Final Project Report: Building GPT-2

June 8, 2025

Contents

1 Basic information

The proposal should have the following information at the top:

- **Title:** A Framework for Building and Fine-Tuning GPT-2 Models
- **Team members:**
  - Songlin Songlin (`zhousl24@tsinghua.edu.cn`)
    Contribution: Doing theoratical research and writing project proposal, implementing the structure of GPT-2 within the process of Stanford NLU Final Project Guidence, which is the first part of our project.
    Sun Zhongwei(`sun-zw23@mails.tsinghua.edu.cn`)
    Shibo Dai(`dsb24@mails.tsinghua.edu.cn`)
- **Custom or Default Project:** Default Project—Building GPT-2

2 Project description

- **Main Goals:**
  - Use the codebase that is adapted from the final project of Stanford CS224n with pre-trained of GPT-2 and implement some important components of the GPT-2 model to better understand its architecture. See this link.
  - Fine-tune the pretrained model on multiple downstream NLP tasks: 1) Sentiment Analysis, 2) Paraphrase Identification, and 3) Sonnet Generation.
  - Train a mini-GPT model from scratch without loading any pretrained transformer weights (word embeddings are allowed).
- **Downstream NLP Tasks:**
  - Sentiment Analysis on IMDB reviews (binary classification).
  - Paraphrase Detection on MRPC (sentence-pair classification).
  - Sonnet Generation given a short Shakespearean prompt (autoregressive generation).
- **Data Usage:**
  - *Pre-training:* None. The pre-trained data is provided within *NLU-Course-Project-GPT-and-Downstream-Tasks*
  - All we need to do is complete the implements of the missing block of algorithm such as Multi-headed Attention, Adam Optimization and position-wise Feed-Frward Network.
  - *Fine-tuning and Prediction for Sentiment Analysis:*
    - Stanford Sentiment Treebank (SST) (8,544 train / 1,101 dev / 1,101 test).
    - CFIMDB dataset (1,701 train / 245 dev / 488 test)
  - *Fine-tuning and Prediction for Paraphrase Detection:*
    - Quora dataset (141,506 train / 20,215 dev / 40,431 test)
  - Test sets and methods are also provided in the e final project of Stanford CS224n.
- **Methods:**
  - Use the GPT-2 structure provided within *NLU-Course-Project-GPT-and-Downstream-Tasks*: GPT-2 small (12 layers, 12 heads, 768-dim).
  - Training with Adam, linear warmup (10 % steps) + linear decay.
  - Fine-tuning: add classification heads for IMDB/MRPC; use beam search for sonnet generation.

- **Baselines:**
  - For SA:
    - Last Linear Layer for SST: Dev Accuracy: 0.462
    - Full Model for SST: Dev Accuracy: 0.513
    - Last Linear Layer for CFIMDB: Dev Accuracy: 0.861
    - Full Model for CFIMDB: Dev Accuracy: 0.976
  - For PD:
    - **IMDB Sentiment Analysis:**
    - BERTbase finetuned (Devlin et al., 2019): 94.5% accuracy.
    - GPT-2 zeroshot (Radford et al., 2019): 70.3% accuracy.
    - **MRPC Paraphrase Detection:**
    - BERTbase finetuned (Devlin et al., 2019): 88.9% accuracy, F1 = 89.2%.
    - GPT-2 zeroshot (Radford et al., 2019): 65.1% F1.
  - For Sonnet: Published BERT-base fine-tuned results (GLUE leaderboard).
- **Evaluation Metrics:**
  - See Baselines for SA.
  - *Accuracy* and *F1* for IMDB and MRPC.
  - *CHRF score* for sonnet generation.

3 Songlin Zhou: Implementing GPT-2

3.1 Folder and File Roles

`__pycache__` A local snapshot of the *official* HuggingFace GPT-2 weights; it is loaded on demand by `base_gpt.py` to avoid an on-line download.

`Downstream-tasks/` Example finetuning code for text classification, summarisation, *etc.* not required in my assignment.

`models/` (central folder)

- `__pycache__` Compiled bytecode and cached tensors; reuse of the pretrained weights.
- `base_gpt.py` Generic *foundation* class (`GPTPreTrainedModel`): stores the `config`, performs weight initialisation and keeps track of global `dtype`. Every concrete GPT variant inherits from this class.
- `gpt2.py` Instantiates the Transformer stack defined in `base_gpt.py`, wiring the attention blocks, feedforward layers, and the final LM head.

modules/ attention.py Full implementation of *Causal Self-Attention* (multi-head, padding and causal masking, output projection).

gpt2_layer.py Combines `attention.py` with *Pre-LayerNorm* and the feed-forward MLP to form one Transformer block (`Dropout` → `Dense` → `Residual` per layer).

test/ Minimal unit tests for GPT-2 model forward-pass correctness and for the `Adam` optimiser.

```
$ cd my-gpt2-project
$ python3 test/optimizer_test.py
Your GPT2 implementation is correct!
$ python3 test/sanity_check.py
Optimizer test passed!
```

The following steps demonstrate the verification process for the GPT-2 implementation and optimizer:

config.py Default hyper-parameters (hidden size, #layers, dropout probability, learning rate, ...).

optimizer.py A clean implementation of the `Adam` algorithm.

README.md Step-by-step running instructions.

Code Repository
https://github.com/Pinarinith/my-gpt2-project

3.2 Attention Module (attention.py)

def __init__(self, config): Initializes the three linear projection layers for **query**, **key** and **value**, each of shape $[H] \to [H]$, where H = hidden_size. Imports all relevant hyperparameters from `config` (e.g. `num_attention_heads`, `hidden_size`, `attention_dropout`). Defines a `Dropout` layer that is applied to the *normalized* attention scores following the original Transformer paper. Although somewhat unusual, we empirically observe that this yields better generalisation.

def transformer(self, x, linear_layer): Given an input feature tensor

$$x \in \mathbb{R}^{B \times T \times H},$$

where B is batch size, T is sequence length and H is hidden dimension, applies a single linear projection (`linear_layer`) to obtain proj $\in \mathbb{R}^{B \times T \times H}$, then reshapes into multiple heads:

proj = `rearrange`(proj, ‘ $b\ t\ (h\ d) \to b\ t\ h\ d'$, h = num_attention_heads),

which yields a tensor of shape $[B, T, h, d]$ and enables parallel computation over h attention heads of size $d = H/h$ for speed and efficiency.

`def attention(self, key, query, value, attention_mask):` Computes masked, scaled dotproduct selfattention. Given

$$Q, K \in \mathbb{R}^{B\times h\times T\times d}, \quad V \in \mathbb{R}^{B\times h\times T\times d},$$

it first forms the unnormalized scores

$$S = \frac{Q\,K^{\top}}{\sqrt{d_k}} \quad \Big[S \in \mathbb{R}^{B\times h\times T\times T}\Big],$$

then applies the mask:

$$S \leftarrow S + \text{attention\_mask},$$

and finally normalises with a dropout to keep robustness:

$$A = \text{softmax}(S,\ \text{dim} = -1), \quad A = \text{Dropout}(A).$$

The output context is

$$\text{context} = A\,V \in \mathbb{R}^{B\times h\times T\times d},$$

which is reshaped back to $[B, T, H]$ before returning.

`def forward(self, hidden_states, attention_mask):` In the feedforward neural network we apply masking of future information to prevent the model from being influenced by future tokens during training, which would degrade prediction performance. Each `forward` call outputs an attention value that becomes a subcomponent in `gpt2_layer.py`.

3.3 Layer Module (`gpt2_layer.py`)

`def __init__(self, config):` Initializes all submodules for one Transformer block:

- Multihead selfattention (`CausalSelfAttention`) parameters
- Two linear layers for the positionwise feedforward network
- Two `LayerNorm` instances for PreLN
- Dropout probabilities from `config`

`def add(self, residual, sublayer_out, dense_layer, dropout):`

- Projects `sublayer_out` back to the hidden dimension via `dense_layer`.
- Applies `dropout` for regularisation.
- Adds the original `residual` tensor (residual connection).

`def forward(self, hidden_states, attention_mask):` Implements one full block in the PreLayerNorm style:

1. **MultiHead SelfAttention**
   (1a) *Prenormalize*: apply `LayerNorm` to `hidden_states`.
   (1b) *Attention*: compute selfattention with `attention_mask`.
   (1c) *Dropout + Residual*: use `add(...)` to project, drop out, and add back the input.
2. **Positionwise FeedForward**
   (2a) *Prenormalize*: apply `LayerNorm` to the attention output.
   (2b) *MLP*: apply `Linear GELU Linear`.
   (2c) *Dropout + Residual*: use `add(...)` again to project, drop out, and add back.

3.4 GPT2 Module Summary (`gpt2.py`)

`def __init__(self, config):` Initializes embeddings, Transformer blocks, layernorm and weights.

`def embed(self, input_ids):` Token + position lookup + dropout.

`def encode(self, hidden, mask):` Applies each layer with the extended attention mask.

`def forward(self, input_ids, attention_mask):` Embed encode final norm select last token.

```
x    = self.embed(input_ids)
x    = self.encode(x, attention_mask)
x    = self.ln_f(x)
last = x[torch.arange(B), attention_mask.sum(1)-1]
return {'last_hidden_state': x, 'last_token': last}
```

`def hidden_state_to_token(self, h):` Weighttie projection to vocabulary logits.

```
return h @ self.wte.weight.T
```

`@classmethod def from_pretrained(cls, ...):` Load HuggingFace weights into this model.

```
hf    = HFModel.from_pretrained(model_name)
model = cls(our_cfg)
model.load_state_dict(filter_weights(hf.state_dict()))
return model
```

3.5 Optimizer Module (`optimizer.py`)

`def __init__(self, params, lr, betas, eps, weight_decay, correct_bias):`
Validates hyperparameters, stores defaults, and calls the base `Optimizer` constructor.

`def step(self, closure=None):` Performs one optimization step of AdamW:

(1) **(Optional)** call `closure()` to recompute loss.

(2) Loop over each parameter group and each parameter `p`:

(3) **State Initialization** (first time only):

```
if len(state)==0:
    state["step"]       = 0
    state["exp_avg"]    = torch.zeros_like(p.data)
    state["exp_avg_sq"] = torch.zeros_like(p.data)
```

(4) **Moment Updates**:

```
state["step"] += 1
exp_avg.mul_(beta1).add_(grad, alpha=1-beta1)
exp_avg_sq.mul_(beta2).addcmul_(grad, grad, value=1-beta2)
```

(5) **BiasCorrected Step Size**:

```
if correct_bias:
    bc1 = 1 - beta1**t
    bc2 = 1 - beta2**t
    step_size = lr * math.sqrt(bc2) / bc1
else:
    step_size = lr
```

(6) **Parameter Update**:

```
denom = exp_avg_sq.sqrt().add_(eps)
p.data.addcdiv_(exp_avg, denom, value=-step_size)
```

(7) **Decoupled Weight Decay**:

```
if weight_decay != 0:
    p.data.add_(p.data, alpha=-lr * weight_decay)
```

Returns the (possibly recomputed) loss.

4 Sun Zhongwei: Training and evaluating a mini-GPT model

This part documents the process of training and evaluating a mini-GPT model for generating text in the style of William Shakespeare. The model was implemented using PyTorch and trained from scratch without pretrained transformer weights. We experimented with different architectures and hyperparameters to understand their impact on model performance.

Code Repository
https://gitee.com/crazyysun/minigpt

4.1 Dataset Preparation

The dataset used was `tinyshakespeare.txt` containing Shakespeare's works. The data was processed as follows:

- Character-level tokenization was performed
- The dataset was split into:
  - Training set: 80% of the data
  - Validation set: 10% of the data
  - Test set: 10% of the data
- Context length (block size) was set to 128 characters

The vocabulary size was determined to be `65` unique characters after processing.

4.2 Model Architecture

The mini-GPT model follows the standard Transformer architecture with the following components:

- Token embeddings layer
- Positional embeddings
- Multiple transformer layers with self-attention
- Feed-forward networks
- Final linear projection to vocabulary size

4.3 Training Procedure

We experimented with multiple configurations as specified in the `config.json` file. Two representative configurations are highlighted below:

Configuration 1: Smaller Model

- Layers: 4
- Attention heads: 4
- Embedding dimension: 256
- Dropout: 0.1
- Learning rate: 3e-4

Configuration 2: Larger Model

- Layers: 8
- Attention heads: 8
- Embedding dimension: 512
- Dropout: 0.1
- Learning rate: 1e-4

Training was performed with:

- Batch size: 64
- Epochs: 5
- Optimizer: AdamW
- Evaluation interval: Half of training set size

4.4 Results

The results of the two configurations are shown below, for more details see the training log on the repository website.

Configuration 1: Smaller Model

The model architecture used the following parameters:

- Transformer Layers: 4
- Attention Heads: 4
- Embedding Dimension: 256
- Dropout Rate: 0.1
- Learning Rate: 0.0003

Training Results

The model achieved optimal performance after 20 epochs:

- Best Validation Loss: 0.295
- Final Batch: 13,940

Test Performance

The model was evaluated on a held-out test set with the following results:

Table 1: Test Performance Metrics

Metric	Value
Average Loss	0.2950
Token Accuracy	91.45%
Perplexity	1.34
Vocabulary Diversity	0.3484
Sequence Accuracy	91.32%
Exact Match Rate	0.00%

Generated Text Samples

The model was tested with different generation parameters:

Sample 1: Temperature=0.8, Top-k=50

```
HAMLET: To be or not to be talk'd for, to accept it.

KING RICHARD II:
And buried, gentle Tyrrel?

TYRREL:
The grey-office with such a deceitful soul
That hath two them kings murder'd:
But they shall set up forward towards on their market-place;
Call me to the master?

Page:
No, no, no, no; for she hath praised his master.

TRANIO:
Sir, how now, sir! What's the matter? My dear inclinest maid
That wash'd my hands with thy sword sweet wives and with
Rainous with silver savage and talk of perpetual
To greet the traitor's na
```

Sample 2: Temperature=0.5, Top-k=50

```
HAMLET: To be or not to be endured!

GLOUCESTER:
I hope the king is not here pass'd for so fair?
She may year more, may betide to my suit;
And so, here comes Katharina, thy supper is:
Where is the best of Katharina must be.

SAMPSON:
The better for his father's death, and I will speak:
There is no more but strange.

Shepherd:
I cannot speak, Northumberland, I crave it,
When I shall not do it.

CLARENCE:
Then this is the duke my father, cry 'Come;'
O, 'alce on father, but not who came.

First Murderer:
Ay, stay we now no
```

Sample 3: Temperature=0.8, Top-k=0 (No restriction)

```
HAMLET: To be or not to be so resolved
As vaults should be thine. To have heard your person;
'Tis more, thanks, my gracious lord.

DUKE VINCENTIO:

ISABELLA:

DUKE VINCENTIO:
Let's hear. O sir, ha! Away with him!

LUCIO:
A hundred moulderous leisure; the worst is full of
meat, and here be absent: it may be denied but my
believe in Vienna, is the grave of men alive?

PETRUCHIO:
It was an errand.
```

```
HORTENSIO:
Sir, fairly dream!

GREMIO:
Horse! a devil, a red son, or an apparent,
Of such post surfeits that we have always bef
```

Analysis

Key observations from the experiment:

- The model achieved excellent token accuracy (91.45%) while maintaining low perplexity (1.34)
- Higher temperature (0.8) produced more creative but less coherent text
- Lower temperature (0.5) resulted in more conservative but grammatically sound output
- Disabling Top-k sampling led to slightly less coherent generations
- The zero exact match rate confirms the model isn't simply memorizing text

Here is another example in a similar format, simulating results from a different model configuration:

Configuration 2: Medium Model

The model architecture used the following parameters:

- Transformer Layers: 8
- Attention Heads: 8
- Embedding Dimension: 512
- Dropout Rate: 0.1
- Learning Rate: 0.0001

Training Results

The model achieved optimal performance after 25 epochs:

- Best Validation Loss: 0.100
- Final Batch: 17,425

Test Performance

The model was evaluated on a held-out test set with the following results:

Table 2: Test Performance Metrics

Metric	Value
Average Loss	0.1001
Token Accuracy	96.74%
Perplexity	1.11
Vocabulary Diversity	0.3766
Sequence Accuracy	96.46%
Exact Match Rate	0.00%

Generated Text Samples

The model was tested with different generation parameters:

Sample 1: Temperature=0.8, Top-k=50

```
HAMLET: To be or not to be spoke withal.
Here comes his servant: how now, Catesby,
What says he?

CATESBY:
My lord: he doth entreat your grace;
To visit him to-morrow or next day:
He is within, with two right reverend fathers,
Divinely bent to meditation;
And no worldly suit would he be moved,
To draw him from his holy exercise.

BUCKINGHAM:
Return, good Catesby, to thy lord again;
Tell him, myself, the mayor and citizens,
In deep designs and matters of great moment,
No less importing than our general good,
Are come to ha
```

Sample 2: Temperature=0.5, Top-k=50

```
HAMLET: To be or not to be spoke withal.
Here comes his servant: how now, Catesby,
What says he?

CATESBY:
My lord: he doth entreat your grace;
To visit him to-morrow or next day:
He is within, with two right reverend fathers,
Divinely bent to meditation;
```

```
And no worldly suit would he be moved,
To draw him from his holy exercise.

BUCKINGHAM:
Return, good Catesby, to thy lord again;
Tell him, myself, the mayor and citizens,
In deep designs and matters of great moment,
No less importing than our general good,
Are come to ha
```

Sample 3: Temperature=0.8, Top-k=0 (No restriction)

```
HAMLET: To be or not to be talked on, you shall bear more.

CORIOLANUS:
The gods begin to mock me. I, that now
Refused most princely gifts, am bound to beg
Of my lord general.

COMINIUS:
Take't; 'tis yours. What is't?

CORIOLANUS:
I sometime lay here in Corioli
At a poor man's house; he used me kindly:
He cried to me; I saw him prisoner;
But then Aufidius was within my view,
And wrath o'erwhelm'd my pity: I request you
To give my poor host freedom.

COMINIUS:
O, well begg'd!
Were he the butcher of my son, he should
Be fre
```

Analysis

Key observations from the experiment:

- The model showed improved performance over the smaller variant with a token accuracy of 96.74% and perplexity of 1.11
- At temperature 0.8, the model generated rich and contextually appropriate text while maintaining coherence
- Lower temperature settings produced more conservative but still stylistically consistent output

- Vocabulary diversity increased slightly compared to the smaller model, indicating better use of language variation
- Despite high sequence accuracy, the zero exact match rate suggests strong generalization rather than memorization
- Generated samples demonstrated understanding of Shakespearean style and structure

4.5 Conclusion

We successfully trained and evaluated mini-GPT models for Shakespearean text generation. The larger model with 8 layers and 512 embedding dimension performed better but at the cost of increased computational requirements. Future work could explore more sophisticated sampling techniques and larger context windows.