Package: torchtransformers (via r-universe)

October 31, 2024

Title Transformer Models in Torch

Version 0.0.0.9600

Description Work with transformer models in R using torch.

Encoding UTF-8

LazyData true

URL https://github.com/macmillancontentscience/torchtransformers

BugReports https://github.com/macmillancontentscience/torchtransformers/issues

Depends R (>= 3.5.1)

License Apache License (>= 2)

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.1

Imports cli, dlr (>= 1.0.0), fastmatch, glue, purrr, rlang, stringr (>= 1.4.0), torch (>= 0.8.0), wordpiece (>= 2.1.3), wordpiece.data (>= 2.0.0)

Suggests dplyr, luz, tibble, testthat (>= 3.0.0), knitr, rmarkdown, roxygen2

Config/testthat/edition 3

VignetteBuilder knitr

Repository https://jonthegeek.r-universe.dev

RemoteUrl https://github.com/macmillancontentscience/torchtransformers

RemoteRef HEAD

RemoteSha 513d9b51c1bf61d723055632a78ace17a6163954

Contents

`attention_bert` *BERT-Style Attention*

Description

Takes in an input tensor (e.g. sequence of token embeddings), applies an attention layer and layer-norms the result. Returns both the attention weights and the output embeddings.

Usage

```
attention_bert(embedding_size, n_head, attention_dropout = 0.1)
```

Arguments

`embedding_size`	Integer; the dimension of the embedding vectors.
`n_head`	Integer; the number of attention heads per layer.
`attention_dropout`	Numeric; the dropout probability to apply in attention.

Shape

Inputs:

- input: $(*, sequence_length, embedding_size)$
- optional mask: $(*, sequence_length)$

Output:

- embeddings: $(*, sequence_length, embedding_size)$
- weights: $(*, n_head, sequence_length, sequence_length)$

Examples

```
emb_size <- 4L
seq_len <- 3L
n_head <- 2L
batch_size <- 2L

model <- attention_bert(
  embedding_size = emb_size,
  n_head = n_head
)
# get random values for input
input <- array(
  sample(
    -10:10,
    size = batch_size * seq_len * emb_size,
    replace = TRUE
  ) / 10,
  dim = c(batch_size, seq_len, emb_size)
)
input <- torch::torch_tensor(input)
model(input)
```

`available_berts` *Available BERT Models*

Description

List the BERT models that are defined for this package.

Usage

```
available_berts()
```

Details

Note that some of the models listed here are actually repeats, listed under different names. For example, "bert_L2H128_uncased" and "bert_tiny_uncased" point to the same underlying weights. In general, models with the same values of hyperparameters (accessed by `config_bert`) are identical. However, there is one exception to this: the "bert_base_uncased" and "bert_L12H768_uncased" models have the same hyperparameters and training regime, but are actually distinct models with different actual weights. Any differences between the models are presumably attributable to different random seeds.

Value

A character vector of BERT types.

Examples

```
available_berts()
```

config_bert *BERT Model Parameters*

Description

Several parameters define a BERT model. This function can be used to easily load them.

Usage

```
config_bert(
  bert_type,
 parameter = c("embedding_size", "n_layer", "n_head", "max_tokens", "vocab_size",
    "tokenizer_scheme")
)
```

Arguments

bert_type	Character scalar; the name of a known BERT model.
parameter	Character scalar; the desired parameter.

Value

Integer scalar; the value of that parameter for that model.

Examples

```
config_bert("bert_medium_uncased", "n_head")
```

dataset_bert *BERT Dataset*

Description

Prepare a dataset for BERT-like models.

Usage

```
dataset_bert(x, y = NULL, tokenizer = tokenize_bert, n_tokens = 128L)
```

Arguments

x	A data.frame with one or more character predictor columns.
y	A factor of outcomes, or a data.frame with a single factor column. Can be NULL (default).
tokenizer	A tokenization function (signature compatible with tokenize_bert).
n_tokens	Integer scalar; the number of tokens expected for each example.

Value

An initialized `torch::dataset()`.

Methods

`initialize` Initialize this dataset. This method is called when the dataset is first created.

`.getitem` Fetch an individual predictor (and, if available, the associated outcome). This function is called automatically by `{luz}` during the fitting process.

`.length` Determine the length of the dataset (the number of rows of predictors). Generally superseded by instead calling `length()`.

`dataset_bert_pretrained` *BERT Pretrained Dataset*

Description

Prepare a dataset for pretrained BERT models.

Usage

```
dataset_bert_pretrained(
  x,
  y = NULL,
  bert_type = NULL,
  tokenizer_scheme = NULL,
  n_tokens = NULL
)
```

Arguments

`x`	A data.frame with one or more character predictor columns, or a list, matrix, or character vector that can be coerced to such a data.frame.
`y`	A factor of outcomes, or a data.frame with a single factor column. Can be NULL (default).
`bert_type`	A bert_type from `available_berts()` to use to choose the other properties. If `bert_type` and `n_tokens` are set, they overrule this setting.
`tokenizer_scheme`	A character scalar that indicates vocabulary + tokenizer.
`n_tokens`	An integer scalar indicating the number of tokens in the output.

Value

An initialized `torch::dataset()`. If it is not yet tokenized, the `tokenize()` method must be called before the dataset will be usable.

Fields

`input_data (private)` The input predictors (`x`) standardized to a data.frame of character columns, and outcome (`y`) standardized to a factor or `NULL`.

`tokenizer_metadata (private)` A list indicating the `tokenizer_scheme` and `n_tokens` that have been or will be used to tokenize the predictors (`x`).

`tokenized (private)` A single logical value indicating whether the data has been tokenized.

Methods

`initialize` Initialize this dataset. This method is called when the dataset is first created.

`tokenize` Tokenize this dataset.

`untokenize` Remove any tokenization from this dataset.

`.tokenize_for_model` Tokenize this dataset for a particular model. Generally superseded by instead calling `luz_callback_bert_tokenize()`.

`.getitem` Fetch an individual predictor (and, if available, the associated outcome). Generally superseded by instead calling `.getbatch()` (or by letting the luz modeling process fit automatically).

`.getbatch` Fetch specific predictors (and, if available, the associated outcomes). This function is called automatically by `{luz}` during the fitting process.

`.length` Determine the length of the dataset (the number of rows of predictors). Generally superseded by instead calling `length()`.

`embeddings_bert` *Create BERT Embeddings*

Description

There are three components which are added together to give the input embeddings in a BERT model: the embedding of the tokens themselves, the segment ("token type") embedding, and the position (token index) embedding. This function sets up the embedding layer for all three of these.

Usage

```
embeddings_bert(
  embedding_size,
  max_position_embeddings,
  vocab_size,
  token_type_vocab_size = 2L,
  hidden_dropout = 0.1
)
```

Arguments

`embedding_size`	Integer; the dimension of the embedding vectors.
`max_position_embeddings`	Integer; maximum number of tokens in each input sequence.
`vocab_size`	Integer; number of tokens in vocabulary.
`token_type_vocab_size`	Integer; number of input segments that the model will recognize. (Two for BERT models.)
`hidden_dropout`	Numeric; the dropout probability to apply to dense layers.

Shape

With `sequence_length` <= `max_position_embeddings`:

Inputs:

- token_ids: $(*, sequence_length)$
- token_type_ids: $(*, sequence_length)$

Output:

- $(*, sequence_length, embedding_size)$

Examples

```
emb_size <- 3L
mpe <- 5L
vs <- 7L
n_inputs <- 2L
# get random "ids" for input
t_ids <- matrix(sample(2:vs, size = mpe * n_inputs, replace = TRUE),
  nrow = n_inputs, ncol = mpe
)
ttype_ids <- matrix(rep(1L, mpe * n_inputs), nrow = n_inputs, ncol = mpe)

model <- embeddings_bert(
  embedding_size = emb_size,
  max_position_embeddings = mpe,
  vocab_size = vs
)
model(
  torch::torch_tensor(t_ids),
  torch::torch_tensor(ttype_ids)
)
```

increment_list_index *Convert from Python Standard to torch*

Description

The torch R package uses the R standard of starting counts at 1. Many tokenizers use the Python standard of starting counts at 0. This function converts a list of token ids provided by such a tokenizer to torch-friendly values (by adding 1 to each id).

Usage

```
increment_list_index(list_of_integers)
```

Value

The list of integers, with 1 added to each integer.

Examples

```
increment_list_index(
  list(
    1:5,
    2:6,
    3:7
  )
)
```

luz_callback_bert_tokenize *BERT Tokenization Callback*

Description

Data used in pretrained BERT models must be tokenized in the way the model expects. This `luz_callback` checks that the incoming data is tokenized properly, and triggers tokenization if necessary. This function should be passed to `luz::fit.luz_module_generator()` or `luz::predict.luz_module_fitted()` via the `callbacks` argument, not called directly.

Usage

```
luz_callback_bert_tokenize(
  submodel_name = NULL,
  n_tokens = NULL,
  verbose = TRUE
)
```

Arguments

submodel_name	An optional character scalar identifying a model inside the main torch::nn_module() that was built using model_bert_pretrained(). See vignette("entailment") for an example of a model with a submodel.
n_tokens	An optional integer scalar indicating the number of tokens to which the data should be tokenized. If present it must be equal to or less than the max_tokens allowed by the pretrained model.
verbose	A logical scalar indicating whether the callback should report its progress (default TRUE).

Examples

```
if (rlang::is_installed("luz")) {
  luz_callback_bert_tokenize()
  luz_callback_bert_tokenize(n_tokens = 32L)
}
```

---

model_bert *Construct a BERT Model*

Description

BERT models are the family of transformer models popularized by Google's BERT (Bidirectional Encoder Representations from Transformers). They include any model with the same general structure.

Usage

```
model_bert(
  embedding_size,
  intermediate_size = 4 * embedding_size,
  n_layer,
  n_head,
  hidden_dropout = 0.1,
  attention_dropout = 0.1,
  max_position_embeddings,
  vocab_size,
  token_type_vocab_size = 2L
)
```

Arguments

embedding_size	Integer; the dimension of the embedding vectors.
intermediate_size	Integer; size of dense layers applied after attention mechanism.

`n_layer` Integer; the number of attention layers.

`n_head` Integer; the number of attention heads per layer.

`hidden_dropout` Numeric; the dropout probability to apply to dense layers.

`attention_dropout` Numeric; the dropout probability to apply in attention.

`max_position_embeddings` Integer; maximum number of tokens in each input sequence.

`vocab_size` Integer; number of tokens in vocabulary.

`token_type_vocab_size` Integer; number of input segments that the model will recognize. (Two for BERT models.)

Shape

Inputs:

With `sequence_length` <= `max_position_embeddings`:

- token_ids: $(*, sequence_length)$
- token_type_ids: $(*, sequence_length)$

Output:

- initial_embeddings: $(*, sequence_length, embedding_size)$
- output_embeddings: list of $(*, sequence_length, embedding_size)$ for each transformer layer.
- attention_weights: list of $(*, n_head, sequence_length, sequence_length)$ for each transformer layer.

Examples

```
emb_size <- 128L
mpe <- 512L
n_head <- 4L
n_layer <- 6L
vocab_size <- 30522L
model <- model_bert(
  embedding_size = emb_size,
  n_layer = n_layer,
  n_head = n_head,
  max_position_embeddings = mpe,
  vocab_size = vocab_size
)

n_inputs <- 2
n_token_max <- 128L
# get random "ids" for input
t_ids <- matrix(
  sample(
    2:vocab_size,
    size = n_token_max * n_inputs,
```

```
    replace = TRUE
  ),
  nrow = n_inputs, ncol = n_token_max
)
ttype_ids <- matrix(
  rep(1L, n_token_max * n_inputs),
  nrow = n_inputs, ncol = n_token_max
)
model(
  torch::torch_tensor(t_ids),
  torch::torch_tensor(ttype_ids)
)
```

`model_bert_pretrained` *Construct a Pretrained BERT Model*

Description

Construct a BERT model (using `model_bert()`) and load pretrained weights.

Usage

```
model_bert_pretrained(bert_type = "bert_tiny_uncased", redownload = FALSE)
```

Arguments

`bert_type`	Character; which flavor of BERT to use. See `available_berts()` for known models.
`redownload`	Logical; should the weights be downloaded fresh even if they're cached?

Value

The model with pretrained weights loaded.

Methods

`initialize` Initialize this model. This method is called when the model is first created.

`forward` Use this model. This method is called during training, and also during prediction. `x` is a list of `torch::torch_tensor()` values for `token_ids` and `token_type_ids`.

`.get_tokenizer_metadata` Look up the tokenizer metadata for this model. This method is called automatically when `luz_callback_bert_tokenize()` validates that a dataset is tokenized properly for this model.

`.load_weights` Load the pretrained weights for this model. This method is called automatically during initialization of this model.

position_embedding *Create Position Embeddings*

Description

Position embeddings are how BERT-like language models represent the order of input tokens. Each token gets a position embedding vector which is completely determined by its position index. Because these embeddings don't depend on the actual input, it is implemented by simply initializing a matrix of weights.

Usage

```
position_embedding(embedding_size, max_position_embeddings)
```

Arguments

embedding_size	Integer; the dimension of the embedding vectors.
max_position_embeddings	Integer; maximum number of tokens in each input sequence.

Shape

Inputs:

No input tensors. Optional input parameter to limit number of positions (tokens) considered.

Output:

- $(*, max_position_embeddings, embedding_size)$

Examples

```
emb_size <- 3L
mpe <- 2L
model <- position_embedding(
  embedding_size = emb_size,
  max_position_embeddings = mpe
)
model(seq_len_cap = 1)
model()
```

`proj_add_norm` *Project, Add, and Normalize*

Description

Takes in two tensors, an "input" and a "residual". Applies a linear projector to the input (changing the size to match residual), performs dropout, adds the result to the residual, then applies layer normalization to the sum.

Usage

```
proj_add_norm(input_size, output_size, hidden_dropout = 0.1)
```

Arguments

`input_size`	Integer; the size of input tensor.
`output_size`	Integer; the size of output tensor (must match residual).
`hidden_dropout`	Numeric; dropout probability applied after projection.

Shape

Inputs:

- input: $(*, input_size)$
- residual: $(*, output_size)$

Output:

- $(*, output_size)$

Examples

```
in_size <- 4L
out_size <- 3L
model <- proj_add_norm(input_size = in_size, output_size = out_size)
input <- torch::torch_randn(in_size)
residual <- torch::torch_randn(out_size)
model(input, residual)
```

simplify_bert_token_list *Simplify Token List to Matrix*

Description

BERT-like models expect a matrix of tokens for each example. This function converts a list of equal-length vectors (such as a padded list of tokens) into such a matrix.

Usage

```
simplify_bert_token_list(token_list)
```

Arguments

token_list	A list of vectors. Each vector should have the same length.

Value

A matrix of tokens. Rows are text sequences, and columns are tokens.

Examples

```
simplify_bert_token_list(
  list(
    1:5,
    2:6,
    3:7
  )
)
```

tokenize_bert *Prepare Text for a BERT Model*

Description

To be used in a BERT-style model, text must be tokenized. In addition, text is optionally preceded by a `cls_token`, and segments are ended with a `sep_token`. Finally each example must be padded with a `pad_token`, or truncated if necessary (preserving the wrapper tokens). Many use cases use a matrix of tokens x examples, which can be extracted directly with the `simplify` argument.

Usage

```
tokenize_bert(
  ...,
  n_tokens = 64L,
  increment_index = TRUE,
  pad_token = "[PAD]",
  cls_token = "[CLS]",
  sep_token = "[SEP]",
  tokenizer = wordpiece::wordpiece_tokenize,
  vocab = wordpiece.data::wordpiece_vocab(),
  tokenizer_options = NULL
)
```

Arguments

`...`	One or more character vectors or lists of character vectors. Currently we support a single character vector, two parallel character vectors, or a list of length-1 character vectors. If two vectors are supplied, they are combined pairwise and separated with `sep_token`.
`n_tokens`	Integer scalar; the number of tokens expected for each example.
`increment_index`	Logical; if TRUE, add 1L to all token ids to convert from the Python-inspired 0-indexed standard to the torch 1-indexed standard.
`pad_token`	Character scalar; the token to use for padding. Must be present in the supplied vocabulary.
`cls_token`	Character scalar; the token to use at the start of each example. Must be present in the supplied vocabulary, or `NULL`.
`sep_token`	Character scalar; the token to use at the end of each segment within each example. Must be present in the supplied vocabulary, or `NULL`.
`tokenizer`	The tokenizer function to use to break up the text. It must have a `vocab` argument.
`vocab`	The vocabulary to use to tokenize the text. This vocabulary must include the `pad_token`, `cls_token`, and `sep_token`.
`tokenizer_options`	A named list of additional arguments to pass on to the tokenizer.

Value

An object of class "bert_tokens", which is a list containing a matrix of token ids, a matrix of token type ids, and a matrix of token names.

Examples

```
tokenize_bert(
  c("The first premise.", "The second premise."),
  c("The first hypothesis.", "The second hypothesis.")
)
```

`transformer_encoder_bert` *Transformer Stack*

Description

Build a BERT-style multi-layer attention-based transformer.

Usage

```
transformer_encoder_bert(
  embedding_size,
  intermediate_size = 4 * embedding_size,
  n_layer,
  n_head,
  hidden_dropout = 0.1,
  attention_dropout = 0.1
)
```

Arguments

`embedding_size`	Integer; the dimension of the embedding vectors.
`intermediate_size`	Integer; size of dense layers applied after attention mechanism.
`n_layer`	Integer; the number of attention layers.
`n_head`	Integer; the number of attention heads per layer.
`hidden_dropout`	Numeric; the dropout probability to apply to dense layers.
`attention_dropout`	Numeric; the dropout probability to apply in attention.

Shape

Inputs:

With each input token list of length `sequence_length`:

- input: $(*, sequence_length, embedding_size)$
- optional mask: $(*, sequence_length)$

Output:

- embeddings: list of $(*, sequence_length, embedding_size)$ for each transformer layer.
- weights: list of $(*, n_head, sequence_length, sequence_length)$ for each transformer layer.

Examples

```
emb_size <- 4L
seq_len <- 3L
n_head <- 2L
n_layer <- 5L
batch_size <- 2L

model <- transformer_encoder_bert(
  embedding_size = emb_size,
  n_head = n_head,
  n_layer = n_layer
)
# get random values for input
input <- array(
  sample(
    -10:10,
    size = batch_size * seq_len * emb_size,
    replace = TRUE
  ) / 10,
  dim = c(batch_size, seq_len, emb_size)
)
input <- torch::torch_tensor(input)
model(input)
```

`transformer_encoder_single_bert` *Single Transformer Layer*

Description

Build a single layer of a BERT-style attention-based transformer.

Usage

```
transformer_encoder_single_bert(
  embedding_size,
  intermediate_size = 4 * embedding_size,
  n_head,
  hidden_dropout = 0.1,
  attention_dropout = 0.1
)
```

Arguments

`embedding_size`	Integer; the dimension of the embedding vectors.
`intermediate_size`	Integer; size of dense layers applied after attention mechanism.
`n_head`	Integer; the number of attention heads per layer.

`hidden_dropout` Numeric; the dropout probability to apply to dense layers.

`attention_dropout`
Numeric; the dropout probability to apply in attention.

Shape

Inputs:

- input: $(*, sequence_length, embedding_size)$
- optional mask: $(*, sequence_length)$

Output:

- embeddings: $(*, sequence_length, embedding_size)$
- weights: $(*, n_head, sequence_length, sequence_length)$

Examples

```
emb_size <- 4L
seq_len <- 3L
n_head <- 2L
batch_size <- 2L

model <- transformer_encoder_single_bert(
  embedding_size = emb_size,
  n_head = n_head
)
# get random values for input
input <- array(
  sample(
    -10:10,
    size = batch_size * seq_len * emb_size,
    replace = TRUE
  ) / 10,
  dim = c(batch_size, seq_len, emb_size)
)
input <- torch::torch_tensor(input)
model(input)
```

Index