







> 1  1  .gitignore 

> 199  demo.ipynb 

> BIN +743 KB diff_file.pdf 

> 1  lm-evaluation-harness 

▼  34  mingpt/model.py 

```

44      44      self.attn_dropout = nn.Dropout(config.attn_pdrop)
45      45      self.resid_dropout = nn.Dropout(config.resid_pdrop)
46      46      # causal mask to ensure that attention is only applied to
the left in the input sequence
47      -      self.register_buffer("bias",
torch.tril(torch.ones(config.block_size, config.block_size))
48      -      .view(1, 1, config.block_size,
config.block_size))
47      +      if config.use_ul2:
48      +          self.register_buffer("bias",
torch.ones(config.block_size, config.block_size).view(1, 1,
config.block_size, config.block_size))
49      +      elif not config.use_ul2:
50      +          self.register_buffer("bias",
torch.tril(torch.ones(config.block_size,
config.block_size)).view(1, 1, config.block_size,
config.block_size))
49      51      self.n_head = config.n_head
50      52      self.n_embd = config.n_embd
51      53
110     112      C.embd_pdrop = 0.1
111     113      C.resid_pdrop = 0.1
112     114      C.attn_pdrop = 0.1
115     +      C.use_ul2 = False
113     116      return C
114     117
115     118      def __init__(self, config):
116     119          super().__init__()
117     120          assert config.vocab_size is not None
118     121          assert config.block_size is not None
119     122          self.block_size = config.block_size

```

```

123 +         self.config = config
120 124
121 125         type_given = config.model_type is not None
122 126         params_given = all([config.n_layer is not None,
123         config.n_head is not None, config.n_embd is not None])
150 154         ))
151 155         self.lm_head = nn.Linear(config.n_embd, config.vocab_size,
124         bias=False)
152 156
153 -         # init all weights, and apply a special scaled init to the
125         residual projections, per GPT-2 paper
154 -         self.apply(self._init_weights)
155 -         for pn, p in self.named_parameters():
156 -             if pn.endswith('c_proj.weight'):
157 -                 torch.nn.init.normal_(p, mean=0.0,
126         std=0.02/math.sqrt(2 * config.n_layer))
157 +         '''This if statement is a change.'''
158 +         # If we are using a checkpoint, load it.
159 +         if config.checkpoint is not None:
160 +             self.checkpoint = torch.load(config.checkpoint)
161 +
127         self.transformer.load_state_dict(self.checkpoint['model_transformer
128         '])
162 +
129         self.lm_head.load_state_dict(self.checkpoint['model_lm_head'])
163 +         self.iter_num = self.checkpoint['iter_num']
164 +         self.checkpoint_num = self.checkpoint['checkpoint_num']
165 +         self.saved_loss = self.checkpoint['saved_loss']
166 +
167 +         else:
168 +             # init all weights, and apply a special scaled init to
130         the residual projections, per GPT-2 paper
169 +             self.checkpoint = None
170 +             self.apply(self._init_weights)
171 +             for pn, p in self.named_parameters():
172 +                 if pn.endswith('c_proj.weight'):
173 +                     torch.nn.init.normal_(p, mean=0.0,
131         std=0.02/math.sqrt(2 * config.n_layer))
158 174
159 175         # report number of parameters (note we don't count the
132         decoder parameters in lm_head)
160 176         n_params = sum(p.numel() for p in
133         self.transformer.parameters())
255 271
134         {"params": [param_dict[pn] for pn in
135         sorted(list(no_decay))], "weight_decay": 0.0},

```

```

256 | 272 |         ]
257 | 273 |         optimizer = torch.optim.AdamW(optim_groups,
      |      |         lr=train_config.learning_rate, betas=train_config.betas)
      | 274 | +         if self.checkpoint:
      | 275 | +
      |      |         optimizer.load_state_dict(self.checkpoint['optimizer_state_dict'])
      |      |         # This is a change.
258 | 276 |         return optimizer
259 | 277 |
260 | 278 |     def forward(self, idx, targets=None):
275 | 293 |         # if we are given some desired targets also calculate the
      |      |         loss
276 | 294 |         loss = None
277 | 295 |         if targets is not None:
278 |      | -         loss = F.cross_entropy(logits.view(-1,
      |      |         logits.size(-1)), targets.view(-1), ignore_index=-1)
      | 296 | +         loss = F.cross_entropy(logits.view(-1,
      |      |         logits.size(-1)), targets.view(-1), ignore_index=-1) # Changed
      |      |         class!
279 | 297 |
280 | 298 |         return logits, loss
281 | 299 |

```

> 41 mingpt/trainer.py

> 16 mingpt_jobscript

> BIN +37.5 KB mingpt_loss.png

> 506 output_results.txt

> 341 project2a.ipynb

> 88 project2a.py

> 520 project2b.ipynb

```
In [ ]: import torch
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from mingpt.model import GPT
from mingpt.utils import set_seed
from datasets import load_dataset
import torch
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from datasets import load_dataset
import numpy as np
from scipy.stats import norm
set_seed(3407)
```

```
In [ ]: # Load in UL2 tokenizer to see what's going on
from transformers import AutoTokenizer, GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

new_tokens = [f'<new_id_{i}>' for i in range(200)]
tokenizer.add_tokens(new_tokens)
tokenizer.add_tokens(['[S2S]', '[NLU]', '[NLG]'])

tokenizer
```

```
Out[ ]: PreTrainedTokenizer(name_or_path='gpt2', vocab_size=50257, model_max_len=1024, is_fast=False, padding_side='right', truncation_side='right', special_tokens={'bos_token': AddedToken("<|endoftext|>", rstrip=False, lstrip=False, single_word=False, normalized=True), 'eos_token': AddedToken("<|endoftext|>", rstrip=False, lstrip=False, single_word=False, normalized=True), 'unk_token': AddedToken("<|endoftext|>", rstrip=False, lstrip=False, single_word=False, normalized=True)})
```

```
In [ ]: dataset = load_dataset("togethercomputer/RedPajama-Data-1T-Sample", 'plain_text')
dataset = dataset['train']

# Custom dataset class for the Red Pajama dataset
class RedPajamaDataset(Dataset):
    def __init__(self, data, max_length=1024, ul2_switch=False):
        self.data = data
        self.tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
        self.tokenizer.add_tokens([f'new_id_{i}' for i in range(200)])
        self.tokenizer.add_tokens(['[S2S]', '[NLU]', '[NLG]'])
        self.tokenizer.pad_token_id = 50256
        self.max_length = max_length - 1
        self.vocab_size = len(self.tokenizer)
        self.token_dict = {'s': ['[S2S]', self._s_denoising], 'r': ['[NLU]', self._r_denoising]}
        self.ul2_switch = ul2_switch
```

```

def __len__(self):
    return len(self.data)

def __getitem__(self, idx):
    text = self.data[idx]['text']

    if self.ul2_switch:
        # Get the token to prepend and function to use.
        begin_id, func = self.token_dict[np.random.choice(['s', 'r', 'x'])]

        # Prepend token to string and tokenize
        text = begin_id + ' ' + text
        ids = self.tokenizer.encode(text, truncation=True, max_length=se

        # Return the tokens
        return func(ids, self.tokenizer)

    elif not self.ul2_switch:
        # Tokenize the text
        tokens = self.tokenizer.encode(text, add_special_tokens=True, ma

        # Split the tokens into chunks of max_length
        # Shift the tokens to get targets (excluding the [CLS] token)
        target_tokens = tokens[:, 1:].clone() # Exclude the [CLS] token
        tokens = tokens[:, :-1] # Exclude the last token to match the s
        return tokens, target_tokens

# Helper functions! These will implement the UL2 tokenization.
def _r_denoising(self, ids, tokenizer, corruption_pct=0.15, span_length=

    # Calculate the chance of corruption based on the corruption percent
    # mean span length, and the maximum span length
    chance = (corruption_pct / np.mean(span_length)) * (1 + np.max(span_

    # Variable to store the old tokens (before corruption)
    old_toks = None

    # Variables for tracking the number of steps to skip and tokens used
    steps_to_skip = 0
    tokens_used = 0

    ids_shape = ids.shape[1]
    # Iterate through the tokens in the input sequence
    for i in range(1, ids_shape):
        # Skip steps if needed (due to recent corruption)
        if steps_to_skip > 0:
            steps_to_skip -= 1
            continue

        # Randomly decide whether to corrupt the current token
        rnd = np.random.random()

```

```

        if rnd < chance:
            # Get the token used for masking (corruption)
            mask_token = tokenizer.convert_tokens_to_ids(new_tokens[tokenizer.eos_token_id])
            tokens_used += 1

            # Randomly choose a span length for corruption
            span = np.random.choice(span_length)

            # Update old_toks and ids with the corrupted span
            if old_toks is None:
                old_toks = torch.tensor([[mask_token]]) # Initialize old_toks
                old_toks = torch.cat((old_toks, ids[:, i:i + span]), dim=-1)
                ids = torch.cat((ids[:, :i], torch.tensor([[mask_token]] * span)), dim=-1)

                steps_to_skip = span
            else:
                old_toks = torch.cat((old_toks, torch.tensor([[mask_token]] * span)), dim=-1)
                ids = torch.cat((ids[:, :i], torch.tensor([[mask_token]] * span)), dim=-1)

            # Update steps_to_skip to avoid overlapping corruption
            steps_to_skip = span

        # Pad ids and old_toks to match the desired maximum length (R, X)
        ids = torch.cat((ids, torch.tensor([[tokenizer.eos_token_id] * (self.max_length - len(ids))])), dim=-1)
        old_toks = torch.cat((old_toks, torch.tensor([[tokenizer.eos_token_id] * (self.max_length - len(old_toks))])), dim=-1)

    return ids, old_toks

def _s_denoising(self, ids, tokenizer):
    # Get the length of our input
    len_ids = ids.shape[1]

    # Build Gaussian distribution of probabilities for each token
    vals = np.linspace(-2, 2, len_ids)
    p = norm.pdf(vals, loc=0, scale=1)

    # Normalize the probabilities and get the index to remove
    remove_index = np.random.choice(np.arange(len_ids // 2 - 15, len_ids // 2 + 15), 1)

    # Get the token we are using for this space
    mask_token = tokenizer.convert_tokens_to_ids(new_tokens[0])

    # Get the tokens we are removing
    old_toks = torch.cat((torch.tensor([[mask_token]] * span), ids[:, remove_index:]), dim=-1)

    # Mask the tokens
    ids = ids[:, :remove_index + 1].clone()
    ids[:, -1] = mask_token

```

```

# Pad ids and old_toks to match the desired maximum length (S)
ids = torch.cat((ids, torch.tensor([[tokenizer.eos_token_id] * (self
old_toks = torch.cat((old_toks, torch.tensor([[tokenizer.eos_token_i

return ids, old_toks

```

```

def _x_denoising(self, ids, tokenizer, corruption_pct=0.50, span_length=
return self._r_denoising(ids, tokenizer, corruption_pct, span_length

```

```

# Create an instance of the custom dataset
red_pajama_dataset_F = RedPajamaDataset(dataset)
red_pajama_dataset_T = RedPajamaDataset(dataset, ul2_switch=True)

```

```

Found cached dataset red_pajama-data-1_t-sample (/Users/dylanskiner/Desktop/CS 674 Projects/MinGPT_UL2/datasets/togethercomputer__red_pajama-data-1_t-sample/plain_text/1.0.0/6ea3bc8ec2e84ec6d2df1930942e9028ace8c5b9d9143823cf911c50bbd92039)

```

```

0%|          | 0/1 [00:00<?, ?it/s]

```

```
In [ ]: # create a GPT instance
from mingpt.model import GPT
import os

checkpoint_dir = 'red_pajama'
dir_path = f'./checkpoints/{checkpoint_dir}'

if not os.path.exists(dir_path):
    # If the directory doesn't exist, create it
    os.makedirs(dir_path)
    checkpoints = os.listdir(dir_path)
else:
    checkpoints = os.listdir(dir_path)

checkpoints.sort()

model_config = GPT.get_default_config()
model_config.model_type = 'gpt-nano'
model_config.vocab_size = red_pajama_dataset_T.vocab_size
model_config.block_size = red_pajama_dataset_T.max_length
# model_config.checkpoint = f'checkpoints/{checkpoint_dir}/' + checkpoints[-1]
model_config.checkpoint = None
model_config.use_ul2 = True
model = GPT(model_config)

model_config = GPT.get_default_config()
model_config.model_type = 'gpt-nano'
model_config.vocab_size = red_pajama_dataset_F.vocab_size
model_config.block_size = red_pajama_dataset_F.max_length
# model_config.checkpoint = f'checkpoints/{checkpoint_dir}/' + checkpoints[-1]
model_config.checkpoint = None
model_config.use_ul2 = False
model2 = GPT(model_config)

number of parameters: 2.56M
number of parameters: 2.56M
```



```
In [ ]: # create a Trainer object
from mingpt.trainer import Trainer
iters = 500

train_config = Trainer.get_default_config()
train_config.learning_rate = 5e-4 # the model we're using is so small that w
train_config.max_iters = iters + model.iter_num if model_config.checkpoint e
train_config.num_workers = 0
train_config.checkpoint_iters = 100 # This is a change
train_config.batch_size = 1
train_config.checkpoint_name = f'{checkpoint_dir}/checkpoint_ul2' # This is
trainer = Trainer(train_config, model, red_pajama_dataset_T)

train_config = Trainer.get_default_config()
train_config.learning_rate = 5e-4 # the model we're using is so small that w
train_config.max_iters = iters + model.iter_num if model_config.checkpoint e
train_config.num_workers = 0
train_config.checkpoint_iters = 100 # This is a change
train_config.batch_size = 1
train_config.checkpoint_name = f'{checkpoint_dir}/checkpoint_break' # This
trainer2 = Trainer(train_config, model2, red_pajama_dataset_F)

running on device cpu
running on device cpu
```

```
In [ ]: def batch_end_callback(trainer):
    if trainer.iter_num % 100 == 0:
        print(f"iter_dt {trainer.iter_dt * 1000:.2f}ms; iter {trainer.iter_r
trainer.set_callback('on_batch_end', batch_end_callback)

trainer.run()

# Plot the loss
losses = trainer.curr_loss
x = 10
new_losses = np.mean(np.array(losses).reshape(-1, x), axis=1)

plt.plot(np.arange(len(new_losses)), new_losses, label='UL2')

def batch_end_callback(trainer):
    if trainer2.iter_num % 100 == 0:
        print(f"iter_dt {trainer2.iter_dt * 1000:.2f}ms; iter {trainer2.iter
trainer2.set_callback('on_batch_end', batch_end_callback)

trainer2.run()

# Plot the loss
losses = trainer2.curr_loss
x = 10
new_losses = np.mean(np.array(losses).reshape(-1, x), axis=1)
plt.plot(np.arange(len(new_losses)), new_losses, label='Regular')
plt.title('Training of MinGPT on The Pile')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.show()

iter_dt 0.00ms; iter 0: train loss 10.69954
iter_dt 971.65ms; iter 100: train loss 5.62758
iter_dt 501.43ms; iter 200: train loss 1.12558
iter_dt 481.43ms; iter 300: train loss 3.26710
iter_dt 506.68ms; iter 400: train loss 1.15159
iter_dt 0.00ms; iter 0: train loss 10.83819
iter_dt 178.93ms; iter 100: train loss 9.08965
iter_dt 941.06ms; iter 200: train loss 8.23680
iter_dt 32.46ms; iter 300: train loss 9.81027
iter_dt 481.70ms; iter 400: train loss 7.58708
```



```
In [ ]: # Show that ul2 improves performance on checkpoint 1
# create a GPT instance
from mingpt.model import GPT
import os

checkpoint_dir = 'red_pajama'
dir_path = f'./checkpoints/{checkpoint_dir}'

if not os.path.exists(dir_path):
    # If the directory doesn't exist, create it
    os.makedirs(dir_path)
    checkpoints = os.listdir(dir_path)
else:
    checkpoints = os.listdir(dir_path)

checkpoints.sort()

# UL2
model_config = GPT.get_default_config()
model_config.model_type = 'gpt-nano'
model_config.vocab_size = red_pajama_dataset.T.vocab_size
```

```

model_config.block_size = red_pajama_dataset_T.max_length
model_config.checkpoint = f'checkpoints/{checkpoint_dir}/checkpoint_break_1.
model_config.use_ul2 = True
model = GPT(model_config)

# Regular
model_config = GPT.get_default_config()
model_config.model_type = 'gpt-nano'
model_config.vocab_size = red_pajama_dataset_F.vocab_size
model_config.block_size = red_pajama_dataset_F.max_length
model_config.checkpoint = f'checkpoints/{checkpoint_dir}/checkpoint_break_1.
model_config.use_ul2 = False
model2 = GPT(model_config)

# create a Trainer object
from mingpt.trainer import Trainer
iters = 200

train_config = Trainer.get_default_config()
train_config.learning_rate = 5e-4 # the model we're using is so small that w
train_config.max_iters = iters + model.iter_num if model_config.checkpoint e
train_config.num_workers = 0
train_config.checkpoint_iters = 500 # This is a change
train_config.batch_size = 1
train_config.checkpoint_name = f'{checkpoint_dir}/checkpoint_ul2' # This is
trainer = Trainer(train_config, model, red_pajama_dataset_T)

train_config = Trainer.get_default_config()
train_config.learning_rate = 5e-4 # the model we're using is so small that w
train_config.max_iters = iters + model.iter_num if model_config.checkpoint e
train_config.num_workers = 0
train_config.checkpoint_iters = 500 # This is a change
train_config.batch_size = 1
train_config.checkpoint_name = f'{checkpoint_dir}/checkpoint_break' # This
trainer2 = Trainer(train_config, model2, red_pajama_dataset_F)

def batch_end_callback(trainer):
    if trainer.iter_num % 50 == 0:
        print(f"iter_dt {trainer.iter_dt * 1000:.2f}ms; iter {trainer.iter_n
trainer.set_callback('on_batch_end', batch_end_callback)

trainer.run()

# Plot the loss
losses = trainer.curr_loss
print(losses)
x = 5
new_losses = np.mean(np.array(losses).reshape(-1, x), axis=1)
plt.plot(np.arange(len(new_losses)), new_losses, label='UL2')

```

```

def batch_end_callback(trainer):
    if trainer2.iter_num % 50 == 0:
        print(f"iter_dt {trainer2.iter_dt * 1000:.2f}ms; iter {trainer2.iter}
trainer2.set_callback('on_batch_end', batch_end_callback)

trainer2.run()

# Plot the loss
losses = trainer2.curr_loss
x = 5
new_losses = np.mean(np.array(losses).reshape(-1, x), axis=1)
plt.plot(np.arange(len(new_losses)), new_losses, label='Regular')
plt.title('Training UL2 and Regular On Regular\'s Checkpoint 1')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

number of parameters: 2.56M

number of parameters: 2.56M

running on device cpu

running on device cpu

iter_dt 543.64ms; iter 250: train loss 8.90695

iter_dt 474.03ms; iter 300: train loss 7.24735

iter_dt 504.72ms; iter 350: train loss 4.78618

iter_dt 482.22ms; iter 400: train loss 3.18715

[tensor(11.8283), tensor(10.6837), tensor(10.7594), tensor(12.2985), tensor(10.2331), tensor(10.0490), tensor(10.7800), tensor(10.4051), tensor(10.4136), tensor(10.5508), tensor(10.3679), tensor(10.6896), tensor(10.5892), tensor(10.4705), tensor(10.3906), tensor(10.2769), tensor(10.2266), tensor(10.1427), tensor(10.1367), tensor(10.0487), tensor(10.0865), tensor(10.0414), tensor(9.9979), tensor(9.9229), tensor(9.7591), tensor(9.7966), tensor(9.7747), tensor(9.7576), tensor(9.6463), tensor(9.6980), tensor(9.6802), tensor(9.5646), tensor(9.5861), tensor(9.3779), tensor(9.4754), tensor(9.4247), tensor(9.3315), tensor(9.4123), tensor(9.2360), tensor(9.3217), tensor(9.0653), tensor(9.0450), tensor(8.6310), tensor(9.2764), tensor(8.8116), tensor(9.0521), tensor(9.8390), tensor(9.6590), tensor(9.1361), tensor(8.9070), tensor(9.2218), tensor(8.5756), tensor(8.9577), tensor(8.0849), tensor(8.6252), tensor(9.8494), tensor(8.5421), tensor(8.7333), tensor(9.1506), tensor(8.4016), tensor(8.4268), tensor(8.4636), tensor(8.6159), tensor(8.7648), tensor(8.2736), tensor(8.6904), tensor(8.5776), tensor(8.2979), tensor(8.2501), tensor(8.3470), tensor(8.3438), tensor(8.0693), tensor(8.0823), tensor(8.0209), tensor(8.3071), tensor(8.0350), tensor(7.8821), tensor(7.7855), tensor(8.7389), tensor(8.5559), tensor(7.8719), tensor(8.2810), tensor(7.9316), tensor(8.1112), tensor(7.9912), tensor(8.1203), tensor(7.8876), tensor(7.7702), tensor(7.9138), tensor(8.4398), tensor(7.3866), tensor(7.6399), tensor(7.2900), tensor(7.2236), tensor(7.0840), tensor(7.0955), tensor(7.5591), tensor(7.0550), tensor(7.0921), tensor(7.2474), tensor(6.8987), tensor(6.8797), tensor(7.2069), tensor(7.0060), tensor(6.8601), tensor(7.1307), tensor(7.1364), tensor(7.0231), tensor(6.7697), tensor(7.4277), tensor(6.9369), te

```
nsor(6.5008), tensor(7.0173), tensor(6.9349), tensor(6.5284), tensor(6.2185
), tensor(6.1363), tensor(6.3819), tensor(6.0102), tensor(6.7265), tensor(6
.3413), tensor(5.7782), tensor(6.3322), tensor(6.1029), tensor(5.8656), te
nsor(6.1411), tensor(5.9497), tensor(5.5931), tensor(5.9326), tensor(5.6386
), tensor(5.9088), tensor(5.6203), tensor(5.3520), tensor(5.6596), tensor(5.
1824), tensor(5.1969), tensor(5.3858), tensor(5.5697), tensor(5.0174), tens
or(5.4399), tensor(5.0367), tensor(4.8776), tensor(5.8094), tensor(5.0260),
tensor(5.1450), tensor(4.9619), tensor(4.9448), tensor(6.9152), tensor(4.61
83), tensor(4.7862), tensor(4.6500), tensor(4.4318), tensor(4.8743), tensor
(6.4284), tensor(4.9083), tensor(4.5193), tensor(4.2447), tensor(6.1127), t
ensor(4.2594), tensor(3.9423), tensor(4.3336), tensor(3.8836), tensor(4.063
2), tensor(5.8834), tensor(4.0293), tensor(4.1404), tensor(4.9000), tensor(
3.4799), tensor(3.6729), tensor(5.3732), tensor(5.8962), tensor(4.0350), te
nsor(3.6107), tensor(3.8380), tensor(3.4443), tensor(3.9603), tensor(3.8562
), tensor(4.8296), tensor(3.2377), tensor(3.7922), tensor(4.2662), tensor(2
.8044), tensor(4.3058), tensor(2.9410), tensor(6.1025), tensor(5.1841), ten
sor(3.3615), tensor(3.1630), tensor(2.9210), tensor(2.8610), tensor(2.9173)
, tensor(3.5551), tensor(2.8490), tensor(3.2203), tensor(2.6683), tensor(3.
9535), tensor(2.4661), tensor(2.2567), tensor(2.2690), tensor(3.1872)]
iter_dt 172.41ms; iter 250: train loss 8.03310
iter_dt 264.90ms; iter 300: train loss 7.66088
iter_dt 476.78ms; iter 350: train loss 7.52221
iter_dt 151.13ms; iter 400: train loss 9.00874
```

Training UL2 and Regular On Regular's Checkpoint 1

