Transformers Classification Cheat Sheet

by aoi-dev via cheatography.com/160438/cs/38602/

The encoder branch

BERT	BERT is pretrained with the two objectives of predicting masked tokens in texts and determining if one text passage is likely to follow another. The former task is called masked language modeling (MLM) and the latter next sentence prediction (NSP).
DistilBERT	Although BERT delivers great results, it’s size can make it tricky to deploy in environments where low latencies are required. By using a technique known as knowledge distillation during pretraining, DistilBERT achieves 97% of BERT’s performance while using 40% less memory and being 60% faster. You can find more details on knowledge distillation in Chapter 8.
RoBERTa	A study following the release of BERT revealed that its performance can be further improved by modifying the pretraining scheme. RoBERTa is trained longer, on larger batches with more training data, and it drops the NSP task. Together, these changes significantly improve its performance compared to the original BERT model.
XLM	Several pretraining objectives for building multilingual models were explored in the work on the cross-lingual language model (XLM), including the autoregressive language modeling from GPT-like models and MLM from BERT. In addition, the authors of the paper on XLM pretraining introduced translation language modeling (TLM), which is an extension of MLM to multiple language inputs. Experimenting with these pretraining tasks, they achieved state-of-the-art results on several multilingual NLU benchmarks as well as on translation tasks.
XLM-RoBERTa	Following the work of XLM and RoBERTa, the XLM-RoBERTa or XLM-R model takes multilingual pretraining one step further by massively upscaling the training data. Using the Common Crawl corpus, its developers created a dataset with 2.5 terabytes of text; they then trained an encoder with MLM on this dataset. Since the dataset only contains data without parallel texts (i.e., translations), the TLM objective of XLM was dropped. This approach beats XLM and multilingual BERT variants by a large margin, especially on low-resource languages.
ALBERT	The ALBERT model introduced three changes to make the encoder architecture more efficient. First, it decouples the token embedding dimension from the hidden dimension, thus allowing the embedding dimension to be small and thereby saving parameters, especially when the vocabulary gets large. Second, all layers share the same parameters, which decreases the number of effective parameters even further. Finally, the NSP objective is replaced with a sentence-ordering prediction: the model needs to predict whether or not the order of two consecutive sentences was swapped rather than predicting if they belong together at all. These changes make it possible to train even larger models with fewer parameters and reach superior performance on NLU tasks.

The encoder branch (cont)

ELECTRA	One limitation of the standard MLM pretraining objective is that at each training step only the representations of the masked tokens are updated, while the other input tokens are not. To address this issue, ELECTRA uses a two-model approach: the first model (which is typically small) works like a standard masked language model and predicts masked tokens. The second model, called the discriminator, is then tasked to predict which of the tokens in the first model’s output were originally masked. Therefore, the discriminator needs to make a binary classification for every token, which makes training 30 times more efficient. For downstream tasks the discriminator is fine-tuned like a standard BERT model.
DeBERTa	The DeBERTa model introduces two architectural changes. First, each token is represented as two vectors: one for the content, the other for relative position. By disentangling the tokens’ content from their relative positions, the self-attention layers can better model the dependency of nearby token pairs. On the other hand, the absolute position of a word is also important, especially for decoding. For this reason, an absolute position embedding is added just before the softmax layer of the token decoding head. DeBERTa is the first model (as an ensemble) to beat the human baseline on the SuperGLUE benchmark, a more difficult version of GLUE consisting of several subtasks used to measure NLU performance.

Bert

```
BertModel(
    (em bed dings): BertEm bed dings(
        (wo rd_ emb edd ings): Embedd ing (30522, 768, paddin g_i dx=0)
        (po sit ion _em bed dings): Embedd ing (512, 768)
        (to ken _ty pe_ emb edd ings): Embedd ing(2, 768)
        (La yer Norm): LayerN orm ((7 68,), eps=1e-12, elemen twi se_ aff ine =True)
        (dr opout): Dropou t(p =0.1, inplac e=F alse)
   )
    (en coder): BertEn coder(
        (la yer): Module List(
            (0-11): 12 x BertLayer(
                (at ten tion): BertAt ten tion(
                    (self): BertSe lfA tte ntion(
                        (qu ery): Linear (in _fe atu res =768, out_fe atu res =768, bias=True)
                        (key): Linear (in _fe atu res =768, out_fe atu res =768, bias=True)
                        (va lue): Linear (in _fe atu res =768, out_fe atu res =768, bias=True)
                        (dr opout): Dropou t(p =0.1, inplac e=F alse)
                   )
                    (ou tput): BertSe lfO utput(
                        (de nse): Linear (in _fe atu res =768, out_fe atu res =768, bias=True)
                        (La yer Norm): LayerN orm ((7 68,), eps=1e-12, elemen twi se_ aff ine =True)
                        (dr opout): Dropou t(p =0.1, inplac e=F alse)
```

Bert (cont)

```
>          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
)
```

The Decoder Branch

GPT	The introduction of GPT combined two key ideas in NLP: the novel and efficient transformer decoder architecture, and transfer learning. In that setup, the model was pretrained by predicting the next word based on the previous ones. The model was trained on the BookCorpus and achieved great results on downstream tasks such as classification.
GPT-2	Inspired by the success of the simple and scalable pretraining approach, the original model and training set were upscaled to produce GPT-2. This model is able to produce long sequences of coherent text. Due to concerns about possible misuse, the model was released in a staged fashion, with smaller models being published first and the full model later.
CTRL	Models like GPT-2 can continue an input sequence (also called a prompt). However, the user has little control over the style of the generated sequence. The Conditional Transformer Language (CTRL) model addresses this issue by adding “control tokens” at the beginning of the sequence. These allow the style of the generated text to be controlled, which allows for diverse generation.

The Decoder Branch (cont)

GPT-3	Following the success of scaling GPT up to GPT-2, a thorough analysis on the behavior of language models at different scales revealed that there are simple power laws that govern the relation between compute, dataset size, model size, and the performance of a language model. Inspired by these insights, GPT-2 was upscaled by a factor of 100 to yield GPT-3, with 175 billion parameters. Besides being able to generate impressively realistic text passages, the model also exhibits few-shot learning capabilities: with a few examples of a novel task such as translating text to code, the model is able to accomplish the task on new examples. OpenAI has not open-sourced this model, but provides an interface through the OpenAI API.
GPT-Neo/GPT-J-6B	GPT-Neo and GPT-J-6B are GPT-like models that were trained by EleutherAI, a collective of researchers who aim to re-create and release GPT-3 scale models. The current models are smaller variants of the full 175-billion-parameter model, with 1.3, 2.7, and 6 billion parameters, and are competitive with the smaller GPT-3 models OpenAI offers. The final branch in the transformers tree of life is the encoder-decoder models. Let’s take a look.

GPT2

```
GPT2Model(
    (wte): Embedd ing (50257, 768)
    (wpe): Embedd ing (1024, 768)
    (drop): Dropou t(p =0.1, inplac e=F alse)
   (h): Module List(
        (0-11): 12 x GPT2Block(
            (ln_1): LayerN orm ((7 68,), eps=1e-05, elemen twi se_ aff ine =True)
            (attn): GPT2At ten tion(
                (c_ attn): Conv1D()
                (c_ proj): Conv1D()
                (at tn_ dro pout): Dropou t(p =0.1, inplac e=F alse)
                (re sid _dr opout): Dropou t(p =0.1, inplac e=F alse)
           )
            (ln_2): LayerN orm ((7 68,), eps=1e-05, elemen twi se_ aff ine =True)
            (mlp): GPT2MLP(
                (c_fc): Conv1D()
                (c_ proj): Conv1D()
                (act): NewGEL UAc tiv ation()
                (dr opout): Dropou t(p =0.1, inplac e=F alse)
           )
       )
   )
    (ln_f): LayerN orm ((7 68,), eps=1e-05, elemen twi se_ aff ine =True)
)
```

The Encoder-Decoder Branch

T5	The T5 model unifies all NLU and NLG tasks by converting them into text-to-text tasks. All tasks are framed as sequence-to-sequence tasks, 20 21 22 23 where adopting an encoder-decoder architecture is natural. For text classification problems, for example, this means that the text is used as the encoder input and the decoder has to generate the label as normal text instead of a class. We will look at this in more detail in Chapter 6. The T5 architecture uses the original Transformer architecture. Using the large crawled C4 dataset, the model is pretrained with masked language modeling as well as the SuperGLUE tasks by translating all of them to text-to-text tasks. The largest model with 11 billion parameters yielded state-of-the-art results on several benchmarks.
BART	BART combines the pretraining procedures of BERT and GPT within the encoder-decoder architecture. The input sequences undergo one of several possible transformations, from simple masking to sentence permutation, token deletion, and document rotation. These modified inputs are passed through the encoder, and the decoder has to reconstruct the original texts. This makes the model more flexible as it is possible to use it for NLU as well as NLG tasks, and it achieves state-of-the-art- performance on both.
M2M-100	Conventionally a translation model is built for one language pair and translation direction. Naturally, this does not scale to many languages, and in addition there might be shared knowledge between language pairs that could be leveraged for translation between rare languages. M2M- 100 is the first translation model that can translate between any of 100 languages. This allows for high-quality translations between rare and underrepresented languages. The model uses prefix tokens (similar to the special [CLS] token) to indicate the source and target language.
BigBird	One main limitation of transformer models is the maximum context size, due to the quadratic memory requirements of the attention mechanism. 24 25 BigBird addresses this issue by using a sparse form of attention that scales linearly. This allows for the drastic scaling of contexts from 512 tokens in most BERT models to 4,096 in BigBird. This is especially useful in cases where long dependencies need to be conserved, such as in text summarization.

BERT2BERT

```
EncoderDecoderModel(
    (en coder): BertModel(
        (em bed dings): BertEm bed dings(
            (wo rd_ emb edd ings): Embedd ing (30522, 768, paddin g_i dx=0)
            (po sit ion _em bed dings): Embedd ing (512, 768)
            (to ken _ty pe_ emb edd ings): Embedd ing(2, 768)
            (La yer Norm): LayerN orm ((7 68,), eps=1e-12, elemen twi se_ aff ine =True)
            (dr opout): Dropou t(p =0.1, inplac e=F alse)
       )
        (en coder): BertEn coder(
            (la yer): Module List(
                (0-11): 12 x BertLayer(
```

BERT2BERT (cont)

```
>           (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (decoder): BertLMHeadModel(
    (bert): BertModel(
      (embeddings): BertEmbeddings(
        (word_embeddings): Embedding(30522, 768, padding_idx=0)
        (position_embeddings): Embedding(512, 768)
        (token_type_embeddings): Embedding(2, 768)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (encoder): BertEncoder(
        (layer): ModuleList(
          (0-11): 12 x BertLayer(
            (attention): BertAttention(
```

BERT2BERT (cont)

```
>             (self): BertSelfAttention(
                (query): Linear(in_features=768, out_features=768, bias=True)
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): Linear(in_features=768, out_features=768, bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
              (output): BertSelfOutput(
                (dense): Linear(in_features=768, out_features=768, bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
            (crossattention): BertAttention(
              (self): BertSelfAttention(
                (query): Linear(in_features=768, out_features=768, bias=True)
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): Linear(in_features=768, out_features=768, bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
              (output): BertSelfOutput(
                (dense): Linear(in_features=768, out_features=768, bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
            (intermediate): BertIntermediate(
              (dense): Linear(in_features=768, out_features=3072, bias=True)
              (intermediate_act_fn): GELUActivation()
            )
            (output): BertOutput(
              (dense): Linear(in_features=3072, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
    )
    (cls): BertOnlyMLMHead(
      (predictions): BertLMPredictionHead(
        (transform): BertPredictionHeadTransform(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (transform_act_fn): GELUActivation()
```

BERT2BERT (cont)

```
>          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
        (decoder): Linear(in_features=768, out_features=30522, bias=True)
      )
    )
  )
)
Process finished with exit code 0
```

T5

```
T5Model(
    (sh ared): Embedd ing (32128, 768)
    (en coder): T5Stack(
        (em bed _to kens): Embedd ing (32128, 768)
        (bl ock): Module List(
           (0): T5Block(
                (la yer): Module List(
                   (0): T5Laye rSe lfA tte ntion(
                        (Se lfA tte ntion): T5Atte ntion(
                           (q): Linear (in _fe atu res =768, out_fe atu res =768, bias=F alse)
                           (k): Linear (in _fe atu res =768, out_fe atu res =768, bias=F alse)
                           (v): Linear (in _fe atu res =768, out_fe atu res =768, bias=F alse)
                           (o): Linear (in _fe atu res =768, out_fe atu res =768, bias=F alse)
                            (re lat ive _at ten tio n_b ias): Embedd ing(32, 12)
                       )
                        (la yer _norm): T5Laye rNorm()
                        (dr opout): Dropou t(p =0.1, inplac e=F alse)
                   )
                   (1): T5LayerFF(
                        (De nse Rel uDe nse): T5Dens eAc tDense(
                            (wi): Linear (in _fe atu res =768, out_fe atu res =3072, bias=F alse)
                            (wo): Linear (in _fe atu res =3072, out_fe atu res =768, bias=F alse)
                            (dr opout): Dropou t(p =0.1, inplac e=F alse)
                            (act): ReLU()
                       )
                        (la yer _norm): T5Laye rNorm()
                        (dr opout): Dropou t(p =0.1, inplac e=F alse)
                   )
               )
           )
            (1-11): 11 x T5Block(
                (la yer): Module List(
                   (0): T5Laye rSe lfA tte ntion(
                        (Se lfA tte ntion): T5Atte ntion(
```

T5 (cont)

```
>               (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=768, out_features=3072, bias=False)
              (wo): Linear(in_features=3072, out_features=768, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (decoder): T5Stack(
    (embed_tokens): Embedding(32128, 768)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
              (relative_attention_bias): Embedding(32, 12)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
```

T5 (cont)

```
>               (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=768, out_features=3072, bias=False)
              (wo): Linear(in_features=3072, out_features=768, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-11): 11 x T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=768, out_features=768, bias=False)
              (k): Linear(in_features=768, out_features=768, bias=False)
              (v): Linear(in_features=768, out_features=768, bias=False)
              (o): Linear(in_features=768, out_features=768, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
```

T5 (cont)

```
>             (wi): Linear(in_features=768, out_features=3072, bias=False)
            (wo): Linear(in_features=3072, out_features=768, bias=False)
            (dropout): Dropout(p=0.1, inplace=False)
            (act): ReLU()
          )
          (layer_norm): T5LayerNorm()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (final_layer_norm): T5LayerNorm()
  (dropout): Dropout(p=0.1, inplace=False)
 )
)
```