

# Verständnis von Neuronalen Netzwerken

Zhiwei:

Natürlich! Bitte teile mir den Code mit, den du erklärt haben möchtest, und ich werde dir gerne dabei helfen, ihn zu verstehen. 😊

```
self.weights = [w-(eta/len(mini_batch))*nw
                 for w, nw in zip(self.weights, nabla_w)]
self.biases = [b-(eta/len(mini_batch))*nb
               for b, nb in zip(self.biases, nabla_b)]
```

(Die Übersetzung des Codes ist nicht notwendig, da es sich um Programmcode handelt, der in der Regel nicht übersetzt wird.)

ChatGPT:

Natürlich! Der von Ihnen bereitgestellte Code scheint Teil eines Algorithmus für maschinelles Lernen zu sein, insbesondere eine Implementierung des stochastischen Gradientenabstiegs (SGD) zur Aktualisierung der Gewichte und Bias-Werte in einem neuronalen Netzwerk während des Trainingsprozesses. Lassen Sie uns dies Schritt für Schritt aufschlüsseln:

1. `self.weights` und `self.biases`: Diese Variablen sind wahrscheinlich Teil einer neuronalen Netzwerkklassse und repräsentieren die Gewichte und Verzerrungen (Biases) des Netzwerks. In einem neuronalen Netzwerk sind Gewichte Parameter, die die Stärke der Verbindungen zwischen Neuronen in verschiedenen Schichten anpassen, während Verzerrungen Parameter sind, die den Aktivierungsschwellenwert der Neuronen steuern.
2. `nabla_w` und `nabla_b`: Diese Variablen speichern wahrscheinlich die Gradienten der Kostenfunktion in Bezug auf die Gewichte und die Bias-Werte. Während des Trainingsprozesses werden Gradienten berechnet, um festzustellen, wie sich die Kostenfunktion in Bezug auf die Gewichte und Bias-Werte ändert. Diese Informationen werden verwendet, um die Parameter zu aktualisieren und die Kosten zu minimieren.
3. `eta`: Dies ist die Lernrate, die die Schrittweite steuert, die in Richtung der Gradienten während der Parameteraktualisierung genommen wird. Eine höhere Lernrate kann zu einer schnelleren Konvergenz während des Trainings führen, aber sie kann auch Instabilität verursachen. Eine kleinere Lernrate könnte die Stabilität verbessern, aber die Trainingszeit verlängern.

4. `mini_batch`: Dies ist eine Teilmenge der Trainingsdaten, die für jede Iteration des stochastischen Gradientenabstiegs verwendet wird. Mini-Batch-Gradientenabstieg ist eine Variante von SGD, bei der der gesamte Trainingsdatensatz in kleinere Batches unterteilt wird und die Parameteraktualisierungen mit jedem Mini-Batch durchgeführt werden.

Nun schauen wir uns den Code selbst an:

```
self.weights = [w-(eta/len(mini_batch))*nw
                 for w, nw in zip(self.weights, nabla_w)]
```

(Der Code bleibt unverändert, da es sich um eine Programmiersprache handelt und keine Übersetzung erforderlich ist.)

Diese Zeile aktualisiert die Gewichte des neuronalen Netzwerks. Sie verwendet eine List Comprehension, um durch jedes Gewicht `w` und den entsprechenden Gradienten `nw` zu iterieren. Für jedes Gewicht wird ein Teil des Gradienten (skaliert durch die Lernrate `eta` und die Größe des Mini-Batches) vom ursprünglichen Gewicht subtrahiert. Auf diese Weise werden die Gewichte aktualisiert, um die Kostenfunktion zu minimieren.

```
self.biases = [b-(eta/len(mini_batch))*nb
                for b, nb in zip(self.biases, nabla_b)]
```

*Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird.*

Diese Zeile aktualisiert die Bias-Werte des neuronalen Netzwerks auf ähnliche Weise. Es wird eine List Comprehension verwendet, um durch jeden Bias-Wert `b` und den entsprechenden Gradienten `nb` zu iterieren. Für jeden Bias-Wert wird ein Teil des Gradienten (skaliert mit der Lernrate `eta` und der Größe des Mini-Batches) vom ursprünglichen Bias-Wert subtrahiert.

Zusammenfassend führen diese Codezeilen die Aktualisierungen der Gewichte und Bias-Werte für das neuronale Netzwerk mittels stochastischem Gradientenabstieg durch. Dieser Prozess wird für jedes Mini-Batch während der Trainingsphase wiederholt, um die Parameter des Netzwerks zu optimieren und seine Leistung bei der gegebenen Aufgabe zu verbessern.

---

Zhiwei: Was ist ein SpaCy-Sprachmodell?

ChatGPT: SpaCy ist eine Open-Source-Bibliothek für die Verarbeitung natürlicher Sprache (Natural Language Processing, NLP), entwickelt von Explosion AI. Sie ist darauf ausgelegt, effizient, schnell und produktionsreif zu sein, was sie zu einer beliebten Wahl für verschiedene NLP-Aufgaben macht. SpaCy bietet vortrainierte Sprachmodelle, die Text verarbeiten und verstehen können, wodurch Entwickler verschiedene NLP-Aufgaben wie Part-of-Speech-Tagging, Named Entity Recognition, Satzparsing und mehr durchführen können.

Die wichtigsten Funktionen des SpaCy-Sprachmodells umfassen:

1. Tokenisierung: SpaCy kann einen Text in einzelne Wörter oder Tokens aufteilen und dabei verschiedene linguistische Regeln und Komplexitäten berücksichtigen.
2. Part-of-Speech-Tagging: Die Bibliothek kann jedem Token Part-of-Speech-Tags zuweisen, die angeben, ob es sich um ein Substantiv, Verb, Adjektiv usw. handelt.
3. Named Entity Recognition (NER): SpaCy kann Entitäten wie Namen von Personen, Organisationen, Orten, Datumsangaben usw. identifizieren und kategorisieren.
4. Abhängigkeitsparsing: Es kann die grammatikalische Struktur eines Satzes analysieren und die Beziehungen zwischen den Wörtern bestimmen.
5. Lemmatisierung: SpaCy kann Wörter in ihre Grund- oder Stammform (Lemmata) umwandeln.
6. Wortvektoren: Die Bibliothek bietet Wortvektoren, auch bekannt als Wort-Embeddings, bei denen es sich um dichte numerische Darstellungen von Wörtern handelt, die semantische Bedeutungen erfassen.

SpaCy unterstützt mehrere Sprachen und bietet vortrainierte Modelle für verschiedene Sprachen, was es in einer Vielzahl von NLP-Anwendungen weit verbreitet macht, darunter Informationsgewinnung, Sentiment-Analyse, Chatbots, Sprachübersetzung und mehr. Darüber hinaus ermöglicht es Entwicklern, bei Bedarf benutzerdefinierte Modelle auf ihren spezifischen Datensätzen zu trainieren.

---

Zhiwei: Was ist ein SpaCy-Sprachmodell?

ChatGPT: SpaCy ist eine Open-Source-Bibliothek für die Verarbeitung natürlicher Sprache (Natural Language Processing, NLP), entwickelt von Explosion AI. Sie ist darauf ausgelegt, effizient, schnell und einfach zu bedienen zu sein, was sie zu einer beliebten Wahl für verschiedene

NLP-Aufgaben macht. SpaCy bietet vortrainierte Sprachmodelle für mehrere Sprachen an, die es Benutzern ermöglichen, Aufgaben wie Tokenisierung, Part-of-Speech-Tagging, Erkennung von benannten Entitäten (Named Entity Recognition), Abhängigkeitsparsing und mehr durchzuführen.

Das Herzstück von Spacys Geschwindigkeit und Effizienz liegt in seinen optimierten Cython-Implementierungen und Datenstrukturen, die es besonders geeignet für Produktionsumgebungen und reale Anwendungen machen. Die Bibliothek ist in Python geschrieben und wird mit vortrainierten Modellen für verschiedene Sprachen geliefert, darunter Englisch, Deutsch, Französisch, Spanisch und mehr.

SpaCy-Sprachmodelle verwenden Deep-Learning-Techniken, um die zugrunde liegenden Muster in natürlichen Sprachdaten zu erlernen. Diese Modelle werden auf großen Textkorpora trainiert, und die erlernten Repräsentationen können dann für verschiedene NLP-Aufgaben verwendet werden. Die vortrainierten Modelle können weiter auf spezifische Daten feinabgestimmt werden, um eine bessere Leistung bei domänenspezifischen Aufgaben zu erzielen.

Die Architektur und Benutzerfreundlichkeit haben SpaCy zu einer beliebten Wahl unter Entwicklern und Forschern in der NLP-Community gemacht. Es wird häufig für Aufgaben wie Informationsextraktion, Textklassifizierung, Named Entity Recognition, Part-of-Speech-Tagging und syntaktische Analyse verwendet. Darüber hinaus ermöglicht das modulare Design von SpaCy den Benutzern, es einfach in ihre bestehenden NLP-Workflows und -Pipelines zu integrieren.

---

Zhiwei: Könnten Sie das erklären?

```
class Llama:
    @staticmethod
    def build(
        ckpt_dir: str,
        tokenizer_path: str,
        max_seq_len: int,
        max_batch_size: int,
        model_parallel_size: Optional[int] = None,
    ) -> "Llama":
        if not torch.distributed.is_initialized():
            torch.distributed.init_process_group("nccl")
```

```

if not model_parallel_is_initialized():
    if model_parallel_size is None:
        model_parallel_size = int(os.environ.get("WORLD_SIZE", 1))
        initialize_model_parallel(model_parallel_size)

```

*Hinweis: Der Code wurde nicht übersetzt, da es sich um Programmcode handelt, der in der Regel in der Originalsprache belassen wird, um die Funktionalität und Lesbarkeit zu gewährleisten.*

```

local_rank = int(os.environ.get("LOCAL_RANK", 0))
torch.cuda.set_device(local_rank)

# Der Seed muss in allen Prozessen gleich sein
torch.manual_seed(1)

if local_rank > 0:
    sys.stdout = open(os.devnull, "w")

    start_time = time.time()
    checkpoints = sorted(Path(ckpt_dir).glob("*.pth"))
    assert len(checkpoints) > 0, f"keine Checkpoint-Dateien in {ckpt_dir} gefunden"
    assert model_parallel_size == len(
        checkpoints
    ), f"Laden eines Checkpoints für MP={len(checkpoints)}, aber die Weltgröße ist {model_parallel_size}"
    ckpt_path = checkpoints[get_model_parallel_rank()]
    checkpoint = torch.load(ckpt_path, map_location="cpu")
    with open(Path(ckpt_dir) / "params.json", "r") as f:
        params = json.loads(f.read())

```

Führen Sie den Befehl aus und rufen Sie dann das Fehlerprotokoll ab.

```

% torchrun --nproc_per_node 1 example_text_completion.py \
    --ckpt_dir llama-2-7b/ \
    --tokenizer_path tokenizer.model \
    --max_seq_len 128 --max_batch_size 4

```

HINWEIS: Umleitungen werden derzeit unter Windows oder MacOS nicht unterstützt.

Traceback (letzter Aufruf zuletzt):

```

Datei "/Users/lzwjava/projects/llama/example_text_completion.py", Zeile 55, in <module>
    fire.Fire(main)
Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/fire/core.py", Zeile 141, in Fire
    component_trace = _Fire(component, args, parsed_flag_args, context, name)
Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/fire/core.py", Zeile 475, in _Fire
    component, remaining_args = _CallAndUpdateTrace(
Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/fire/core.py", Zeile 691, in _CallAndUpdateTrace
    component = fn(*varargs, **kwargs)
Datei "/Users/lzwjava/projects/llama/example_text_completion.py", Zeile 18, in main
    generator = Llama.build(
Datei "/Users/lzwjava/projects/llama/llama/generation.py", Zeile 62, in build
    torch.distributed.init_process_group("nccl")
Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/distributed_c10d.py", Zeile 100, in _new_process_group_helper
    default_pg = _new_process_group_helper(
Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/distributed_c10d.py", Zeile 100, in _new_process_group_helper
    raise RuntimeError("Distributed package doesn't have NCCL " "built in")
RuntimeError: Das verteilte Paket hat NCCL nicht integriert
ERROR:torch.distributed.elastic.multiprocessing.api:fehlgeschlagen (Exitcode: 1) local_rank: 0 (pid: 23273)

Traceback (letzter Aufruf zuletzt): Datei "/Users/lzwjava/anaconda3/bin/torchrun", Zeile
33, in sys.exit(load_entry_point('torch==2.0.1', 'console_scripts', 'torchrun'))() Datei
"/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/elastic/multiprocessing/errors/init.py",
Zeile 346, in wrapper return f(*args, **kwargs) Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-
packages/torch/distributed/run.py", Zeile 794, in main run(args) Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-
packages/torch/distributed/run.py", Zeile 785, in run elastic_launch( Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-
packages/torch/distributed/launcher/api.py", Zeile 134, in  return launch_agent(self._config,
self._entrypoint, list(args)) Datei "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/launcher/api.py",
Zeile 250, in launch_agent raise ChildFailedError( torch.distributed.elastic.multiprocessing.errors.ChildFailedError:
=====
example_text_completion.py  FEHLGESCHLAGEN  =====
Fehler:  ----- Ursache (erster beobachteter Fehler):
[0]: Zeit : 2023-08-03_15:35:37 Host : 1.0.0.127.in-addr.arpa Rang : 0 (local_rank: 0)
Exitcode : 1 (pid: 23273) Fehlerdatei: Traceback : Um den Traceback zu aktivieren, siehe:
https://pytorch.org/docs/stable/elastic/errors.html =====

```

Lerne die Fehler-Stack-Trace kennen.

Frage ChatGPT, was Backend.NCCL in PyTorch bedeutet?

---

Zhiwei:

```
```code:```

```python
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM

model_path = '.'
# model_path = 'openlm-research/open_llama_7b'

tokenizer = LlamaTokenizer.from_pretrained(model_path)
model = LlamaForCausalLM.from_pretrained(
    model_path, torch_dtype=torch.float16, device_map='auto',
)

prompt = 'Q: What is the largest animal?\nA:'
input_ids = tokenizer(prompt, return_tensors="pt").input_ids
generation_output = model.generate(
    input_ids=input_ids, max_new_tokens=32
)
print(tokenizer.decode(generation_output[0]))
```

Fehler:

```
/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py:1445: UserWarning:
  warnings.warn(
Traceback (letzter Aufruf zuletzt):
  Datei "/home/lzw/Projects/open_llama_3b/run.py", Zeile 17, in <module>
    generation_output = model.generate(
                        ~~~~~~
  Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/utils/_contextlib.py", Zeile
    return func(*args, **kwargs)
```

```

~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py",
    return self.greedy_search(
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py",
    outputs = self(
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", Zeile
    return forward_call(*args, **kwargs)
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/models/llama/modeling
    outputs = self.model(
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", Zeile
    return forward_call(*args, **kwargs)
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/models/llama/modeling
    inputs_embeds = self.embed_tokens(input_ids)
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", Zeile
    return forward_call(*args, **kwargs)
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/sparse.py", Zeile
    return F.embedding(
~~~~~
Datei "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/functional.py", Zeile 221
    return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)
~~~~~

```

RuntimeError: Es wurde erwartet, dass alle Tensoren auf demselben Gerät liegen, aber es wurden mindestens

wie behebt man das?

ChatGPT:

Die Fehlermeldung, die Sie erhalten, deutet darauf hin, dass es eine Diskrepanz in den Gerätetypen zwischen Ihrem Eingabe-Tensor (`input_ids`) und Ihrem Modell (`LlamaForCausalLM`) gibt. Der Eingabe-Tensor befindet sich auf der CPU (`cpu`), während das Modell auf der GPU (`cuda`) liegt. Um dies zu beheben, müssen Sie sicherstellen, dass der Eingabe-Tensor auf demselben



Gerät wie das Modell liegt, bevor Sie Text mit dem Modell generieren.

Hier ist der angepasste Code, um das Problem zu beheben:

```
import torch
from transformers import LlamaTokenizer, LlamaForCausalLM

model_path = '.'
# model_path = 'openlm-research/open_llama_7b'

tokenizer = LlamaTokenizer.from_pretrained(model_path)
model = LlamaForCausalLM.from_pretrained(
    model_path, torch_dtype=torch.float16, device_map='auto',
)

prompt = 'Q: What is the largest animal?\nA:'
input_ids = tokenizer(prompt, return_tensors="pt").input_ids
```

## Verschiebe den input\_ids-Tensor auf dasselbe Gerät wie das Modell

```
input_ids = input_ids.to(model.device)

generation_output = model.generate(
    input_ids=input_ids, max_new_tokens=32
)

print(tokenizer.decode(generation_output[0]))
```

In diesem Code habe ich die Zeile `input_ids = input_ids.to(model.device)` hinzugefügt, nachdem der `input_ids`-Tensor vom Tokenizer erhalten wurde. Diese Zeile verschiebt den `input_ids`-Tensor auf dasselbe Gerät wie das Modell (entweder CPU oder GPU), bevor der Text generiert wird. Dies sollte den Fehler “Expected all tensors to be on the same device” beheben.

Denken Sie daran, dass es, wenn Sie diesen Code auf einer GPU-fähigen Maschine ausführen und Ihr Modell auf der GPU läuft, im Allgemeinen effizienter ist, sowohl das Modell als auch die Eingabedaten auf der GPU zu belassen, um die Vorteile der parallelen Verarbeitungsfähigkeiten zu nutzen.

---

Zhiwei: Könntest du den folgenden Code erklären?

```
def generate(
    self,
    prompt_tokens: List[List[int]],
    max_gen_len: int,
    temperature: float = 0.6,
    top_p: float = 0.9,
    logprobs: bool = False,
    echo: bool = False,
) -> Tuple[List[List[int]], Optional[List[List[float]]]]:
    params = self.model.params
    bsz = len(prompt_tokens)
    assert bsz <= params.max_batch_size, (bsz, params.max_batch_size)

    min_prompt_len = min(len(t) for t in prompt_tokens)
    max_prompt_len = max(len(t) for t in prompt_tokens)
    assert max_prompt_len <= params.max_seq_len
    total_len = min(params.max_seq_len, max_gen_len + max_prompt_len)

    pad_id = self.tokenizer.pad_id
    tokens = torch.full((bsz, total_len), pad_id, dtype=torch.long, device="cuda")
    for k, t in enumerate(prompt_tokens):
        tokens[k, : len(t)] = torch.tensor(t, dtype=torch.long, device="cuda")
    if logprobs:
        token_logprobs = torch.zeros_like(tokens, dtype=torch.float)

    prev_pos = 0
    eos_reached = torch.tensor([False] * bsz, device="cuda")
    input_text_mask = tokens != pad_id
    for cur_pos in range(min_prompt_len, total_len):
        logits = self.model.forward(tokens[:, prev_pos:cur_pos], prev_pos)
        if logprobs:
            token_logprobs[:, prev_pos + 1 : cur_pos + 1] = -F.cross_entropy(
                input=logits.transpose(1, 2),
```

```

        target=tokens[:, prev_pos + 1 : cur_pos + 1],
        reduction="none",
        ignore_index=pad_id,
    )
if temperature > 0:
    probs = torch.softmax(logits[:, -1] / temperature, dim=-1)
    next_token = sample_top_p(probs, top_p)
else:
    next_token = torch.argmax(logits[:, -1], dim=-1)

    next_token = next_token.reshape(-1)
    # Token nur ersetzen, wenn der Prompt bereits generiert wurde
    next_token = torch.where(
        input_text_mask[:, cur_pos], tokens[:, cur_pos], next_token
    )
    tokens[:, cur_pos] = next_token
    eos_reached |= (~input_text_mask[:, cur_pos]) & (
        next_token == self.tokenizer.eos_id
    )
    prev_pos = cur_pos
    if all(eos_reached):
        break

if logprobs:
    token_logprobs = token_logprobs.tolist()
out_tokens, out_logprobs = [], []
for i, toks in enumerate(tokens.tolist()):
    # auf maximale Generierungslänge kürzen
    start = 0 if echo else len(prompt_tokens[i])
    toks = toks[start : len(prompt_tokens[i]) + max_gen_len]
    probs = None
    if logprobs:
        probs = token_logprobs[i][start : len(prompt_tokens[i]) + max_gen_len]
    # bis zum EOS-Token kürzen, falls vorhanden
    if self.tokenizer.eos_id in toks:
        eos_idx = toks.index(self.tokenizer.eos_id)
        toks = toks[:eos_idx]

```

```

        probs = probs[:eos_idx] if logprobs else None
        out_tokens.append(toks)
        out_logprobs.append(probs)
    return (out_tokens, out_logprobs if logprobs else None)

```

ChatGPT: Sicherlich! Der bereitgestellte Code ist eine Methode namens `generate`, die einen Prompt (Eingabetext) entgegennimmt und Text mithilfe eines Sprachmodells generiert. Diese Methode ist dafür ausgelegt, mit einem bestimmten Modell und Tokenizer zu arbeiten und führt die Textgenerierung entweder durch Greedy Sampling oder Top-p Sampling durch. Hier ist eine Aufschlüsselung, wie die Methode funktioniert:

### 1. Methodenparameter:

- `prompt_tokens`: Eine Liste von tokenisierten Eingabeaufforderungen, wobei jede Eingabeaufforderung als Liste von Ganzzahlen (Token-IDs) dargestellt wird.
- `max_gen_len`: Die maximale Länge der generierten Ausgabe (in Tokens).
- `temperature`: Ein Parameter, der die Zufälligkeit der Generierung steuert. Höhere Werte (z. B. 1,0) machen die Ausgabe zufälliger, während niedrigere Werte (z. B. 0,6) sie deterministischer machen.
- `top_p`: Ein Parameter, der die Vielfalt des generierten Textes steuert. Höhere Werte (z. B. 0,9) ermöglichen eine diversere Stichprobenziehung.
- `logprobs`: Wenn `True`, gibt die Methode auch die Log-Wahrscheinlichkeiten der generierten Tokens zurück.
- `echo`: Wenn `True`, enthält die generierte Ausgabe die Eingabeaufforderung selbst.

2. **Initialisierung:** Die Methode initialisiert Variablen wie `params`, `bsz` (Batch-Größe), `min_prompt_len` und `max_prompt_len`. Außerdem wird `total_len` berechnet, was die Gesamtlänge der generierten Ausgabe einschließlich des Prompts darstellt.

3. **Token-Generierung:** Die Methode generiert Token für den Ausgabebetext. Sie verwendet eine Schleife, um über Positionen in der generierten Sequenz zu iterieren, beginnend am Ende des Prompts und sich vorwärts bewegend.

- Die Logits für jede Position werden erhalten, indem die partielle Sequenz an die Vorwärtsfunktion des Modells übergeben wird.
- Wenn `temperature` größer als 0 ist, werden die Logits mit dem gegebenen Temperaturwert in Wahrscheinlichkeiten umgewandelt, indem die Softmax-Funktion angewendet wird. Wenn `temperature` 0 ist, wird das Token mit dem höchsten Logit-Wert mithilfe von `torch.argmax` ausgewählt.

- Das nächste Token wird basierend auf den generierten Wahrscheinlichkeiten oder dem Argmax bestimmt und in die entsprechende Position des `tokens`-Tensors platziert.
- EOS-Token (End-of-Sequence) werden erkannt, und wenn alle Sequenzen ein EOS-Token erreicht haben, wird der Generierungsprozess gestoppt.
- Die generierten Token werden im `tokens`-Tensor gespeichert.

#### 4. **Verarbeitung des generierten Outputs:**

- Die generierten Tokens werden verarbeitet, um die Ausgabe auf die gewünschte maximale Länge (`max_gen_len`) zu kürzen.
- Wenn `echo` auf `True` gesetzt ist, wird der Prompt in die generierte Ausgabe eingeschlossen.
- Wenn das `eos`-Token (End-of-Sequence-Token) erkannt wird, wird der generierte Text an dieser Stelle abgeschnitten.

5. **Rückgabe:** Die Methode gibt ein Tupel zurück, das die generierten Tokens und, falls `logprobs` auf `True` gesetzt ist, die Log-Wahrscheinlichkeiten der generierten Tokens enthält.

Diese Methode generiert Text, indem sie iterativ Token basierend auf dem Kontext, der durch den Prompt und die zuvor generierten Token bereitgestellt wird, vorhersagt. Sie verwendet die angegebenen Generierungsparameter, um die Zufälligkeit und Vielfalt des generierten Textes zu steuern. Die Methode ist dafür ausgelegt, mit einer spezifischen Modellarchitektur und einem Tokenizer verwendet zu werden, da sie sich auf deren Eigenschaften und Methoden zur Textgenerierung verlässt.

---