# Unified Vision-Language Pre-Training for Image Captioning and VQA

LoongCat

2024 年 5 月 14 日

## 1 Structure

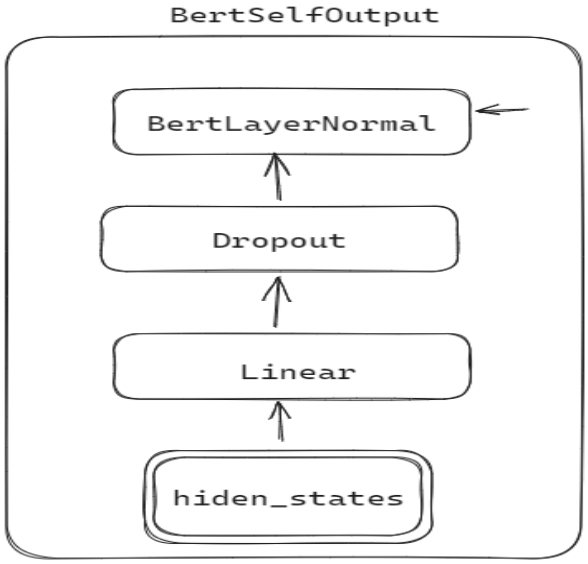


图 1: BertSelfOutput

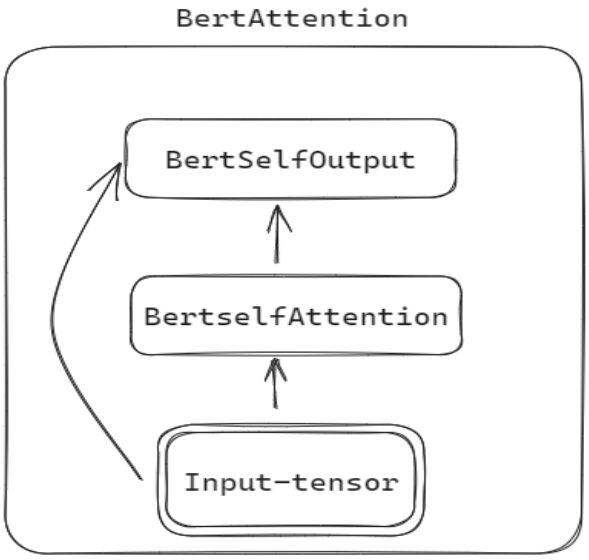


图 2: BertAttention

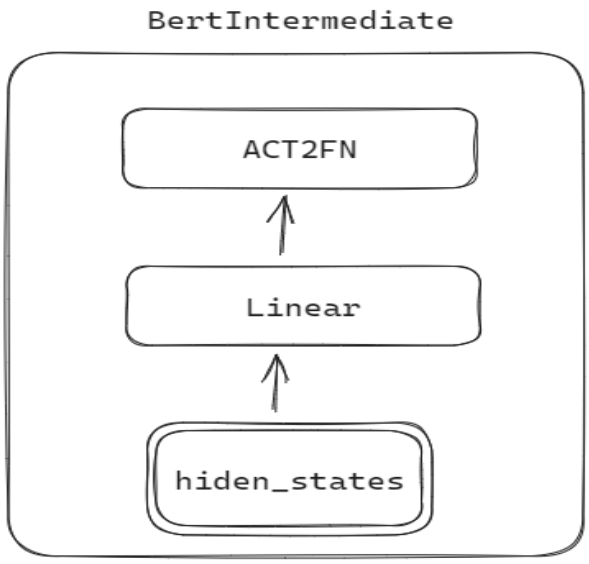


图 3: BertIntermediate

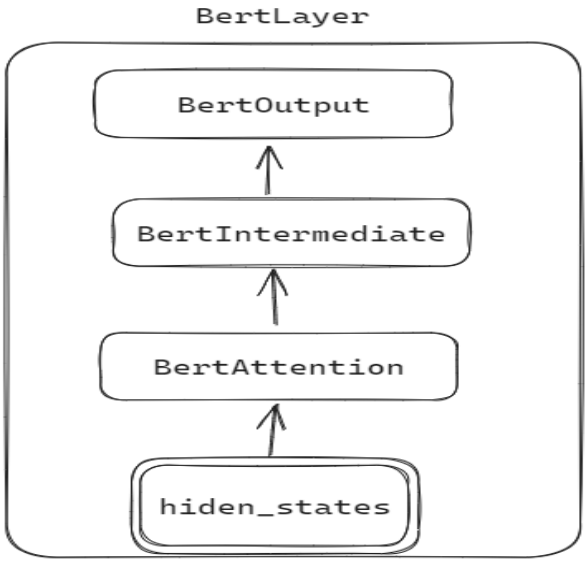


图 4: BertLayer

BertConfig类代码：关注：num_hidden_layers=12，有12个Layer

```
class BertConfig(object):
    """Configuration class to store the configuration of a BertModel.
    """

    def __init__(self,
                 vocab_size_or_config_json_file,
                 hidden_size=768,
                 num_hidden_layers=12,
                 num_attention_heads=12,
                 intermediate_size=3072,
                 hidden_act="gelu",
                 hidden_dropout_prob=0.1,
                 attention_probs_dropout_prob=0.1,
                 max_position_embeddings=512,
                 type_vocab_size=2,
                 relax_projection=0,
                 initializer_range=0.02,
                 task_idx=None,
                 fp32_embedding=False,
                 label_smoothing=None):
```

BertEncoder类代码：关注：self.layer

```
class BertEncoder(nn.Module):
    def __init__(self, config):
        super(BertEncoder, self).__init__()
        layer = BertLayer(config)
        self.layer = nn.ModuleList([copy.deepcopy(layer)
                                    for _ in range(config.num_hidden_layers)])

    def forward(self, hidden_states, attention_mask, prev_embedding=None,
        prev_encoded_layers=None, output_all_encoded_layers=True):
        assert (prev_embedding is None) == (prev_encoded_layers is None), \
                    "history embedding and encoded layer must be simultanously given."
        all_encoder_layers = []
        if (prev_embedding is not None) and (prev_encoded_layers is not None):
            history_states = prev_embedding
            for i, layer_module in enumerate(self.layer):
                    hidden_states = layer_module(
                    hidden_states, attention_mask, history_states=history_states)
                    if output_all_encoded_layers:
                    all_encoder_layers.append(hidden_states)
                    if prev_encoded_layers is not None:
                    history_states = prev_encoded_layers[i]
        else:
            for layer_module in self.layer:
                    hidden_states = layer_module(hidden_states, attention_mask)
                    if output_all_encoded_layers:
                    all_encoder_layers.append(hidden_states)
        if not output_all_encoded_layers:
            all_encoder_layers.append(hidden_states)
        return all_encoder_layers
```

$hidden\_states = layer\_module(hidden\_states, attention\_mask)$ 这行就是表明:

$$H_l = Transformer(H_{l-1}), l \in [1, L]$$

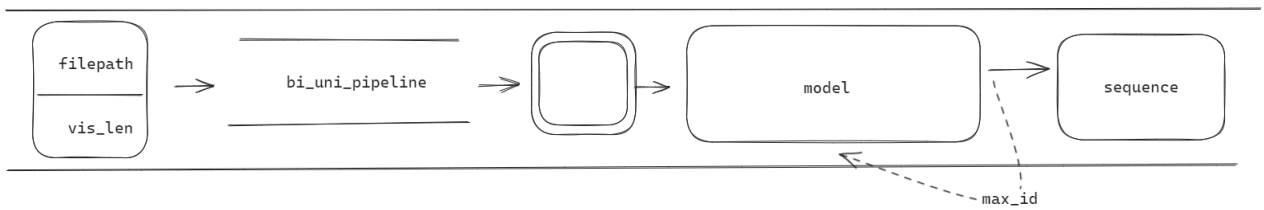


图 5: decode_img2txt文件中提供了i2t的方法，首先建立一个preprocess管道bi_uni_pipeline（来自于s2s_loader的Preprocess4Seq2seqDecoder）用来预处理输入的图像管道的输入：图像路径与图像大小管道的输出：input_id,..position_ids,..input_mask等管道的输出用instances来接收，再次经过处理之后，输送给model模型，为modeling的BertForSeq2SeqDecoder类对象， model输入：vis_feat,vis_pe,..attention_mask等 model输出：max_ids,max_probs

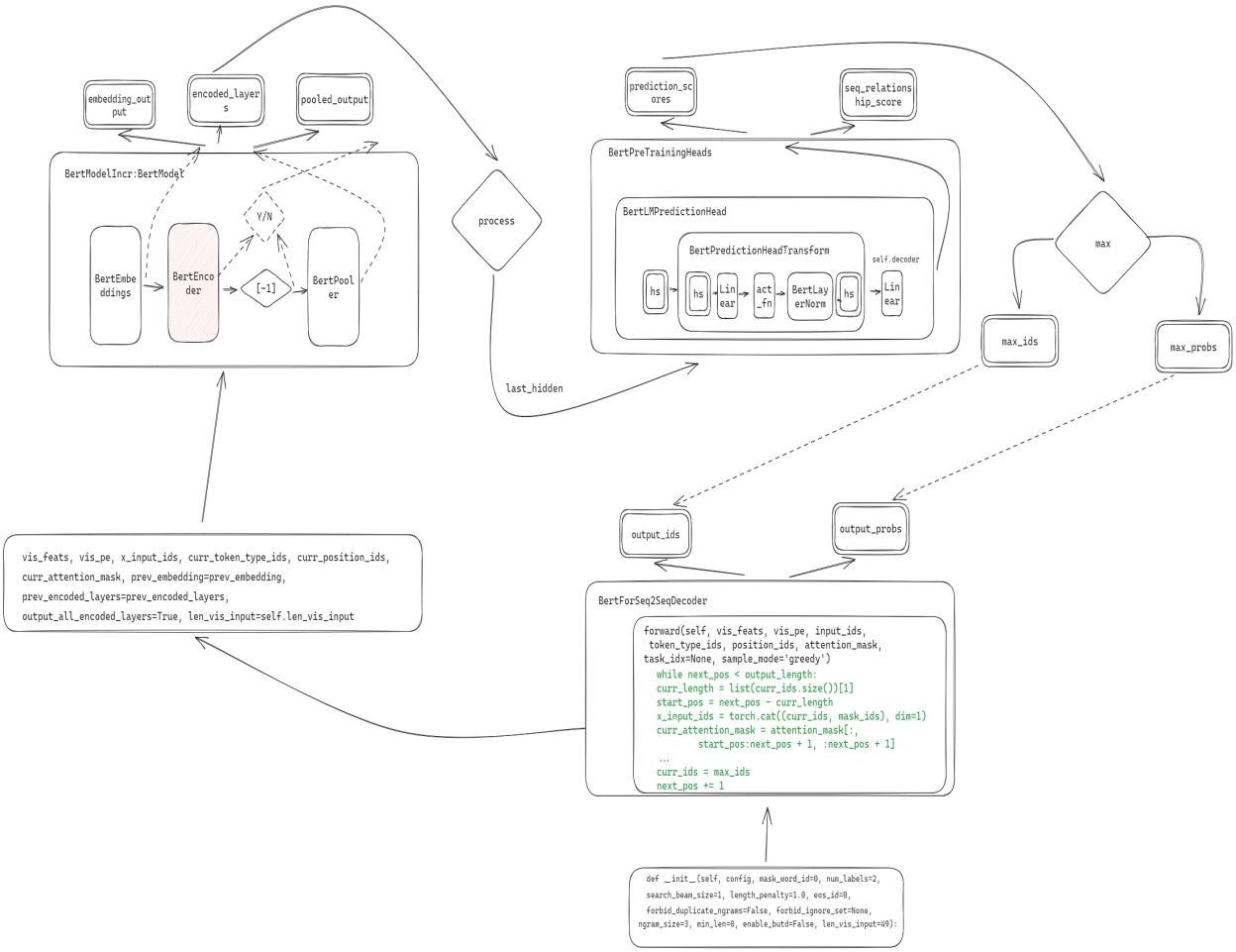


图 6: Bert4S2SDecoder, 生成过程由 next_pos <output_length: 该语句控制，没体现出来 STOP 标记的作用

decoder_img2txt.py:

```
#一个预处理图像的管道
#输入图像路径和图像大小
#返回input_ids, segment_ids(token_type_ids), position_ids, input_mask, task_idx, img, vis_pe
bi_uni_pipeline = []
bi_uni_pipeline.append(seq2seq_loader.Preprocess4Seq2seqDecoder(list(tokenizer.vocab.keys()), tokenizer.convert_tokens_to_ids, args.max_seq_length,
    max_tgt_length=args.max_tgt_length, new_segment_ids=args.new_segment_ids,
    mode='s2s', len_vis_input=args.len_vis_input, enable_butd=args.enable_butd,
    region_bbox_file=args.region_bbox_file,
        region_det_file_prefix=args.region_det_file_prefix))
...

#用来解码的模型
#输入时
model = BertForSeq2SeqDecoder.from_pretrained(args.bert_model,
     max_position_embeddings=args.max_position_embeddings,
         config_path=args.config_path,
     state_dict=model_recover, num_labels=cls_num_labels,
     type_vocab_size=type_vocab_size, task_idx=3, mask_word_id=mask_word_id,
     search_beam_size=args.beam_size, length_penalty=args.length_penalty,
     eos_id=eos_word_ids, forbid_duplicate_ngrams=args.forbid_duplicate_ngrams,
     forbid_ignore_set=forbid_ignore_set, ngram_size=args.ngram_size,
         min_len=args.min_len,
     enable_butd=args.enable_butd, len_vis_input=args.len_vis_input)

...

for src in img_dat:
     src_tk = os.path.join(args.image_root, src.get('filepath', 'trainval'),
         src['filename'])
     input_lines.append((img_idx, imgid, src_tk))


...

_chunk = input_lines[next_i:next_i + args.batch_size]
buf = [x[2] for x in _chunk]
# instance 其实就是图像路径和输入图像大小的一个元组
```

```
35          for instance in [(x, args.len_vis_input) for x in buf]:
36            #用 instances 来接收图像预处理的输出
37            for proc in bi_uni_pipeline:
38                instances.append(proc(instance))
39
40          ...
41
42          input_ids, token_type_ids, position_ids, input_mask, task_idx, img, vis_pe = batch
43
44          #将管道的输出处理之后送入解码模型
45          #输出 max_id 与max_prob
46          traces = model(conv_feats, vis_pe, input_ids, token_type_ids,
47                               position_ids, input_mask, task_idx=task_idx)
48
49          #将返回的 max_id 转换为tokens
50          output_buf = tokenizer.convert_ids_to_tokens(w_ids)
51          output_tokens = []
52          #将非标记 tokens 提取出来
53          for t in output_buf:
54               if t in ("[SEP]", "[PAD]"):
55                    break
56               output_tokens.append(t)
57          # tokens 变句子?
58          output_sequence = ' '.join(detokenize(output_tokens))
59          #记录每个图片对应的句子描述id
60          output_lines[buf_id[i]] = output_sequence
61          #图像标记序号为 tup[1] 的描述为output_lines[imd_idx]
62          predictions = [{'image_id': tup[1], 'caption': output_lines[img_idx]} for img_idx,
                tup in enumerate(input_lines)]
```

seq2seq_loader.py:

```
        #全1 的下三角矩阵, 创建 MASKED 矩阵用
        self._tril_matrix = torch.tril(torch.ones(
             (max_len, max_len), dtype=torch.long))


        def __call__(self, instance):
          img_path, max_a_len = instance[:2]
          tokens_a = ['[UNK]'] * self.len_vis_input

          # Add Special Tokens
          padded_tokens_a = ['[CLS]'] + tokens_a + ['[SEP]']
          tokens = padded_tokens_a
          if self.new_segment_ids:
              segment_ids = [4]*(len(padded_tokens_a)) \
                  + [5]*(max_len_in_batch - len(padded_tokens_a))
          else:
              segment_ids = [0]*(len(padded_tokens_a)) \
                  + [1]*(max_len_in_batch - len(padded_tokens_a))
          ...

          position_ids = []
          for i in range(len(tokens_a) + 2):
                 position_ids.append(i)
          for i in range(len(tokens_a) + 2, max_a_len + 2):
                 position_ids.append(0)
          for i in range(max_a_len + 2, max_len_in_batch):
                 position_ids.append(i - (max_a_len + 2) + len(tokens_a) + 2)

          # Token Indexing
          input_ids = self.indexer(tokens)

          # Zero Padding
          #首先创建一个全 0 的矩阵
          input_mask = torch.zeros(
                 max_len_in_batch, max_len_in_batch, dtype=torch.long)
          #图像区域填充 1
          input_mask[:, :len(tokens_a)+2].fill_(1)

          second_st, second_end = len(padded_tokens_a), max_len_in_batch
```

```
40
41          #句子部分拷贝全 1 矩阵的句子部分 tokens 长度的下三角矩阵
42          input_mask[second_st:second_end, second_st:second_end].copy_(
43                  self._tril_matrix[:second_end-second_st, :second_end-second_st])
44
45          img = torch.from_numpy(region_feat_f[img_id][:]).float()
46          vis_pe = torch.from_numpy(region_bbox_f[img_id][:])
47
48          # lazy normalization of the coordinates...
49          w_est = torch.max(vis_pe[:, [0, 2]])*1.+1e-5
50          h_est = torch.max(vis_pe[:, [1, 3]])*1.+1e-5
51          vis_pe[:, [0, 2]] /= w_est
52          vis_pe[:, [1, 3]] /= h_est
53          rel_area = (vis_pe[:, 3]-vis_pe[:, 1])*(vis_pe[:, 2]-vis_pe[:, 0])
54          rel_area.clamp_(0)
55          vis_pe = torch.cat((vis_pe[:, :4], rel_area.view(-1, 1), vis_pe[:, 5:]), -1) #
                confident score
56          normalized_coord = F.normalize(vis_pe.data[:, :5]-0.5, dim=-1)
57          vis_pe = torch.cat((F.layer_norm(vis_pe, [6]), \
58                  F.layer_norm(cls_label, [1601])), dim=-1) # 1601 hard coded...
59
60          return (input_ids, segment_ids, position_ids, input_mask, self.task_idx, img,
                vis_pe)
```

modeling.py:BertForSeq2SeqDecoder

```
        #输出隐藏层的解码器，输入为：图像特征 ， word_id ，attention_mask ，参数矩阵等
        self.bert = BertModelIncr(config)
        #输出预测结果的解码器，输入为隐藏层状态
        self.cls = BertPreTrainingHeads(
            config, self.bert.embeddings.word_embeddings.weight, num_labels=num_labels)


        def forward(self, vis_feats, vis_pe, input_ids, token_type_ids, position_ids,
             attention_mask, task_idx=None, sample_mode='greedy'):

            vis_feats = self.vis_embed(vis_feats) # image region features
            vis_pe = self.vis_pe_embed(vis_pe) # image region positional encodings

            ...
            mask_ids = input_ids[:, :1] * 0 + self.mask_word_id
            next_pos = list(input_ids.size())[1]
            output_length = list(token_type_ids.size())[1]

            while next_pos < output_length:
                curr_length = list(curr_ids.size())[1]
                start_pos = next_pos - curr_length
                #把当前的输出当做输入来产生下一个词
                x_input_ids = torch.cat((curr_ids, mask_ids), dim=1)
                curr_token_type_ids = token_type_ids[:, start_pos:next_pos + 1]
                #实时更新传入的 MASKED 矩阵
                curr_attention_mask = attention_mask[:,
                    start_pos:next_pos + 1, :next_pos + 1]
                #更新矩阵pos
                curr_position_ids = position_ids[:, start_pos:next_pos + 1]
                #将图像与当前产生的 token 送入decoer 模块，产生所有hidden_states 层的输出
                new_embedding, new_encoded_layers, _ = \
                    self.bert(vis_feats, vis_pe, x_input_ids, curr_token_type_ids,
                        curr_position_ids,
                    curr_attention_mask, prev_embedding=prev_embedding,
                    prev_encoded_layers=prev_encoded_layers,
                    output_all_encoded_layers=True, len_vis_input=self.len_vis_input)

                #只要最后一个隐藏层的输出
                last_hidden = new_encoded_layers[-1][:, -1:, :]
```

```
38
39              #把最后一个隐藏层输出送入预测模块，产生预测结果
40              prediction_scores, _ = self.cls(
41                  last_hidden, None, task_idx=task_idx)
42
43              #根据结果产生 max_id max_probs
44              if sample_mode == 'greedy':
45                  max_probs, max_ids = torch.max(prediction_scores, dim=-1)
46              elif sample_mode == 'sample':
47                  prediction_scores.squeeze_(1)
48                  prediction_probs = F.softmax(prediction_scores, dim=-1).detach()
49                  max_ids = torch.multinomial(prediction_probs, num_samples=1,
50                      replacement=True)
51                  max_probs = torch.gather(F.log_softmax(prediction_scores, dim=-1),
52                      1, max_ids) # this should be logprobs
53              else:
54                  raise NotImplementedError
55              output_ids.append(max_ids)
56              output_probs.append(max_probs)
57
58              #更新参数矩阵??
59              if prev_embedding is None:
60                  prev_embedding = new_embedding[:, :-1, :]
61              else:
62                  prev_embedding = torch.cat(
63                      (prev_embedding, new_embedding[:, :-1, :]), dim=1)
64              if prev_encoded_layers is None:
65                  prev_encoded_layers = [x[:, :-1, :]
66                                         for x in new_encoded_layers]
67              else:
68                  prev_encoded_layers = [torch.cat((x[0], x[1][:, :-1, :]), dim=1)
69                                         for x in zip(prev_encoded_layers, new_encoded_layers)]
70              #当前输出作为下轮输入
71              curr_ids = max_ids
72              next_pos += 1
73          return torch.cat(output_ids, dim=1), torch.cat(output_probs, dim=1)
```

decode_img2txt文件中提供了i2t的方法，首先建立一个preprocess管道bi_uni_pipeline（来自于s2s_loader的Preprocess4Seq2seqDecoder）用来预处理输入的图像

管道的输入：图像路径与图像大小

管道的输出：input_id,..position_ids,..input_mask等

管道的输出用instances来接收，再次经过处理之后，输送给model模型，为modeling的BertForSeq2SeqDecoder类对象，

model输入：vis_feat,vis_pe,..attention_mask等

model输出：max_ids,max_probs