ckkissane update default save path 6de8761 · 4 months ago History

Code Blame 211 lines (179 loc) · 7.24 KB Raw

```python

from utils import *

from torch import nn
import pprint
import torch.nn.functional as F
from typing import Optional, Union
from huggingface_hub import hf_hub_download

from typing import NamedTuple

DTYPES = {"fp32": torch.float32, "fp16": torch.float16, "bf16": torch.bfloat16}
SAVE_DIR = Path("/workspace/crosscoder-model-diff-replication/checkpoints")

class LossOutput(NamedTuple):
    # loss: torch.Tensor
    l2_loss: torch.Tensor
    l1_loss: torch.Tensor
    l0_loss: torch.Tensor
    explained_variance: torch.Tensor
    explained_variance_A: torch.Tensor
    explained_variance_B: torch.Tensor

class CrossCoder(nn.Module):
    def __init__(self, cfg):
```

Symbols

Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols r

- const DTYPES
- const SAVE_DIR
- class LossOutput
  - const l2_loss
  - const l1_loss
  - const l0_loss
  - const explained_variance
  - const explained_variance_A
  - const explained_variance_B
- class CrossCoder
  - func __init__
  - func encode
  - func decode

```
        super().__init__()
        self.cfg = cfg
        d_hidden = self.cfg["dict_size"]
        d_in = self.cfg["d_in"]
        self.dtype = DTYPES[self.cfg["enc_dtype"]]
        torch.manual_seed(self.cfg["seed"])
        # hardcoding n_models to 2
        self.W_enc = nn.Parameter(
            torch.empty(2, d_in, d_hidden, dtype=self.dtype)
        )
        self.W_dec = nn.Parameter(
            torch.nn.init.normal_(
                torch.empty(
                    d_hidden, 2, d_in, dtype=self.dtype
                )
            )
        )
        self.W_dec = nn.Parameter(
            torch.nn.init.normal_(
                torch.empty(
                    d_hidden, 2, d_in, dtype=self.dtype
                )
            )
        )
        # Make norm of W_dec 0.1 for each column, separate per layer
        self.W_dec.data = (
            self.W_dec.data / self.W_dec.data.norm(dim=-1, keepdim=True) * self.cfg
        )
        # Initialise W_enc to be the transpose of W_dec
        self.W_enc.data = einops.rearrange(
            self.W_dec.data.clone(),
            "d_hidden n_models d_model -> n_models d_model d_hidden",
        )
        self.b_enc = nn.Parameter(torch.zeros(d_hidden, dtype=self.dtype))
        self.b_dec = nn.Parameter(
            torch.zeros((2, d_in), dtype=self.dtype)
        )
        self.d_hidden = d_hidden

```

func forward

func get_losses

func create_save_dir

func save

```
        self.to(self.cfg["device"])
        self.save_dir = None
        self.save_version = 0

    def encode(self, x, apply_relu=True):
        # x: [batch, n_models, d_model]
        x_enc = einops.einsum(
            x,
            self.W_enc,
            "batch n_models d_model, n_models d_model d_hidden -> batch d_hidden",
        )
        if apply_relu:
            acts = F.relu(x_enc + self.b_enc)
        else:
            acts = x_enc + self.b_enc
        return acts

    def decode(self, acts):
        # acts: [batch, d_hidden]
        acts_dec = einops.einsum(
            acts,
            self.W_dec,
            "batch d_hidden, d_hidden n_models d_model -> batch n_models d_model",
        )
        return acts_dec + self.b_dec

    def forward(self, x):
        # x: [batch, n_models, d_model]
        acts = self.encode(x)
        return self.decode(acts)

    def get_losses(self, x):
        # x: [batch, n_models, d_model]
        x = x.to(self.dtype)
        acts = self.encode(x)
        # acts: [batch, d_hidden]
        x_reconstruct = self.decode(acts)
        diff = x_reconstruct.float() - x.float()
        squared_diff = diff.pow(2)
```

```
        l2_per_batch = einops.reduce(squared_diff, 'batch n_models d_model -> batch
        l2_loss = l2_per_batch.mean()

        total_variance = einops.reduce((x - x.mean(0)).pow(2), 'batch n_models d_mo
        explained_variance = 1 - l2_per_batch / total_variance

        per_token_l2_loss_A = (x_reconstruct[:, 0, :] - x[:, 0, :]).pow(2).sum(dim=
        total_variance_A = (x[:, 0, :] - x[:, 0, :].mean(0)).pow(2).sum(-1).squeeze
        explained_variance_A = 1 - per_token_l2_loss_A / total_variance_A

        per_token_l2_loss_B = (x_reconstruct[:, 1, :] - x[:, 1, :]).pow(2).sum(dim=
        total_variance_B = (x[:, 1, :] - x[:, 1, :].mean(0)).pow(2).sum(-1).squeeze
        explained_variance_B = 1 - per_token_l2_loss_B / total_variance_B

        decoder_norms = self.W_dec.norm(dim=-1)
        # decoder_norms: [d_hidden, n_models]
        total_decoder_norm = einops.reduce(decoder_norms, 'd_hidden n_models -> d_h
        l1_loss = (acts * total_decoder_norm[None, :]).sum(-1).mean(0)

        l0_loss = (acts>0).float().sum(-1).mean()

        return LossOutput(l2_loss=l2_loss, l1_loss=l1_loss, l0_loss=l0_loss, explai

    def create_save_dir(self):
        base_dir = Path("/workspace/crosscoder-model-diff-replication/checkpoints")
        version_list = [
            int(file.name.split("_")[1])
            for file in list(SAVE_DIR.iterdir())
            if "version" in str(file)
        ]
        if len(version_list):
            version = 1 + max(version_list)
        else:
            version = 0
        self.save_dir = base_dir / f"version_{version}"
        self.save_dir.mkdir(parents=True)

    def save(self):
        if self.save_dir is None:
```

```
143             self.create_save_dir()
144         weight_path = self.save_dir / f"{self.save_version}.pt"
145         cfg_path = self.save_dir / f"{self.save_version}_cfg.json"
146 
147         torch.save(self.state_dict(), weight_path)
148         with open(cfg_path, "w") as f:
149             json.dump(self.cfg, f)
150 
151         print(f"Saved as version {self.save_version} in {self.save_dir}")
152         self.save_version += 1
153 
154     @classmethod
155     def load_from_hf(
156         cls,
157         repo_id: str = "ckkissane/crosscoder-gemma-2-2b-model-diff",
158         path: str = "blocks.14.hook_resid_pre",
159         device: Optional[Union[str, torch.device]] = None
160     ) -> "CrossCoder":
161         """
162         Load CrossCoder weights and config from HuggingFace.
163 
164         Args:
165             repo_id: HuggingFace repository ID
166             path: Path within the repo to the weights/config
167             model: The transformer model instance needed for initialization
168             device: Device to load the model to (defaults to cfg device if not spec
169 
170         Returns:
171             Initialized CrossCoder instance
172         """
173 
174         # Download config and weights
175         config_path = hf_hub_download(
176             repo_id=repo_id,
177             filename=f"{path}/cfg.json"
178         )
179         weights_path = hf_hub_download(
180             repo_id=repo_id,
181             filename=f"{path}/cc_weights.pt"
```

```
        )

        # Load config
        with open(config_path, 'r') as f:
            cfg = json.load(f)

        # Override device if specified
        if device is not None:
            cfg["device"] = str(device)

        # Initialize CrossCoder with config
        instance = cls(cfg)

        # Load weights
        state_dict = torch.load(weights_path, map_location=cfg["device"])
        instance.load_state_dict(state_dict)

        return instance

    @classmethod
    def load(cls, version_dir, checkpoint_version):
        save_dir = Path("/workspace/crosscoder-model-diff-replication/checkpoints")
        cfg_path = save_dir / f"{str(checkpoint_version)}_cfg.json"
        weight_path = save_dir / f"{str(checkpoint_version)}.pt"

        cfg = json.load(open(cfg_path, "r"))
        pprint.pprint(cfg)
        self = cls(cfg=cfg)
        self.load_state_dict(torch.load(weight_path))
        return self
```