Dear Harshil,

Sorry for the delay. Attached is my AdapterBERT code. The main class for you to use is the AdapterBertModel (inherits from BertModel and injects instances of BottleneckAdapter in each layer of Bert).

There a couple of parameters that define the "look and feel" of the adapter layers, which you set as properties of the BottleneckAdapterBertConfig instance:

1) layers_to_adapt: a list of integers (0 to 11 for Bert Base) indicating in which layers of Bert to inject the adapters -- by default they are added to all, but you can change this and experiment with different configurations (e.g., add adapters only to the last few layers).

2) adapter_latent_size: the size of the adapter, typically significantly lower than BERT's hidden size (768 for Bert Base). I've player with the adapter_latent_size of 32 and 64, but it really depends on the amount of fine-tuning data you have. The more data you have, the larger the adapter size (but again, the larger the adapter size, the slower the training, so it's a trade-off).

3) adapter_non_linearity: the non-linear function applied after the down-projection and before the up-projection in each adapter (currently supported: tanh, relu, gelu, sigmoid). I tried out tanh and gelu and they worked similarly.

4) adapter_residual: although this can be configured, you should always set it to True (setting it to False leads to bad results, as after adapters are initialized, they have to initially behave as an near-identity function, and this is ensured with the residual connection together with the near-zero initialization of the down- and up-projection weight matrices)

5) add_intermediate_adapter: this indicates whether you also inject an adapter in the BertIntermediate part of each BertLayer (by default, as stated in the Adapter paper, in each BertLayer, we inject two adapters, one in BertIntermediate and one in BertOutput parts; setting this config option to False, allows you to omit the intermediate adapter). I didn't try out yet what happens in terms of performance if you omit the intermediate adapters and only inject output adapters.

In adapter.py there is also the AdapterBertForSequenceClassification class, which inherits from the BertForSequenceClassification which you can directly use for text classification tasks (i.e., for tasks where you just need to feed the CLS representation into a feed-forward classifier). Loading pre-trained BERT variants is the same as with the as with original Bert classes, just call the from_pretrained method.

One important thing: by default, when you instantiate a AdapterBertModel or AdapterBertForSequenceClassification, Transformer's parameters will be frozen (due to a call to freeze_original_params in the constructor of AdapterBertModel). Before you start fine-tuning the AdapterBert model on you final downstream task, you will need to unfreeze the original Transformer's parameters: just call the unfreeze_original_params method of AdapterBertModel before the final fine-tuning.

Let me know if you run into any problems ;).

I also have Adapter variants for Roberta and XLM-R, but we can talk about those later, after you get familiar with the BERT variant.

Cheers,
Goran