

# 理解神经网络

Zhiwei:

你能解释一下这段代码吗？

```
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)]
```

ChatGPT:

当然！你提供的代码似乎是机器学习算法的一部分，具体来说，是在神经网络训练过程中使用随机梯度下降（SGD）来更新权重和偏差。我们逐步分析一下：

1. `self.weights` 和 `self.biases`：这些变量可能是神经网络类的一部分，表示网络的权重和偏差。在神经网络中，权重是调整不同层之间神经元连接强度的参数，而偏差是控制神经元激活阈值的参数。
2. `nabla_w` 和 `nabla_b`：这些变量可能存储了关于权重和偏差的损失函数的梯度。在训练过程中，通过计算梯度来确定损失函数如何随着权重和偏差变化，而这些信息用于更新参数，以最小化损失。
3. `eta`：这是学习率，控制在更新参数时沿着梯度方向采取的步长。较高的学习率可以加快训练过程中的收敛，但也可能导致不稳定。较小的学习率可以提高稳定性，但可能增加训练时间。
4. `mini_batch`：这是每次随机梯度下降迭代使用的训练数据子集。小批量梯度下降是 SGD 的一种变体，它将整个训练数据集划分为更小的批次，并使用每个小批次进行参数更新。

现在，看看代码本身：

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

这一行更新神经网络的权重。它使用列表推导式遍历每个权重 `w` 及其对应的梯度 `nw`。对于每个权重，它从原始权重中减去一部分梯度（乘以学习率 `eta` 和小批量的大小）。这就是如何更新权重以最小化损失函数。

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

这一行以类似的方式更新神经网络的偏差。它使用列表推导式遍历每个偏差 `b` 及其对应的梯度 `nb`。对于每个偏差，它从原始偏差中减去一部分梯度（乘以学习率 `eta` 和小批量的大小）。

总之，这些代码行使用随机梯度下降进行神经网络的权重和偏差更新。这个过程会在每个小批量的训练阶段重复进行，以优化网络的参数并提高其在给定任务上的表现。

---

Zhiwei: 什么是 SpaCy 语言模型？

ChatGPT: SpaCy 是一个开源的自然语言处理（NLP）库，由 Explosion AI 开发。它被设计为高效、快速，并且可以在生产环境中使用，因此是各种 NLP 任务中的流行选择。SpaCy 提供了经过预训练的语言模型，可以处理和理解文本，使得开发者能够执行各种 NLP 任务，如词性标注、命名实体识别、句子解析等。

SpaCy 语言模型的主要特点包括：

1. **分词**：SpaCy 可以将文本拆分为单个单词或标记，并考虑到各种语言规则和复杂性。
2. **词性标注**：该库可以为每个标记分配词性标签，指明它是名词、动词、形容词等。
3. **命名实体识别（NER）**：SpaCy 可以识别并分类实体，如人名、组织名、地点、日期等。
4. **依赖句法分析**：它可以分析句子的语法结构，并确定单词之间的关系。
5. **词形还原**：SpaCy 可以将单词转换为其基础形式或词根。
6. **词向量**：该库提供词向量，也称为词嵌入，是单词的密集数值表示，捕捉语义意义。

SpaCy 支持多种语言，并提供不同语言的预训练模型，使其广泛应用于各种 NLP 任务，包括信息提取、情感分析、聊天机器人、语言翻译等。此外，SpaCy 还允许开发者根据需要在特定数据集上训练自定义模型。

---

Zhiwei: 什么是 SpaCy 语言模型？

ChatGPT: SpaCy 是一个开源的自然语言处理（NLP）库，由 Explosion AI 开发。它被设计为高效、快速且易于使用，因此成为执行各种 NLP 任务的流行选择。SpaCy 提供了适用于多种语言的预训练语言模型，允许用户执行分词、词性标注、命名实体识别、依赖解析等任务。

SpaCy 的速度和效率核心在于其优化的 Cython 实现和数据结构，使其特别适用于生产环境和实际应用。该库用 Python 编写，且提供了针对多种语言的预训练模型，包括英语、德语、法语、西班牙语等。

SpaCy 语言模型使用深度学习技术来学习自然语言数据中的潜在模式。这些模型是基于大量文本语料库训练的，学习到的表示可以用于各种 NLP 任务。预训练模型可以进一步在特定数据上微调，以在特定领域任务中获得更好的性能。

其架构和易用性使得 SpaCy 在 NLP 社区中的开发者和研究人员中广受欢迎。它通常用于信息提

取、文本分类、命名实体识别、词性标注和句法解析等任务。此外，SpaCy 的模块化设计使用户能够轻松地将其集成到现有的 NLP 工作流和管道中。

---

Zhiwei: 你能解释一下以下代码吗？

```
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)

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

        # seed must be the same in all processes
        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"no checkpoint files found in {ckpt_dir}"
        assert model_parallel_size == len(
            checkpoints
        ), f"Loading a checkpoint for MP={len(checkpoints)} but world size is {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())

```

运行命令后，得到错误日志。

```

% 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

```

NOTE: Redirects are currently not supported in Windows or MacOS.

Traceback (most recent call last):

```

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

```

Traceback (most recent call last):

```

File "/Users/lzwjava/anaconda3/bin/torchrun", line 33, in <module>
    sys.exit(load_entry_point('torch==2.0.1', 'console_scripts', 'torchrun')())
File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/elastic/multiprocessing/api.py", line 129, in _run
    return f(*args, **kwargs)

```

```

File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/run.py", line 794, in m
    run(args)
File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/run.py", line 785, in r
    elastic_launch(
File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/launcher/api.py", line
    return launch_agent(self._config, self._entrypoint, list(args))
File "/Users/lzwjava/anaconda3/lib/python3.10/site-packages/torch/distributed/launcher/api.py", line
    raise ChildFailedError(
torch.distributed.elastic.multiprocessing.errors.ChildFailedError:
=====
example_text_completion.py FAILED
-----
Failures:
  <NO_OTHER_FAILURES>
-----
Root Cause (first observed failure):
[0]:
  time      : 2023-08-03_15:35:37
  host      : 1.0.0.127.in-addr.arpa
  rank      : 0 (local_rank: 0)
  exitcode  : 1 (pid: 23273)
  error_file: <N/A>
  traceback : To enable traceback see: https://pytorch.org/docs/stable/elastic/errors.html
=====

```

学习错误堆栈跟踪。

向 ChatGPT 询问 Backend.NCCL 在 Pytorch 中意味着什么？

---

Zhiwei:

代码:

```

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]))

```

错误:

```

/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py:1445: UserWarning:
  warnings.warn(
Traceback (most recent call last):
  File "/home/lzw/Projects/open_llama_3b/run.py", line 17, in <module>
    generation_output = model.generate(
                        ~~~~~~
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/utils/_contextlib.py", line 116, in __call__
    return func(*args, **kwargs)
           ~~~~~~
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py", line 1311, in generate
    return self.greedy_search(
           ~~~~~~
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/generation/utils.py", line 1147, in greedy_search
    outputs = self(
              ~~~~~
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1553, in _wrapped_call
    return forward_call(*args, **kwargs)
           ~~~~~~
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/models/llama/modeling_llama.py", line 1147, in forward
    outputs = self.model(
              ~~~~~~
  File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1553, in _wrapped_call
    return forward_call(*args, **kwargs)
           ~~~~~~

```

```

    return forward_call(*args, **kwargs)
    ~~~~~
File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/transformers/models/llama/modeling_
    inputs_embeds = self.embed_tokens(input_ids)
    ~~~~~
File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/module.py", line 1
    return forward_call(*args, **kwargs)
    ~~~~~
File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/modules/sparse.py", line 1
    return F.embedding(
    ~~~~~
File "/home/lzw/anaconda3/envs/llama/lib/python3.11/site-packages/torch/nn/functional.py", line 2210,
    return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)
    ~~~~~
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and

```

如何修复？

ChatGPT:

你遇到的错误信息表明输入张量 (`input_ids`) 和模型 (`LlamaForCausalLM`) 之间存在设备类型不匹配。输入张量在 CPU (`cpu`) 上，而模型在 GPU (`cuda`) 上。为了解决这个问题，你需要确保输入张量与模型在同一个设备上，然后再进行生成。

以下是修复该问题的修改代码：

```

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

```

# 将 `input_ids` 张量移动到与模型相同的设备

```
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]))
```

在这段代码中,我在从分词器获得 `input_ids` 张量后,添加了 `input_ids = input_ids.to(model.device)` 这一行。这行代码将 `input_ids` 张量移动到与模型相同的设备 (CPU 或 GPU) 上,然后再进行文本生成。这应该能解决“期望所有张量在同一设备”的错误。

请记住,如果你在启用 GPU 的机器上运行此代码,且模型也在 GPU 上,通常更高效的做法是将模型和输入数据都保存在 GPU 上,以利用并行处理能力。

---

Zhiwei: 你能解释以下代码吗?

```
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)
    # only replace token if prompt has already been generated
    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()):
        # cut to max gen len
        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]
        # cut to eos tok if any
        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: 当然！这段代码是一个名为 `generate` 的方法，接受一个提示（输入文本）并使用语言模型生成文本。这个方法设计为与特定的模型和分词器配合使用，通过贪婪采样或顶端采样生成文本。以下是方法的详细说明：

#### 1. 方法参数：

- `prompt_tokens`：一个由多个列表组成的列表，每个列表表示一个已标记化的提示（由整数表示的标记 ID）。
- `max_gen_len`：生成输出的最大长度（以标记为单位）。
- `temperature`：一个控制生成随机性的参数。较高的值（例如 1.0）使输出更随机，而较低的值（例如 0.6）使输出更确定。
- `top_p`：一个控制生成文本多样性的参数。较高的值（例如 0.9）允许更丰富的采样。
- `logprobs`：如果为 `True`，该方法还返回生成标记的对数概率。
- `echo`：如果为 `True`，生成的输出将包括提示本身。

2. 初始化：方法初始化了如 `params`、`bsz`（批量大小）、`min_prompt_len` 和 `max_prompt_len` 等变量，并计算了 `total_len`，即包括提示在内的生成输出的总长度。

3. **生成标记：**方法通过循环遍历生成序列中的位置，逐步生成输出文本。

- 对于每个位置，通过将部分序列传递给模型的前向函数来获得 logits。
- 如果 `temperature` 大于 0，则通过给定的温度值使用 `softmax` 将 logits 转化为概率。如果 `temperature` 为 0，则使用 `torch.argmax` 选择具有最高 logit 的标记。
- 根据生成的概率或 `argmax` 确定下一个标记，并将其放置在 `tokens` 张量的相应位置。
- 检测到 EOS（结束序列）标记时，如果所有序列都到达 EOS 标记，生成过程将停止。
- 生成的标记保存在 `tokens` 张量中。

4. **处理生成的输出：**

- 对生成的标记进行处理，裁剪输出到所需的最大长度 (`max_gen_len`)。
- 如果 `echo` 为 `True`，生成的输出将包括提示本身。
- 如果遇到 `eos` 标记，则输出文本将在该点截断。

5. **返回：**方法返回一个包含生成标记的元组，并且如果 `logprobs` 为 `True`，还返回生成标记的对数概率。

该方法通过根据提示和之前生成的标记逐步预测下一个标记来生成文本。它使用指定的生成参数来控制生成文本的随机性和多样性。这个方法旨在与特定的模型架构和分词器一起使用，因为它依赖于它们的属性和方法来生成文本。