Before we start fine-tuning the model as a spam classifier, let's see whether the model already classifies spam messages by prompting it with instructions:

```
text_2 = (
    "Is the following text 'spam'? Answer with 'yes' or 'no':"
    " 'You are a winner you have been specially"
    " selected to receive $1000 cash or a $2000 award.'"
)
token_ids = generate_text_simple(
    model=model,
    idx=text_to_token_ids(text_2, tokenizer),
    max_new_tokens=23,
    context_size=BASE_CONFIG["context_length"]
)
print(token_ids_to_text(token_ids, tokenizer))
```

The model output is

```
Is the following text 'spam'? Answer with 'yes' or 'no': 'You are a winner
you have been specially selected to receive $1000 cash
or a $2000 award.'
The following text 'spam'? Answer with 'yes' or 'no': 'You are a winner
```

Based on the output, it's apparent that the model is struggling to follow instructions. This result is expected, as it has only undergone pretraining and lacks instruction fine-tuning. So, let's prepare the model for classification fine-tuning.

6.5 Adding a classification head

We must modify the pretrained LLM to prepare it for classification fine-tuning. To do so, we replace the original output layer, which maps the hidden representation to a vocabulary of 50,257, with a smaller output layer that maps to two classes: `0` ("not spam") and `1` ("spam"), as shown in figure 6.9. We use the same model as before, except we replace the output layer.

Output layer nodes

We could technically use a single output node since we are dealing with a binary classification task. However, it would require modifying the loss function, as I discuss in "Losses Learned—Optimizing Negative Log-Likelihood and Cross-Entropy in PyTorch" (https://mng.bz/NRZ2). Therefore, we choose a more general approach, where the number of output nodes matches the number of classes. For example, for a three-class problem, such as classifying news articles as "Technology," "Sports," or "Politics," we would use three output nodes, and so forth.

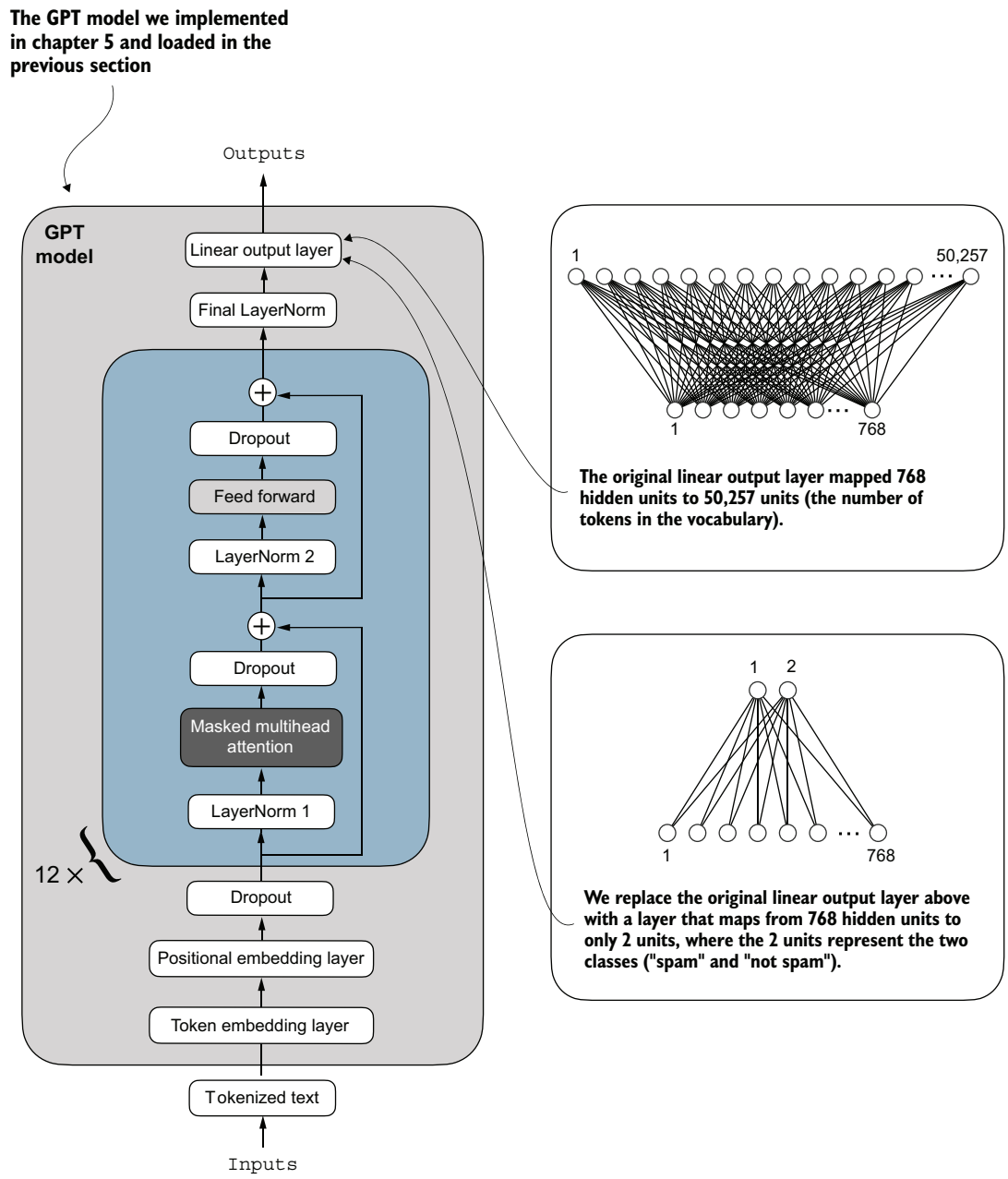

Figure 6.9 Adapting a GPT model for spam classification by altering its architecture. Initially, the model's linear output layer mapped 768 hidden units to a vocabulary of 50,257 tokens. To detect spam, we replace this layer with a new output layer that maps the same 768 hidden units to just two classes, representing "spam" and "not spam."

Before we attempt the modification shown in figure 6.9, let's print the model architecture via `print(model)`:

```
GPTModel(
  (tok_emb): Embedding(50257, 768)
  (pos_emb): Embedding(1024, 768)
  (drop_emb): Dropout(p=0.0, inplace=False)
  (trf_blocks): Sequential(
...
    (11): TransformerBlock(
      (att): MultiHeadAttention(
        (W_query): Linear(in_features=768, out_features=768, bias=True)
        (W_key): Linear(in_features=768, out_features=768, bias=True)
        (W_value): Linear(in_features=768, out_features=768, bias=True)
        (out_proj): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.0, inplace=False)
      )
      (ff): FeedForward(
        (layers): Sequential(
          (0): Linear(in_features=768, out_features=3072, bias=True)
          (1): GELU()
          (2): Linear(in_features=3072, out_features=768, bias=True)
        )
      )
      (norm1): LayerNorm()
      (norm2): LayerNorm()
      (drop_resid): Dropout(p=0.0, inplace=False)
    )
  )
  (final_norm): LayerNorm()
  (out_head): Linear(in_features=768, out_features=50257, bias=False)
)
```

This output neatly lays out the architecture we laid out in chapter 4. As previously discussed, the `GPTModel` consists of embedding layers followed by 12 identical *transformer blocks* (only the last block is shown for brevity), followed by a final `LayerNorm` and the output layer, `out_head`.

Next, we replace the `out_head` with a new output layer (see figure 6.9) that we will fine-tune.

Fine-tuning selected layers vs. all layers

Since we start with a pretrained model, it's not necessary to fine-tune all model layers. In neural network-based language models, the lower layers generally capture basic language structures and semantics applicable across a wide range of tasks and datasets. So, fine-tuning only the last layers (i.e., layers near the output), which are more specific to nuanced linguistic patterns and task-specific features, is often sufficient to adapt the model to new tasks. A nice side effect is that it is computationally more efficient to fine-tune only a small number of layers. Interested readers can find more information, including experiments, on which layers to fine-tune in appendix B.