AIL861 Assignment

Mrinal Tyagi (2024AIY7614)

November 2025

Github URL: `https://github.com/MrinalTyagi/AIL861_Assignment1`
Model Checkpoints: `https://huggingface.co/datasets/tr1gg3r/ail861_checkpoints`

1 Implementing a Decoder-Only Transformer

1.1 Pre-Training Dataset

The pretraining is performed on TinyStories dataset. Spacy is used for preprocessing the dataset. The dataset is preprocessed to not preserve case as the model is only trained on lowercase. The following script is used to perform preprocessing:

```
import pandas as pd
from datasets import load_from_disk, load_dataset
from datasets import Dataset, DatasetDict
from tqdm import tqdm
import re
import spacy

spacy_model = spacy.load("en_core_web_sm", disable=["parser", "ner",
↪  "textcat"])
dataset = load_dataset("roneneldan/TinyStories")


def process(subset):
    texts = subset["text"]
    tokens = []
    for doc in spacy_model.pipe(texts, batch_size=512 * 4, n_process=1):
        tokens.append([token.text.lower() for token in doc if
        ↪  token.text.isprintable() and not token.is_space])
    return {"tokens": tokens}


dataset["train"] = dataset["train"].map(
    process, batched=True, batch_size=512 * 4, num_proc=8
)
dataset["validation"] = dataset["validation"].map(
```

```
    process, batched=True, batch_size=512 * 4, num_proc=8
)

dataset.save_to_disk("TinyStories_processed")
```

The vocabulary of the most common 30000 tokens is created. It includes special tokens $[< pad >, < unk >, < bos >, < eos >]$. We use fasttext embedding as initialization for the transformer model. The following script is used in order to create a matrix of fasttext embedding.

```
import fasttext
import json
from tqdm import tqdm
import torch

fasttext_model = fasttext.load_model("cc.en.300.bin")
vocab = json.load(open("vocab_dict.json"))
embedding_matrix = None
special_tokens = ["<pad>", "<unk>", "<bos>", "<eos>"]
input_dim = 300
for word, idx in tqdm(vocab.items()):
    if word not in special_tokens:
        embed =
        ↪  torch.tensor(fasttext_model.get_word_vector(word)).unsqueeze(0)
        if embedding_matrix is None:
            embedding_matrix = embed
        else:
            embedding_matrix = torch.cat([embedding_matrix, embed],
            ↪  dim=0)
embedding_matrix = torch.cat(
    [torch.empty(size=(len(special_tokens), input_dim)).normal_(mean=0,
    ↪  std=0.02), embedding_matrix], dim=0
)
print("Embedding save of shape: ", embedding_matrix.shape)
torch.save(embedding_matrix, "fasttext_embedding.pt")
```

We use a context size of 64 while tokenizing the dataset and an embedding dimension of 300 for the fassttext embedding.

1.2 Model Architecture

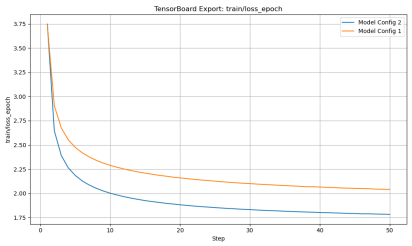
The decoder-only architecture is implemented from scratch using FastText embedding as an initialization point. The script below provides a high-level layout of the architecture:

```
class Decoder(nn.Module):
    def __init__(
        self,
        num_layers,
        num_heads,
        input_dim,
```

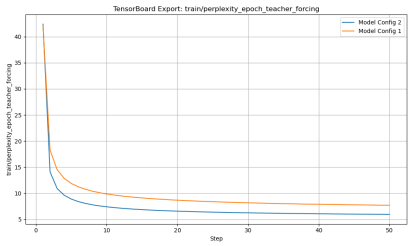
```
    hidden_dim,
    context_size,
    vocab_size,
    vocab,
):
    super(Decoder, self).__init__()
    self.num_layers = num_layers
    self.num_heads = num_heads
    self.input_dim = input_dim
    self.hidden_dim = hidden_dim
    self.context_size = context_size
    self.vocab_size = vocab_size
    self.vocab = vocab
    self.fasttext_embedding = nn.Embedding(self.vocab_size,
    ↪  self.input_dim)
    self.pos_embed = PositionEmbedding(self.context_size,
    ↪  self.input_dim)
    self.special_token_embedding = nn.Embedding(4, self.input_dim)
    self.fasttext_embedding.weight = nn.Parameter(
        torch.load("fasttext_embedding.pt"), requires_grad=False
    )
    print(
        "Embedding weight shape: ",
        self.fasttext_embedding.weight.shape,
        " and sum: ",
        self.fasttext_embedding.weight.sum(),
    )
    self.transformer_layers = nn.ModuleList(
        [
            TransformerBlock(self.num_heads, self.input_dim,
            ↪  self.hidden_dim)
            for _ in range(self.num_layers)
        ]
    )
    self.ln = LayerNorm(self.input_dim)
    self.linear = nn.Linear(self.input_dim, self.vocab_size)

def forward(self, inputs):
    input_ids = inputs["input_ids"]
    attn_mask = inputs["attention_mask"]

    embedded_output = self.fasttext_embedding(input_ids)
    special_token_embedded_output =
    ↪  self.special_token_embedding(torch.clamp(input_ids, max=3))
    mask = (input_ids < 4)
    embedded_output[mask] = special_token_embedded_output[mask]
    pos_embedded_output = self.pos_embed(embedded_output)
    mhsa_output = pos_embedded_output
    for layer in self.transformer_layers:
        mhsa_output = layer(mhsa_output, attn_mask)
    output = self.linear(self.ln(mhsa_output))
```

```
        return output
```

1.3 Training

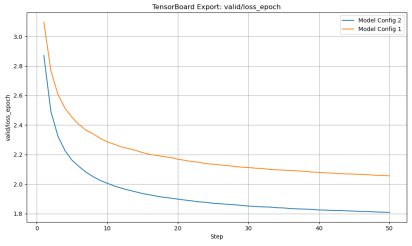


(a) Train loss

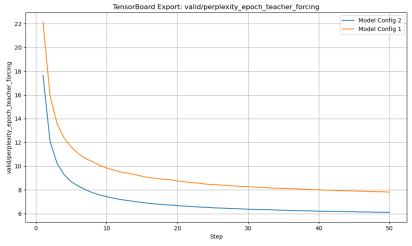


(b) Train perplexity

Figure 1: Loss and Perplexity curves visualisation for Model Configuration 1 for training



(a) Validation loss



(b) Validation perplexity

Figure 2: Loss and Perplexity curves visualisation for Model Configuration 1 for validation

1.4 Inference

1.4.1 Model configuration 1

1. context_size: 64
2. num_layers: 3
3. num_heads: 8
4. input_dim: 300
5. hidden_dim: 64

Performance

1. Total time: 5hr 35min 13sec

2. Last training loss: 2.041
3. Last validation loss: 2.055
4. Last training perplexity: 7.701
5. Last validation perplexity: 5.944
6. Bleu 1: 0.21
7. Bleu 2: 0.15
8. Validation Perplexity: 7.80
9. Decoding: greedy
10. Num of samples: 50

1.4.2 Model configuration 2

1. context_size: 64
2. num_layers: 6
3. num_heads: 16
4. input_dim: 300
5. hidden_dim: 512

Performance

1. Total time: 9hr 15min 34sec
2. Last training loss: 1.782
3. Last validation loss: 1.808
4. Last training perplexity: 7.81
5. Last validation perplexity:6.098
6. Bleu 1: 0.27
7. Bleu 2: 0.17
8. Validation Perplexity: 6.09
9. Decoding: greedy

2 Training and Inference Enhancements

2.1 Beam Search Decoding

Beam search is implemented in the generate() function. The results below highlight the benefits of using beam widths of 5 and 10.

Beam width 5

1. Time taken: 24.90s
2. Bleu 1: 0.50
3. Bleu 2: 0.37
4. Tokens per second: 42414 tokens/sec
5. Number of samples: 100

Beam width 10

1. Time taken: 25.52s
2. Bleu 1: 0.47
3. Bleu 2: 0.33
4. Tokens per second: 82773 tokens/sec
5. Number of samples: 100

Examples comparison

1. **GT**: spot . spot saw the car and said , " wow , kitty , your car is so bright and clean ! " kitty smiled and replied , " thank you , spot . i polish it every day . " after playing with the car , kitty and spot felt thirsty . they found a small pond with clear water . they
   BW 5: spot . spot saw the big red ball in the park . spot wanted to play with the ball . he ran to the ball and kicked it . spot ran after it , but it was too fast . spot was sad . spot 's friend , a dog named max , came to play . max saw spot and wanted to
   BW 10: spot . spot saw the big red ball in the park . spot wanted to play with the ball , but it was too high . he asked his mom , " can i play with the ball ? " his mom said , " no , spot . you ca n't play with it . it 's not safe . " spot

2. **GT**: once upon a time , a big forest , there lived a rhinoceros named roxy . roxy loved to climb . she climbed trees , rocks , and hills . one day , roxy found an icy hill . she had never seen anything like it before . it was shiny and cold , and she wanted to climb it . roxy tried
BW 5: once upon a time , in a small town , there was a little girl named lily . lily loved to play outside in the sun . one day , she saw a big , red ball in the park . she wanted to play with it , but she was too small to reach it . lily asked her mom , "
BW 10: once upon a time , in a small town , there was a little boy named tim . tim loved to play with his toy cars . he had a big red car that he liked to zoom around . one day , tim 's friend , sam , came to play with him . sam saw the car and said , "

3. **GT**: one day , a young named tim found a dull , round rock . he picked it up and looked at it . he thought it was not very fun , but he took it with him to the park . at the park , tim saw a girl named sue . she had a big , red ball . " want to
BW 5: one day , a young boy named tim went to the park with his mom . he saw a big tree and wanted to climb it . tim was scared , but he was brave . he wanted to see what was on the other side . tim 's mom said , " do n't worry , tim . i will help you
BW 10: one day , a young dog named spot went for a walk . he saw a big tree with a hole in it . spot was curious and wanted to find out what was inside . he walked up to the tree and looked inside . he saw a big tree with a hole in it . spot was curious and wanted to

4. **GT**: one day , a little named tim went to play with his friend , sam . they wanted to play a game with a ball . the game was to see who could get the best score . tim threw the ball and sam tried to catch it . sam missed the ball and it went far away . sam was angry .
BW 5: one day , a little dog named spot went for a walk . he saw a big tree with a hole in it . spot wanted to see what was inside . he went to the tree and looked inside . he saw a big tree with a hole in it . spot was curious . he wanted to see what was inside
BW 10: one day , a little cat named tom went to the park . he saw a big tree with a hole in it . tom wanted to see what was inside . he went to the tree and looked inside . he saw a big tree with a hole in it . tom was curious and wanted to see what was inside .

5. **GT**: one day , a girl lily and a boy named tom were playing in their room . they liked to watch videos on their big tv . their favorite video was about a big , impressive dog that could do many tricks . lily began to tease tom , saying , " i bet you ca n't do tricks like the big dog
BW 5: ne day , a girl and her mom went to the park . the girl was so excited . she saw a big slide and wanted to go down it . she asked her mom , " can i go on the slide ? " her mom said , " yes , but be careful . " the girl was very excited .
BW 10: one day , a girl and her mom went to the park . it was a sunny day and the sun was shining . the sun was shining and the birds were singing . the girl was so happy . she saw a big tree with a hole . the tree was very tall and had many leaves . the girl wanted to

2.2 KV Caching

KV caching is implemented in the self-attention mechanism. The results below illustrate the difference in using the generate() function with and without KV caching.

Without KV Caching

1. Number of samples: 20
2. Time taken: 4.871s
3. Tokens per second: 8494 tokens/s
4. Bleu 1: 0.34
5. Bleu 2: 0.20

With KV Caching

1. Number of samples: 20
2. Time taken: 4.544s
3. Tokens per second: 9466 tokens/s
4. Bleu 1: 0.34
5. Bleu 2: 0.20

2.3 Gradient Accumulation

Gradient accumulation is conducted during the model's training. Results are obtained with accumulation steps of 2, 4, and 8 compared to the baseline configuration.

Baseline

1. Total time: 5hr 35min 13sec
2. Last training loss: 2.041
3. Last training perplexity: 7.701
4. Last validation loss: 2.055
5. Last validation perplexity: 7.81

Accumulation steps: 2

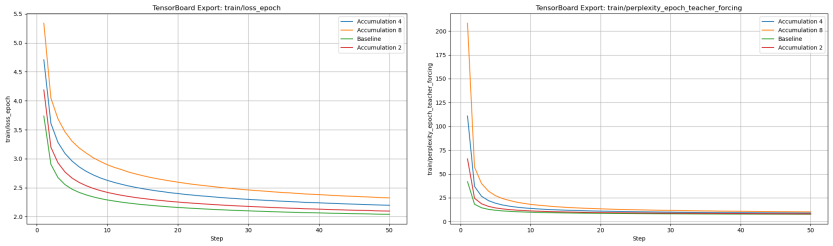
1. Total time: 5hr 31min 29sec
2. Last training loss: 2.097
3. Last training perplexity: 8.138
4. Last validation loss: 2.111
5. Last validation perplexity: 8.259

Accumulation steps: 4

1. Total time: 5hr 31min 4sec
2. Last training loss: 2.197
3. Last training perplexity: 8.997
4. Last validation loss: 2.219
5. Last validation perplexity: 9.196

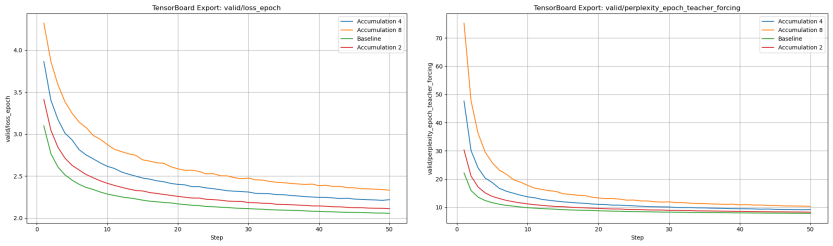
Accumulation steps: 8

1. Total time: 5hr 32min 54sec
2. Last training loss: 2.324
3. Last training perplexity: 10.22
4. Last validation loss: 2.332
5. Last validation perplexity: 10.3



(a) Train loss (b) Train perplexity

Figure 3: Loss and Perplexity curves visualisation for baseline and gradient accumulation for training



(a) Validation loss (b) Validation perplexity

Figure 4: Loss and Perplexity curves visualisation for baseline and gradient accumulation for validation

2.4 Gradient Checkpointing

Gradient checkpointing is utilized during the training process. Below are the results from the gradient checkpointing experiment.
Gradient Checkpointing

1. Total time: 6hr 55min 15sec
2. Last training loss: 2.042
3. Last training perplexity: 7.706
4. Last validation loss: 2.059
5. Last validation perplexity: 7.839

Memory used without gradient checkpointing: 21.1GB
Memory used with gradient checkpointing: 16.4GB