Next, assume the LLM is given the start context `"every effort moves you"` and generates the following next-token logits:

```
next_token_logits = torch.tensor(
    [4.51, 0.89, -1.90, 6.75, 1.63, -1.62, -1.89, 6.28, 1.79]
)
```

As discussed in chapter 4, inside `generate_text_simple`, we convert the logits into probabilities via the `softmax` function and obtain the token ID corresponding to the generated token via the `argmax` function, which we can then map back into text via the inverse vocabulary:

```
probas = torch.softmax(next_token_logits, dim=0)
next_token_id = torch.argmax(probas).item()
print(inverse_vocab[next_token_id])
```

Since the largest logit value and, correspondingly, the largest softmax probability score are in the fourth position (index position 3 since Python uses 0 indexing), the generated word is `"forward"`.

To implement a probabilistic sampling process, we can now replace `argmax` with the `multinomial` function in PyTorch:

```
torch.manual_seed(123)
next_token_id = torch.multinomial(probas, num_samples=1).item()
print(inverse_vocab[next_token_id])
```

The printed output is `"forward"` just like before. What happened? The `multinomial` function samples the next token proportional to its probability score. In other words, `"forward"` is still the most likely token and will be selected by `multinomial` most of the time but not all the time. To illustrate this, let's implement a function that repeats this sampling 1,000 times:

```
def print_sampled_tokens(probas):
    torch.manual_seed(123)
    sample = [torch.multinomial(probas, num_samples=1).item()
              for i in range(1_000)]
    sampled_ids = torch.bincount(torch.tensor(sample))
    for i, freq in enumerate(sampled_ids):
        print(f"{freq} x {inverse_vocab[i]}")

print_sampled_tokens(probas)
```

The sampling output is

```
73 x closer
0 x every
0 x effort
582 x forward
2 x inches
```

```
0 x moves
0 x pizza
343 x toward
```

As we can see, the word `forward` is sampled most of the time (582 out of 1,000 times), but other tokens such as `closer`, `inches`, and `toward` will also be sampled some of the time. This means that if we replaced the `argmax` function with the `multinomial` function inside the `generate_and_print_sample` function, the LLM would sometimes generate texts such as `every effort moves you toward`, `every effort moves you inches`, and `every effort moves you closer` instead of `every effort moves you forward`.

We can further control the distribution and selection process via a concept called *temperature scaling*. Temperature scaling is just a fancy description for dividing the logits by a number greater than 0:

```
def softmax_with_temperature(logits, temperature):
    scaled_logits = logits / temperature
    return torch.softmax(scaled_logits, dim=0)
```

Temperatures greater than 1 result in more uniformly distributed token probabilities, and temperatures smaller than 1 will result in more confident (sharper or more peaky) distributions. Let's illustrate this by plotting the original probabilities alongside probabilities scaled with different temperature values:

```
temperatures = [1, 0.1, 5]                                    ← Original, lower, and higher confidence
scaled_probas = [softmax_with_temperature(next_token_logits, T)
                for T in temperatures]
x = torch.arange(len(vocab))
bar_width = 0.15
fig, ax = plt.subplots(figsize=(5, 3))
for i, T in enumerate(temperatures):
    rects = ax.bar(x + i * bar_width, scaled_probas[i],
                   bar_width, label=f'Temperature = {T}')
ax.set_ylabel('Probability')
ax.set_xticks(x)
ax.set_xticklabels(vocab.keys(), rotation=90)
ax.legend()
plt.tight_layout()
plt.show()
```

The resulting plot is shown in figure 5.14.

A temperature of 1 divides the logits by 1 before passing them to the `softmax` function to compute the probability scores. In other words, using a temperature of 1 is the same as not using any temperature scaling. In this case, the tokens are selected with a probability equal to the original softmax probability scores via the `multinomial` sampling function in PyTorch. For example, for the temperature setting 1, the token corresponding to "forward" would be selected about 60% of the time, as we can see in figure 5.14.

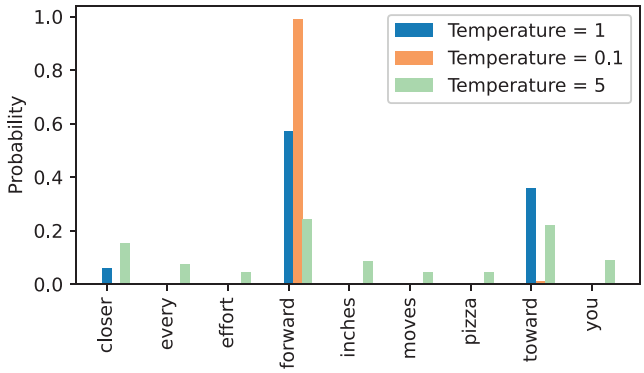

Figure 5.14 A temperature of 1 represents the unscaled probability scores for each token in the vocabulary. Decreasing the temperature to 0.1 sharpens the distribution, so the most likely token (here, "forward") will have an even higher probability score. Likewise, increasing the temperature to 5 makes the distribution more uniform.

Also, as we can see in figure 5.14, applying very small temperatures, such as 0.1, will result in sharper distributions such that the behavior of the `multinomial` function selects the most likely token (here, `"forward"`) almost 100% of the time, approaching the behavior of the `argmax` function. Likewise, a temperature of 5 results in a more uniform distribution where other tokens are selected more often. This can add more variety to the generated texts but also more often results in nonsensical text. For example, using the temperature of 5 results in texts such as `every effort moves you pizza` about 4% of the time.

> **Exercise 5.1**
> Use the `print_sampled_tokens` function to print the sampling frequencies of the softmax probabilities scaled with the temperatures shown in figure 5.14. How often is the word `pizza` sampled in each case? Can you think of a faster and more accurate way to determine how often the word `pizza` is sampled?

5.3.2 Top-k sampling

We've now implemented a probabilistic sampling approach coupled with temperature scaling to increase the diversity of the outputs. We saw that higher temperature values result in more uniformly distributed next-token probabilities, which result in more diverse outputs as it reduces the likelihood of the model repeatedly selecting the most probable token. This method allows for the exploring of less likely but potentially more interesting and creative paths in the generation process. However, one downside of this approach is that it sometimes leads to grammatically incorrect or completely nonsensical outputs such as `every effort moves you pizza`.