MindSpore

MindSpore模型迁移实践

——以BERT迁移为例

作者：CQU弟中弟

目录

contents

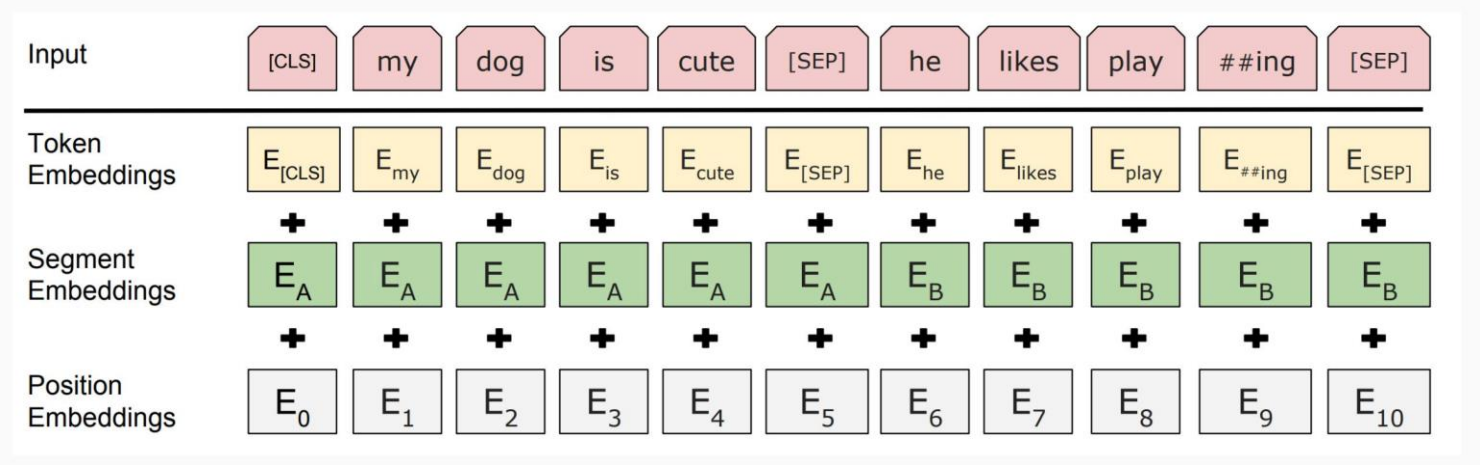
MindSpore

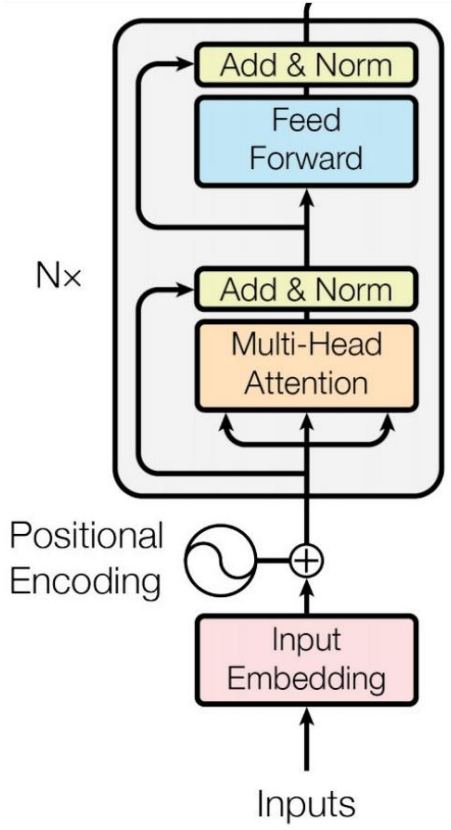
BERT模型分析

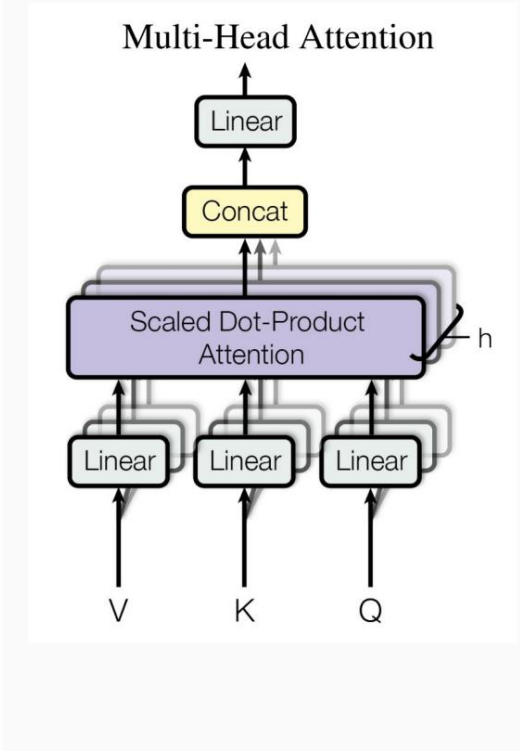
MindSpore

Transformer Encoder结构

- Positional embeddings
- Multi-head self attention
- FeedForward Layers
- LayerNorm and Residuals

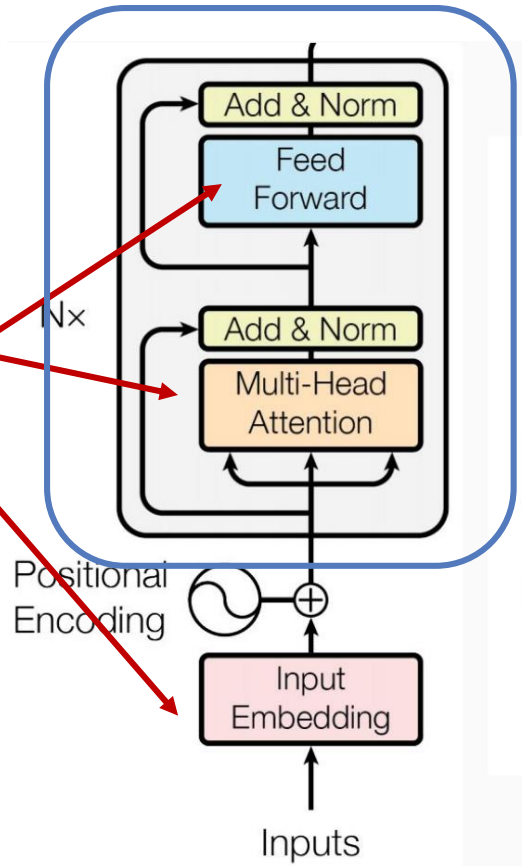


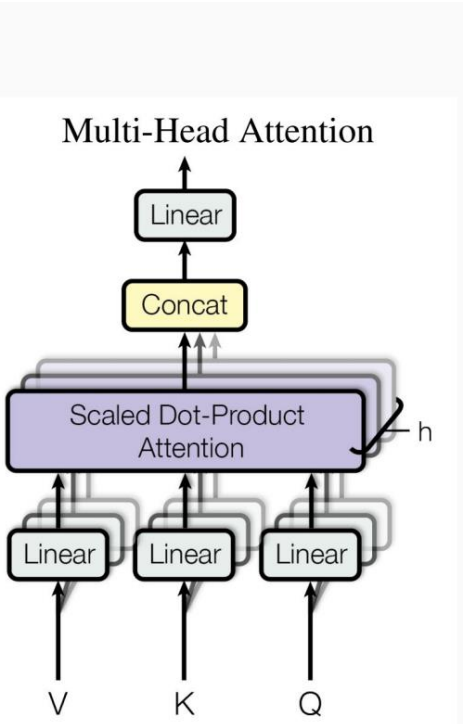




BERT模型分析

```
143  class BertEmbeddings(nn.Module):
173
174
175  class BertSelfAttention(nn.Module):
234
235
236  class BertSelfOutput(nn.Module):
248
249
250  class BertAttention(nn.Module):
285
286
287  class BertIntermediate(nn.Module):
300
301
302  class BertOutput(nn.Module):
314
315
316  class BertLayer(nn.Module):
330
331
332  class BertEncoder(nn.Module):
362
363
364  class BertPooler(nn.Module):
377
```





https://github.com/huggingface/transformers/blob/v2.0.0/transformers/modeling_bert.py

MindSpore算子映射

原则：简洁实现，兼顾性能。功能与初始化都需要对齐。

- nn层映射：
  - mindspore.nn
  - 继承nn.Cell自定义
- 算子映射优先级：
  - Tensor方法
  - ops.functional
  - mindspore.numpy

MindSpore算子映射

```python
class BertEmbeddings(nn.Module):
    """Construct the embeddings from word, position and token_type embeddings.
    """
                                                    BertLayerNorm = torch.nn.LayerNorm
    def __init__(self, config):
        super(BertEmbeddings, self).__init__()
        self.word_embeddings = nn.Embedding(config.vocab_size, config.hidden_size, padding_idx=0)
        self.position_embeddings = nn.Embedding(config.max_position_embeddings, config.hidden_size)
        self.token_type_embeddings = nn.Embedding(config.type_vocab_size, config.hidden_size)

        # self.LayerNorm is not snake-cased to stick with TensorFlow model variable name and be able to
        # any TensorFlow checkpoint file
        self.LayerNorm = BertLayerNorm(config.hidden_size, eps=config.layer_norm_eps)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

    def forward(self, input_ids, token_type_ids=None, position_ids=None):
        seq_length = input_ids.size(1)
        if position_ids is None:
            position_ids = torch.arange(seq_length, dtype=torch.long, device=input_ids.device)
            position_ids = position_ids.unsqueeze(0).expand_as(input_ids)
        if token_type_ids is None:
            token_type_ids = torch.zeros_like(input_ids)

        words_embeddings = self.word_embeddings(input_ids)
        position_embeddings = self.position_embeddings(position_ids)
        token_type_embeddings = self.token_type_embeddings(token_type_ids)

        embeddings = words_embeddings + position_embeddings + token_type_embeddings
        embeddings = self.LayerNorm(embeddings)
        embeddings = self.dropout(embeddings)
        return embeddings
```

MindSpore

```
import mindspore.numpy as mnp
import mindspore.ops as ops
Import mindspore.nn as nn
```

nn:
nn.Embedding
nn.LayerNorm
nn.Dropout

Tensor方法：
Tensor.size -> Tensor.shape
Tensor.unsqueeze -> Tensor.expand_dims
Tensor.expand_as -> Tensor.expand_as

ops:
torch.zeros_like -> ops.zeros_like

mnp:
torch.arange -> mnp.arange

MindSpore算子映射

```
self.query = nn.Linear(config.hidden_size, self.all_head_size)
self.key = nn.Linear(config.hidden_size, self.all_head_size)
self.value = nn.Linear(config.hidden_size, self.all_head_size)

self.dropout = nn.Dropout(config.attention_probs_dropout_prob)
```

```
# Take the dot product between "query" and "key" to get the raw attention scores.
attention_scores = torch.matmul(query_layer, key_layer.transpose(-1, -2))
attention_scores = attention_scores / math.sqrt(self.attention_head_size)
if attention_mask is not None:
    # Apply the attention mask is (precomputed for all layers in BertModel forward() function)
    attention_scores = attention_scores + attention_mask
```

```
# Normalize the attention scores to probabilities.
attention_probs = nn.Softmax(dim=-1)(attention_scores)
```

```
# Mask heads if we want to
if head_mask is not None:
    attention_probs = attention_probs * head_mask

context_layer = torch.matmul(attention_probs, value_layer)

context_layer = context_layer.permute(0, 2, 1, 3).contiguous()
new_context_layer_shape = context_layer.size()[:-2] + (self.all_head_size,)
context_layer = context_layer.view(*new_context_layer_shape)
```

MindSpore

```
import mindspore.numpy as mnp
import mindspore.ops as ops
Import mindspore.nn as nn
```

nn:
nn.Dense
nn.Softmax
nn.Dropout

Tensor方法:
Tensor.transpose -> Tensor.swapaxes
Tensor.permute -> Tensor.transpose
Tensor.view -> Tensor.view

ops:
torch.matmul -> ops.matmul
math.sqrt -> ops.sqrt(ops.scalar_to_tensor)

算子功能对齐

MindSpore

```python
def gelu(x):
    """ Original Implementation of the gelu activation function in Google Bert repo when initialy created.
        For information: OpenAI GPT's gelu is slightly different (and gives slightly different results):
        0.5 * x * (1 + torch.tanh(math.sqrt(2 / math.pi) * (x + 0.044715 * torch.pow(x, 3))))
        Also see https://arxiv.org/abs/1606.08415
    """
    return x * 0.5 * (1.0 + torch.erf(x / math.sqrt(2.0)))

def gelu_new(x):
    """ Implementation of the gelu activation function currently in Google Bert repo (identical to OpenAI GPT).
        Also see https://arxiv.org/abs/1606.08415
    """
    return 0.5 * x * (1 + torch.tanh(math.sqrt(2 / math.pi) * (x + 0.044715 * torch.pow(x, 3))))

def swish(x):
    return x * torch.sigmoid(x)


ACT2FN = {"gelu": gelu, "relu": torch.nn.functior
```

class mindspore.nn.GELU(*approximate=True*)

参数：

approximate (bool): 是否启用approximation，默认值：True。如果approximate的值为True，则高斯误差线性激活函数为:

$0.5 * x * (1 + tanh(sqrt(2/pi) * (x + 0.044715 * x^3)))$,

否则为： $x * P(X <= x) = 0.5 * x * (1 + erf(x/sqrt(2)))$, where P(X) ~ N(0, 1) 。

权重初始化对齐

MindSpore

- Pytorch: nn.Linear

```
def reset_parameters(self) -> None:
    init.kaiming_uniform_(self.weight, a=math.sqrt(5))
    if self.bias is not None:
        fan_in, _ = init._calculate_fan_in_and_fan_out(self.weight)
        bound = 1 / math.sqrt(fan_in)
        init.uniform_(self.bias, -bound, bound)
```

使用mindspore.common.initializer

- Mindspore: nn.Dense

- **weight_init** (Union[Tensor, str, Initializer, numbers.Number]) – The trainable weight_init parameter. The dtype is same as input x. The values of str refer to the function *initializer*. Default: 'normal'.
- **bias_init** (Union[Tensor, str, Initializer, numbers.Number]) – The trainable bias_init parameter. The dtype is same as input x. The values of str refer to the function *initializer*. Default: 'zeros'.

```
class Dense(nn.Dense):
    def __init__(self, in_channels, out_channels, weight_init=None, bias_init=None, has_bias=True, activation=None):
        if weight_init is None:
            weight_init = initializer(HeUniform(math.sqrt(5)), (out_channels, in_channels))
        if bias_init is None:
            fan_in, _ = _calculate_fan_in_and_fan_out((out_channels, in_channels))
            bound = 1 / math.sqrt(fan_in)
            bias_init = initializer(Uniform(bound), (out_channels))
        super().__init__(in_channels, out_channels, weight_init=weight_init, bias_init=bias_init, has_bias=has_bias, activation=activation)
```

模型迁移精度验证

MindSpore

- Checkpoint迁移精度验证：
  - 验证模型结构正确
  - 验证模型算子计算可以精度达标（正向）
- 模型训练精度验证

Checkpoint转换(Pytorch2MindSpore)

MindSpore

```
ms_ckpt = []
state_dict = torch.load(pth_file, map_location=torch.device('cpu'))

for k, v in state_dict.items():
    if 'LayerNorm' in k:
        k = k.replace('LayerNorm', 'layer_norm')
    if 'layer_norm' in k:
        if '.weight' in k:
            k = k.replace('.weight', '.gamma')
        if '.bias' in k:
            k = k.replace('.bias', '.beta')
    if 'embeddings' in k:
        k = k.replace('weight', 'embedding_table')
    if 'self' in k:
        k = k.replace('self', 'self_attn')
    ms_ckpt.append({'name': k, 'data': Tensor(v.numpy())})

ms_ckpt_path = pth_file.replace('.bin','.ckpt')
if not os.path.exists(ms_ckpt_path):
    try:
        save_checkpoint(ms_ckpt, ms_ckpt_path)
    except:
        raise RuntimeError(f'Save checkpoint to {ms_ckpt_path} failed, please checkout the path.')
```

- Checkpoint为Dict，转换需要保证对应层的变量名一致
- MindSpore不支持self作为变量名

Checkpoint转换(Pytorch2MindSpore)

MindSpore

```
ms_ckpt = []
state_dict = torch.load(pth_file, map_location=torch.device('cpu'))

for k, v in state_dict.items():
    if 'LayerNorm' in k:
        k = k.replace('LayerNorm', 'layer_norm')
    if 'layer_norm' in k:
        if '.weight' in k:
            k = k.replace('.weight', '.gamma')
        if '.bias' in k:
            k = k.replace('.bias', '.beta')
    if 'embeddings' in k:
        k = k.replace('weight', 'embedding_table')
    if 'self' in k:
        k = k.replace('self', 'self_attn')
    ms_ckpt.append({'name': k, 'data': Tensor(v.numpy())})

ms_ckpt_path = pth_file.replace('.bin','.ckpt')
if not os.path.exists(ms_ckpt_path):
    try:
        save_checkpoint(ms_ckpt, ms_ckpt_path)
    except:
        raise RuntimeError(f'Save checkpoint to {ms_ckpt_path} failed, please checkout the path.')
```

- Checkpoint为Dict，转换需要保证对应层的变量名一致
- MindSpore不支持self作为变量名

模型Checkpoint加载

MindSpore

```
# load ckpt
try:
    param_dict = load_checkpoint(model_file)
except:
    raise ValueError(f"File {model_file} is not a checkpoint file, please check the path.")

param_not_load = load_param_into_net(model, param_dict)
if len(param_not_load) == len(model.trainable_params()):
    raise KeyError(f"The following weights in model are not found: {param_not_load}")

return model
```

- 必须保证load_param_into_net的返回值为空

整网精度对比

MindSpore

```python
def test_modeling_bert_with_ckpt_pynative(self):
    context.set_context(mode=context.PYNATIVE_MODE)
    model = BertModel.load('bert-base-uncased')
    model.set_train(False)
    input_ids = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] + [0] * 500

    ms_input_ids = Tensor(input_ids, mindspore.int32).reshape(1, -1)
    outputs, pooled = model(ms_input_ids)

    pt_model = ptBertModel.from_pretrained('bert-base-uncased')
    pt_model.eval()
    pt_input_ids = torch.IntTensor(input_ids).reshape(1, -1)
    outputs_pt, pooled_pt = pt_model(input_ids=pt_input_ids)

    assert np.allclose(outputs.asnumpy(), outputs_pt.detach().numpy(), atol=1e-5)
    assert np.allclose(pooled.asnumpy(), pooled_pt.detach().numpy(), atol=1e-5)
```

- 设置同样的输入
- MindSpore实现与Pytorch实现同时加载相同的checkpoint
- 验证输出误差

整网精度对比——定位模块

MindSpore

Assert == False

- 切换Pynative模式
- 找到model的类

```
class BertModel(BertPretrainedCell):
    """
    def __init__(self, config):
        super().__init__(config)
        self.embeddings = BertEmbeddings(config)
        self.encoder = BertEncoder(config)
        self.pooler = BertPooler(config)
        self.num_hidden_layers = self.config.num_hidden_layers
```

- 修改相同模块输出

```
    embedding_output = self.embeddings(input_ids, position_ids=position_ids, token_type_ids=token_type_ids)
    return embedding_output
    encoder_outputs = self.encoder(embedding_output,
                                   extended_attention_mask,
                                   head_mask=head_mask)
    sequence_output = encoder_outputs[0]
    pooled_output = self.pooler(sequence_output)

    outputs = (sequence_output, pooled_output,) + encoder_outputs[1:]  # add hidden_states and attentions if they are here
    return outputs  # sequence_output, pooled_output, (hidden_states), (attentions)
```

THANK YOU