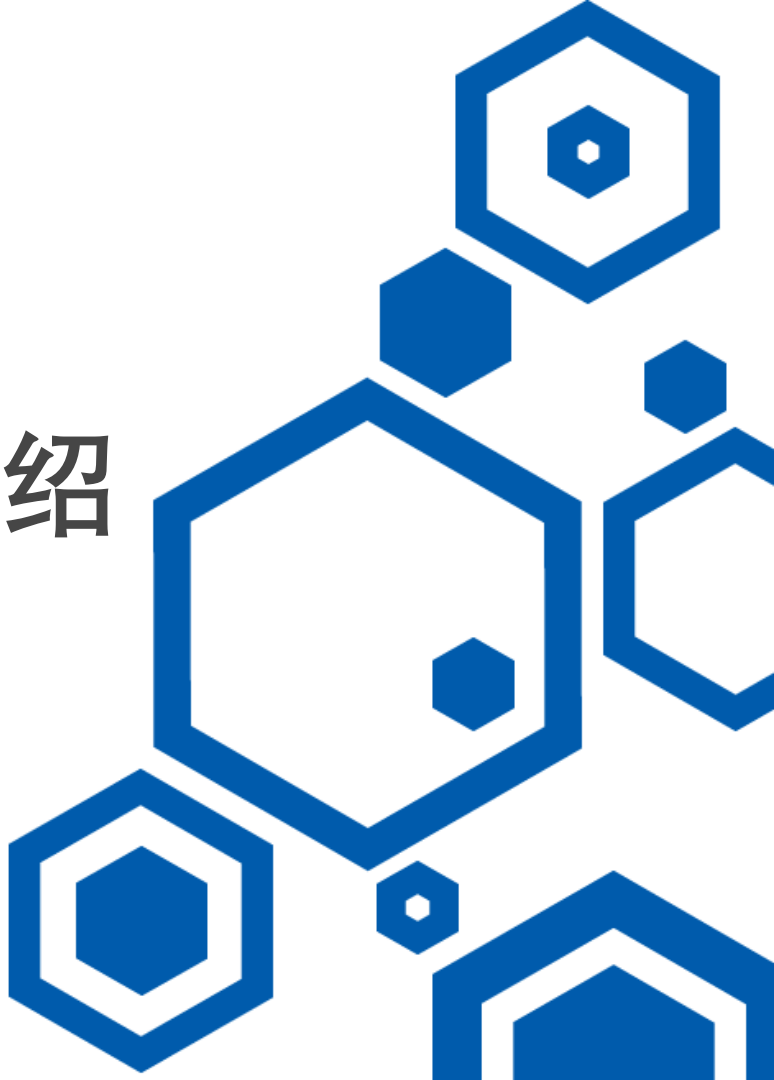
深蓝学院
shenlanxueyuan.com

TensorRT Plugin 介绍



课程目标

理论部分

- TensorRT Plugin 基本概念
- TensorRT Plugin工作流程
- TensorRT Plugin API 介绍
- TensorRT Plugin Demo 代码讲解

课程大纲

基础	1. TensorRT Plugin 介绍 2. TensorRT Plugin 的工作流程 3. Static Shape Plugin API 4. EmbLayerNormPlugin Static Shape Demo 5. Dynamic Shape Plugin API 6. EmbLayerNormPlugin Dynamic Shape Demo 7. PluginCreator 注册
进阶	8. TensorRT 如何 debug – Debug Plugin

TensorRT Plugin介绍

Plugin 存在的意义：

1. TRT支持的算子有限，实现不支持的算子；

2. 进行深度优化-合并算子。

TensorRT Plugin介绍

支持的算子

Caffe

These are the operations that are supported in a Caffe framework:

- Convolution
- Pooling
- InnerProduct
- SoftMax
- ReLU, TanH, and Sigmoid
- LRN
- Power
- ElementWise
- Concatenation
- Deconvolution
- BatchNormalization
- Scale
- Crop
- Reduction
- Reshape
- Permute
- Dropout

TensorFlow

These are the operations that are supported in a TensorFlow framework:

- Placeholder
- Const
- Add, Sub, Mul, Div, Minimum and Maximum
- BiasAdd
- Negative, Abs, Sqrt, Rsqrt, Pow, Exp and Log

Note: The NvUffParser supports Neg, Abs, Sqrt, Rsqrt, Exp and Log for const nodes only.

- FusedBatchNorm
- ReLU, TanH, and Sigmoid
- SoftMax

Note: If the input to a TensorFlow SoftMax op is not NHWC, TensorFlow will automatically insert a transpose layer with a non-constant permutation, causing the UFF converter to fail. It is therefore advisable to manually transpose SoftMax inputs to NHWC using a constant permutation.

- Mean
- ConcatV2
- Reshape
- Transpose
- Conv2D
- DepthwiseConv2dNative
- ConvTranspose2D
- MaxPool
- AvgPool
- Pad is supported if followed by one of these TensorFlow layers: Conv2D, DepthwiseConv2dNative, MaxPool, and AvgPool

ONNX

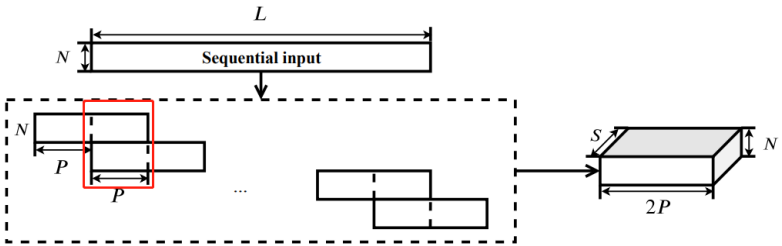
Since the ONNX parser is an open source project, the most up-to-date information regarding the supported operations can be found in GitHub: ONNX TensorRT.
These are the operations that are supported in the ONNX framework:

- Abs,Add,AveragePool
- BatchNormalization,Ceil,Clip,Concat,Constant
- Conv,ConvTranspose,DepthToSpace
- Div,Dropout,Elu,Exp
- Flatten,Floor,Gemm
- GlobalAveragePool,GlobalMaxPool
- HardSigmoid,Identity,InstanceNormalization
- LRN,LeakyRelu,Log,LogSoftmax
- MatMul,Max,MaxPool,Mean,Min,Mul
- Neg,Prelu,Pad,Pow
- Reciprocal,ReduceL1,ReduceL2,ReduceLogSum
- ReduceLogSumExp,ReduceMax
- ReduceMean,ReduceMin,ReduceProd,ReduceSum
- ReduceSumSquare,Relu,Reshape,Selu,Shape
- Sigmoid,Size,Softmax,Softplus
- SpaceToDepth,Split,Squeeze,Sub,Sum
- Tanh
- TopK
- Transpose
- Unsqueeze
- Upsample

不支持的算子：groupnorm，gelu，split等

TensorRT Plugin介绍

对于复杂的网络，合并算子是非常有意义的。比如，可以将下方的代码合并为一个plugin，也就是一个kernel，可以有效提高性能。



DPRNN Segmentation模块流程图

```
# 将输入特征分割成特征块，特征块的大小为(N, segment_size)，再将所有的特征块拼接起来。相邻的两个特征块存在一半的重叠。
# 输入：
# input: 输入特征，大小为(B, N, T)，B 为 batch_size，N 为帧数，T 为特征维度
# segment_size：特征块的维度
# 输出：
# output：输出特征块，大小为(B, S, N, segment_size)。S 为特征块的数量
def Segment_feature(self, input, segment_size):
    # 获得输入的维度和重叠的大小
    batch_size, N, T = input.shape
    segment_stride = segment_size // 2

    # 获得前一半的特征块
    segments1 = input[:, :, :-segment_stride].contiguous().view(batch_size, N, -1, segment_size)
    # 获得后一半的特征块
    segments2 = input[:, :, segment_stride:].contiguous().view(batch_size, N, -1, segment_size)
    # 将两部分特征块拼接后并转置
    segments = torch.cat([segments1, segments2], 3).view(batch_size, N, -1, segment_size).transpose(2, 3)
    return segments.contiguous()
```

DPRNN Segmentation 模块 Pytorch 源码

TensorRT Plugin介绍

官方github给出了很多plugin demo，大都跟计算机视觉和BERT模型相关。

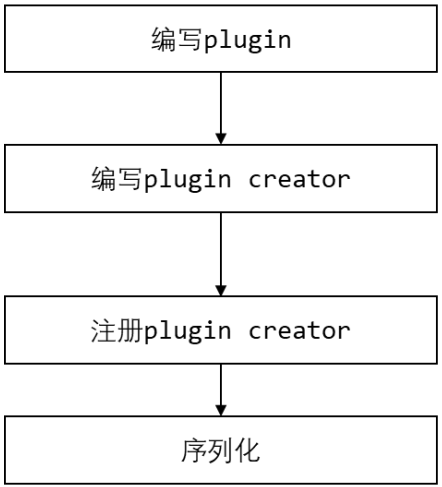
TensorRT version 8.2

Contents

Plugin	Name	Versions
batchTilePlugin	BatchTilePlugin_TRT	1
batchedNMSPlugin	BatchedNMS_TRT	1
batchedNMSDynamicPlugin	BatchedNMSDynamic_TRT	1
bertQKVToContextPlugin	CustomQKVToContextPluginDynamic	1, 2, 3
coordConvACPlugin	CoordConvAC	1
cropAndResizePlugin	CropAndResize	1
detectionLayerPlugin	DetectionLayer_TRT	1
efficientNMSPlugin	EfficientNMS_TRT	1
efficientNMSONNXPlugin	EfficientNMS_ONNX_TRT	1
embLayerNormPlugin	CustomEmbLayerNormPluginDynamic	1, 2
fcPlugin	CustomFCPluginDynamic	1
flattenConcat	FlattenConcat_TRT	1
geluPlugin	CustomGeluPluginDynamic	1
generateDetectionPlugin	GenerateDetection_TRT	1

https://github.com/NVIDIA/TensorRT/tree/release/8.2/plugin

TensorRT Plugin的工作流程



pluginType

pluginVersion

pluginName

pluginVersion

REGISTER_TENSORRT_PLUGIN

getPluginRegistry()->registerCreator(*mCreator, libNamespace);

将参数和权值写到文件中

Static Shape Plugin API

Dynamic Shape：输入维度是动态的；

Static Shape：输入维度是定死的。

IPluginV2IOExt / IPluginV2DynamicExt：插件类，用于写插件的具体实现；

IPluginCreator：插件工厂类，用于根据需求创建该插件。

	Introduced in TensorRT version?	Mixed input/ output formats/ types	Dynamic shapes?	Requires extended runtime?
IPluginV2Ext	5.1	Limited	No	No
IPluginV2IOExt	6.0.1	General	No	No
IPluginV2DynamicExt	6.0.1	General	Yes	Yes

注意：

- 编写plugin，需要继承TRT的base class（不同的base class特性如上表）；
- Static Shape，用IPluginV2IOExt；Dynamic Shape，则使用IPluginV2DynamicExt。

Static Shape Plugin API

```
MyCustomPlugin(int in_channel, nvinfer1::Weights const& weight, nvinfer1::Weights const& bias);
MyCustomPlugin(void const* serialData, size_t serialLength);
int getNbOutputs() const;
nvinfer1::Dims getOutputDimensions(int index, const nvinfer1::Dims* inputs, int nbInputDims);
nvinfer1::DataType getOutputDataType(int index, const nvinfer1::DataType* inputTypes, int nbInputs) const;
size_t getSerializationSize() const;
void serialize(void* buffer) const;
const char* getPluginType() const;
const char* getPluginVersion() const;
int initialize();
void terminate();
void destroy();
void configurePlugin(const nvinfer1::PluginTensorDesc* in, int nbInput,  const
nvinfer1::PluginTensorDesc* out, int nbOutput);
bool supportsFormatCombination(int pos, const nvinfer1::PluginTensorDesc* inOut, int nbInputs, int
nbOutputs) const;
size_t getWorkspaceSize(int maxBatchSize) const;
int enqueue(int batchSize, const void* const* inputs, void** outputs, void* workspace, cudaStream_t
stream);
```

Static Shape Plugin API

构造函数和析构函数

构造函数

1. 用于network definition阶段，PluginCreator创建该插件时调用的构造函数，需要传递权重信息以及参数。也可用于clone阶段，或者再写一个clone构造函数。

```
MyCustomPlugin(int in_channel, nvinfer1::Weights const& weight, nvinfer1::Weights const&
bias);
```

2. 用于在deserialize阶段，用于将序列化好的权重和参数传入该plugin并创建。

```
MyCustomPlugin(void const* serialData, size_t serialLength);
```

3. 注意需要把默认构造函数删掉：

```
MyCustomPlugin() = delete;
```

析构函数

析构函数则需要执行terminate，terminate函数就是释放这个op之前开辟的一些显存空间：

```
MyCustomPlugin::~MyCustomPlugin() {
    terminate();
}
```

Static Shape Plugin API

输出相关函数

1. 获得layer的输出个数

```
int getNbOutputs() const;
```

2. 根据输入个数和输入维度，获得第index个输出的维度

```
nvinfer1::Dims getOutputDimensions(int index, const nvinfer1::Dims* inputs, int
nbInputDims);
```

3. 根据输入个数和输入类型，获得第index个输出的类型

```
nvinfer1::DataType getOutputDataType(int index, const nvinfer1::DataType* inputTypes, int
nbInputs) const;
```

Static Shape Plugin API

序列化和反序列化相关函数

1. 返回序列化时需要写多少字节到buffer中

```
size_t getSerializationSize() const;
```

2. 序列化函数，将plugin的参数权值写入到buffer中

```
void serialize(void* buffer) const;
```

3. 获得plugin的type和version，用于反序列化使用

```
const char* getPluginType() const;
const char* getPluginVersion() const;
```

Static Shape Plugin API

初始化、配置、销毁函数

初始化函数，在这个插件准备开始run之前执行。一般申请权值显存空间并copy权值

```
int initialize();
```

terminate函数就是释放initialize开辟的一些显存空间

```
void terminate();
```

释放整个plugin占用的资源

```
void destroy();
```

配置这个插件op，判断输入和输出类型数量是否正确

```
void configurePlugin(const nvinfer1::PluginTensorDesc* in, int nbInput,  const
nvinfer1::PluginTensorDesc* out, int nbOutput);
```

判断pos索引的输入/输出是否支持inOut[pos].format和inOut[pos].type指定的格式/数据类型

```
bool supportsFormatCombination(int pos, const nvinfer1::PluginTensorDesc* inOut, int
nbInputs, int nbOutputs) const;
```

Static Shape Plugin API

运行相关函数

1. 获得plugin所需要的显存大小。最好不要在plugin enqueue中使用cudaMalloc申请显存。

```
size_t getWorkspaceSize(int maxBatchSize) const;
```

2. inference函数

```
int enqueue(int batchSize, const void* const* inputs, void** outputs, void* workspace,
cudaStream_t stream);
```

Static Shape Plugin API

IPluginCreator 相关函数

获得pluginname和version，用于辨识creator

```
const char* getPluginName() const;
const char* getPluginVersion() const;
```

通过PluginFieldCollection去创建plugin
将op需要的权重和参数一个一个取出来，然后调用上文提到的第一个构造函数：

```
const nvinfer1::PluginFieldCollection* getFieldNames();
nvinfer1::IPluginV2* createPlugin(const char* name, const nvinfer1::PluginFieldCollection*
fc);
```

反序列化，调用反序列化那个构造函数，生成plugin

```
nvinfer1::IPluginV2* deserializePlugin(const char* name, const void* serialData, size_t
serialLength);
```

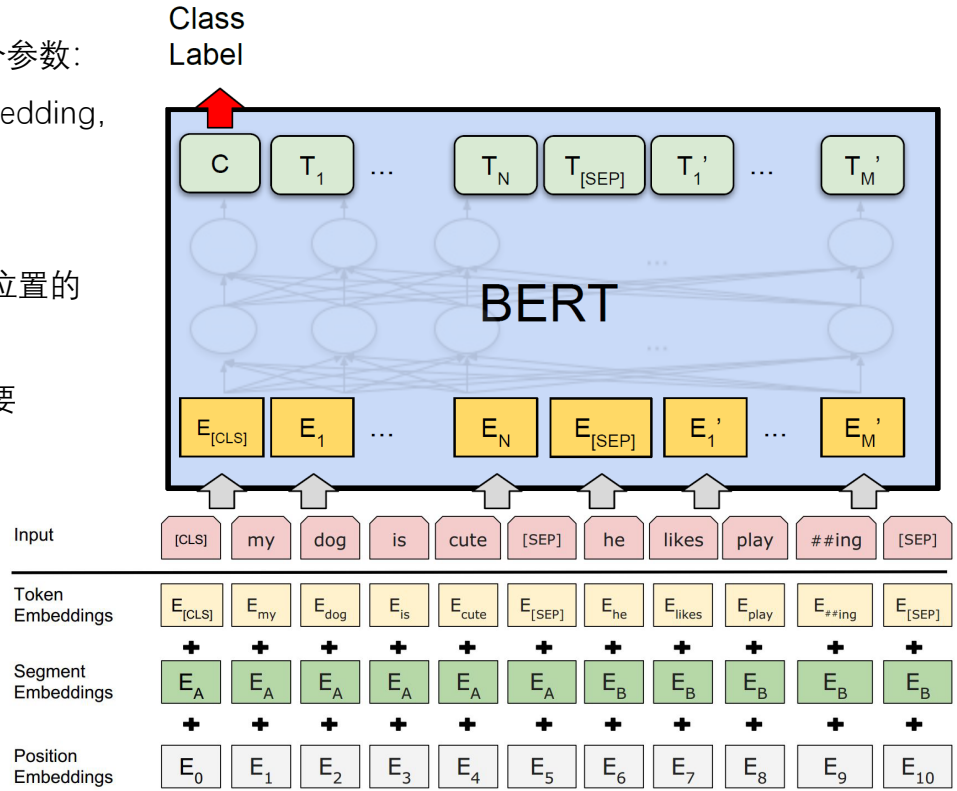
EmbLayerNormPlugin Static Shape Demo

EmbLayerNormPlugin 是 BERT 模型Embedding + Layernorm的合并

BERT 的 EmbLayerNormPlugin 层，主要有以下5个参数：

1. 三个 Embedding 参数矩阵，分别是词语的 Embedding，位置的 Embedding， token type 的 Embedding。

2. Embedding 操作除上面3个 embedding 做对应位置的求和，同时还要过一个 LayerNorm 操作，即对 Embedding 方向的维度做一个归一化，所以还需要 LayerNorm 的 beta 和 gamma 参数。



EmbLayerNormPlugin Static Shape Demo

TensorRT 5.0 BERT 官方demo

https://github.com/NVIDIA/TensorRT/blob/release/5.1/demo/BERT/plugins/embLayerNormPlugin.h

https://github.com/NVIDIA/TensorRT/blob/release/5.1/demo/BERT/plugins/embLayerNormPlugin.cu

Dynamic Shape Plugin API

跟 static shape 相比有差异的函数

static implicit(隐式) batch VS dynamic explicit(显式) batch

1. 根据输入个数和动态输入维度，获得第index个输出的动态维度

static

```
nvinfer1::Dims getOutputDimensions(int index, const nvinfer1::Dims* inputs, int
nbInputDims);
```

dynamic

```
nvinfer1::DimsExprs getOutputDimensions(int outputIndex, const nvinfer1::DimsExprs* inputs,
int nbInputs, nvinfer1::IExprBuilder& exprBuilder);
```

2. enqueue和getWorkspaceSize多了输入输出的信息、维度类型等

static

```
int enqueue(int batchSize, const void* const* inputs, void** outputs, void* workspace,
cudaStream_t stream);
```

dynamic

```
int enqueue(const nvinfer1::PluginTensorDesc* inputDesc, const nvinfer1::PluginTensorDesc*
outputDesc,const void* const* inputs, void* const* outputs, void* workspace, cudaStream_t
stream);
```

EmbLayerNormPlugin Dynamic Shape Demo

TensorRT 6.0 BERT 官方demo

https://github.com/NVIDIA/TensorRT/blob/release/6.0/demo/BERT/plugins/embLayerNormPlugin.h

https://github.com/NVIDIA/TensorRT/blob/release/6.0/demo/BERT/plugins/embLayerNormPlugin.cu

PluginCreator 注册

PluginCreator 注册

在加载NvInferRuntimeCommon.h头文件时，会得到一个getPluginRegistry，这里类中包含了所有已经注册了的IPluginCreator，在使用的时候通过getPluginCreator函数得到相应的IPluginCreator。

两种注册方式，本质上一样

1. 调用API进行注册

```
extern "C" TENSORRTAPI nvinfer1::IPluginRegistry* getPluginRegistry();
getPluginRegistry()->registerCreator(*pluginCreator, libNamespace);
```

2. 直接通过REGISTER_TENSORRT_PLUGIN来注册：

```
class PluginRegistrar {
public:
    PluginRegistrar() { getPluginRegistry()->registerCreator(instance, ""); }
private:
    T instance{};
};

#define REGISTER_TENSORRT_PLUGIN(name) \
    static nvinfer1::PluginRegistrar<name> pluginRegistrar##name {}
```

PluginCreator 注册

如何使用注册好的PluginCreator

```
class IPluginRegistry {
public:
    virtual bool registerCreator(IPluginCreator& creator, const char* pluginNamespace)
noexcept = 0;

    //!
    //! \brief Return all the registered plugin creators and the number of
    //! registered plugin creators. Returns nullptr if none found.
    //!
    virtual IPluginCreator* const* getPluginCreatorList(int* numCreators) const
noexcept = 0;

    virtual IPluginCreator* getPluginCreator(const char* pluginType, const char*
pluginVersion, const char* pluginNamespace = "") noexcept = 0;
```

TensorRT 如何 debug – debug plugin

TRT是闭源软件，API相对比较复杂。

1. 无论是使用API还是parser构建网络，模型转换完后，结果误差很大，怎么办？
2. 增加了自定义plugin 实现算子合并，结果对不上，怎么办？
3. 使用FP16 or INT8优化策略后，算法精确度掉了很多，怎么办？

TensorRT 如何 debug – debug plugin

推荐几种debug方法

1. 使用parser转换网络，使用dump API接口，查看网络结构是否对的上；
2. 使用了plugin，要写单元测试；
3. 通用办法，打印输出

（1）官方建议：将可疑层的输出设置为network output（比较累）；

（2）我的方法：增加一个debug plugin。

TensorRT_Tutorial：https://github.com/LitLeo/TensorRT_Tutorial

master/视频版资料/打造自己的plugin库示例-debug_plugin

深蓝学院
shenlanxueyuan.com

感谢聆听！

Thanks for Listening