# Assignment Week 9

Student Name: Patrick Farmer Student Number: 20501828

Date: 14-11-2024

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

## I(a)

The zip file was download and extracted. I then ran gpt.py but this took an unreasonable length of time to run as it was running on the CPU. Since I have an AMD GPU I had to instal ROCm to run the code on the GPU. I then ran the code again and it ran much faster.
Now that the environment was setup I changed the code to take the file path as a command line argument and ran the code pointing at the input_childSpeech_trainingSet.txt data set. A print message was also added to print the vocabulary size of the dataset wich was 40. This was run for the two other datasets as well. The childSpeechTest dataset also had a vocabulary size of 40 while the shakespeare dataset had a vocabulary size of 65.
The child speech dataset has a very large file which has sentences in childlike English. Each sentence is independent of each other and no story is being told. The sentences are also very small and simple. The phrases used would not be very representative of the English language as a whole, there is a lot of "mummy", "daddy", "book", "bed" etc.

## I(b)

The parameters of the model that I changed were: - Block size: 256 -> 64 - Max Iterations: 5000 -> 3000 - Evaluation Interval: 500 -> 300 - Learning Rate: 3e-4 -> 2e-4 - Embedding Size: 384 -> 128 - Attention Heads: 6 -> 4 - Transformer Layers: 6 -> 4 - Dropout Rate: 0.2 -> 0.3

The parameter count of this model was 0.80772 M parameters
Block size is the maximum context length. Which is the amount of previous tokens that the model will consider when predicting the next token.

- The block size was reduced 4 times to 64. This was done to reduce the parameter count of the model, I chose to reduce this significantly as looking at the data set I saw that each line was quite short and independent of the next and previous lines so a large context size is unnecessary. Decreasing the block size will also decrease the number of computations in self-attention.
  Max Iterations is the maximum number of iterations that the model will train for.
- The max iterations was reduced to 3000 as it was found that the simpler model did not need as many iterations to train and reducing this would reduce the training time and level of overfitting.
  Evaluation Interval is the number of iterations between evaluations against the validation set.
- The evaluation interval was reduced to 300 to match the decrease in the max iterations and have the same count of evaluations.
  Learning Rate is the rate at which the optimiser will update the weights of the model.
- The learning rate was reduced to 2e-4 to stabilise training as the model was simplified and a smaller learning rate will make the model converge more smoothly and less likely to overfit.
  Embedding Size is the size of the embedding layer, which is the layer that converts the input tokens into a dense vector.
- The embedding size was reduced to 128 to reduce the parameter count of the model. This was done as the vocabulary size of the data set was quite small and the embedding size did not need to be as large. The parameter size is significantly reduced with this reduction but makes little difference to the model performance.
  Attention Heads is the number of heads in the multi-head attention layer. This is the number of different attention mechanisms that the model will use.
- The attention heads was reduced to 4 to reduce the parameter count of the model. Due to the short sentences in the dataset the structure cannot get too complex and does not require as many attention heads.
  Transformer Layers is the number of transformer layers in the model.

- The transformer layers was reduced to 4 to reduce the parameter count of the model. For the simple dataset and smaller model decreasing the models chances of creating overly complex features which minimises overfitting and improves generalisation.
  Dropout Rate is the rate of the dropout layer. This is the percentage (in decimal form) of the units that will be dropped during training.
- The dropout rate was increased to 0.3 to reduce the level of overfitting. This was done as the model was simplified and the dropout rate was increased to reduce the level of overfitting.

## I(c)

Two other sets of hyperparameters that I tried were as follows:

**Hyperparameters set 2**

- Block size: 128
- Max Iterations: 3000
- Evaluation Interval: 300
- Learning Rate: 2e-4
- Embedding Size: 128
- Attention Heads: 3
- Transformer Layers: 3
- Dropout Rate: 0.1

**Hyperparameters set 3**

- Block size: 256
- Max Iterations: 3000
- Evaluation Interval: 300
- Learning Rate: 2e-4
- Embedding Size: 172
- Attention Heads: 2
- Transformer Layers: 2
- Dropout Rate: 0.1

The the three models were trained on the childSpeechTraining dataset and the parameter count and loss function for each model was as follows:

**Model 1**

```
0.80772 M parameters
Vocabulary size: 40
step 0: train loss 3.6883, val loss 3.6886
step 300: train loss 0.4891, val loss 0.4938
step 600: train loss 0.3920, val loss 0.3949
step 900: train loss 0.3792, val loss 0.3822
step 1200: train loss 0.3767, val loss 0.3788
step 1500: train loss 0.3744, val loss 0.3785
step 1800: train loss 0.3707, val loss 0.3745
step 2100: train loss 0.3722, val loss 0.3760
step 2400: train loss 0.3711, val loss 0.3748
step 2700: train loss 0.3698, val loss 0.3716
step 2999: train loss 0.3697, val loss 0.3719
```

**Model 2**

```
0.615592 M parameters
Vocabulary size: 40
step 0: train loss 3.7671, val loss 3.7663
step 300: train loss 0.4805, val loss 0.4838
step 600: train loss 0.3643, val loss 0.3675
step 900: train loss 0.3560, val loss 0.3586
step 1200: train loss 0.3510, val loss 0.3547
step 1500: train loss 0.3454, val loss 0.3500
step 1800: train loss 0.3455, val loss 0.3493
step 2100: train loss 0.3433, val loss 0.3488
step 2400: train loss 0.3437, val loss 0.3472
step 2700: train loss 0.3440, val loss 0.3494
step 2999: train loss 0.3425, val loss 0.3468
```

**Model 3**

```
0.769912 M parameters
Vocabulary size: 40
step 0: train loss 3.7828, val loss 3.7833
step 500: train loss 0.4572, val loss 0.4624
step 1000: train loss 0.3463, val loss 0.3504
step 1500: train loss 0.3357, val loss 0.3400
step 2000: train loss 0.3341, val loss 0.3395
step 2500: train loss 0.3319, val loss 0.3378
step 3000: train loss 0.3299, val loss 0.3370
step 3500: train loss 0.3290, val loss 0.3350
step 4000: train loss 0.3281, val loss 0.3362
step 4500: train loss 0.3260, val loss 0.3373
step 4999: train loss 0.3247, val loss 0.3362
```

The three models had very few differences between them. The only observable difference in the output of the models is that model 1 has more typos than the other two. The sentence structure, grammar and content is very similar between the output of the three models though. The loss function of the three models is also very similar with model 3 having the best and model 1 having the worst. This backs up the observation that model 1 has more typos than the other two models. Model 2 has the lowest parameter counts and almost as good a loss function as model 3. For a scenario where the model needs to be as small as possible model 2 would be the best choice.
The three models do not overfit significantly, the loss of the validation set is very close to the loss of the training set for all three models. The models were tested with a reduced level of regularisation to see if the models would perform better but there was no improvement in the loss function of the models and the models were reverted to the above hyperparameters.

## I(d)

Bias terms are parameters added to the output of a neuron before applying an activation function. They are used to shift the output of the neuron by a constant value. This is useful when the data is not centered around zero. The effect of this in a transformer model is that contextual understanding will be improved and the end of a sentence is more likely to fit the start of the sentence. Language is also non-linear so the bias term enables the LLM to learn language better.
In this case the bias term being set to true or false made nearly no difference this is likely due to the

fact that the data is based on sentences from children and do not have a lot of context to them or make very much sense.

## I(e)

Skip layers are connections between layers that skip the layer(s) directly in front of them. In a model they can improve both stability and performance. In the context of GPTs they help prevent the vanishing gradient problem which is where the gradients during backpropagation become so small that they do not update and the model stops learning, it does this because the weight update is based on the previous layer but also the residual connection. The more layers that a model has the more crucial the skip layers are.
Skip layers are not overly necessary in the models used during this project as the parameter count is capped at 1M the skip layers are not as crucial as the model is not very deep. The residual connections will also improve generalisation of the model as the model will not create overly complex features that are not useful (overfitting).

## II(a)

The code was setup so that it could select training data, parameter set and output model path from command line so I created a simple bash script that would loop through the parameter sets and save save the best model to a variable.
The python script was then update with a new CLI argument –no-train which would load the model from the path instead of training and then get the loss of the model on the test set.
The bash script then saw further updates to load the best model and get the loss of the two other datasets that it had not been trained on.
For this section the input_childSpeech_testSet.txt dataset was used to test the models. The loss of the model on this dataset was near identical to the loss of the model on the validation set during training. This is because the data set is the same style (children's speech) as the training set. The loss of the best model on the validation set is shown above as 0.3362. Then when run on the test dataset the loss was 0.3422. This is a very small increase in loss and shows that the model generalises well to unseen data of the same style.
When compared to the baseline model this model does massively better, the baseline model simply predicts the most common token in the training set for every token. This results in a loss of 4781820.0000 which is significantly worse than the best model. The massive difference between the two models show that the model is performing well because it has managed to capture the structure of the data and not just because the data is simple.

## II(b)

The above mentioned bash script would also test the loss of the best child_speech model on the shakespeare dataset which is a dataset of shakespearean text. The major differences between this dataset and the training set is: * The vocabulary size is 65 * The sentence structure is much more complex * The words used are more complex * Old English is used * Lines are not just independent of each other but are part of a larger story

Based on this we can expect to see a much larger loss than was seen for the child_speech_test_set. The code first had to be updated so that the model could handle any vocabulary size instead of being set to 45. After this the model was run against the shakespeare dataset and the loss was 4.2054. This compared to the performance of the child speech dataset shows that the model has learned the structure of the child speech dataset specifically. However, when we compare it to the baseline classifier with a loss of 568631680.0000 we can see that the model is still performing a lot better than baseline which is because it has learned the structure of English language even though it is one specific style.
In practice the model would need to be fine-tuned on the shakespeare dataset so that it could learn the

structure of the shakespearean language. Another cheaper option that takes advantage of the already trained data is to use transfer learning. This is where the trained model is used as a starting point and then some of the layers are unfrozen and the model is trained on the new data. This allows the earlier layers to remember the base structure of words and the later layers to learn the new structure of the shakespearean language.

## Appendices

### I(a)

```
input_file = args.path

with open(input_file, 'r', encoding='utf-8') as f:
    text = f.read()
```

### I(b)

```
if args.parameters == 1:
    batch_size = 64
    block_size = 64
    max_iters = 3000
    eval_interval = 300
    learning_rate = 2e-4
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    eval_iters = 200
    n_embd = 128
    n_head = 4
    n_layer = 4
    dropout = 0.1
```

### I(c)

```
# Hyperparameters set 2
if args.parameters == 2:
    batch_size = 64
    block_size = 128
    max_iters = 3000
    eval_interval = 300
    learning_rate = 2e-4
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    eval_iters = 200
    n_embd = 128
    n_head = 3
    n_layer = 3
    dropout = 0.1

# Hyperparameters set 3
if args.parameters == 3:
    batch_size = 64
    block_size = 256
    max_iters = 3000
    eval_interval = 300
    learning_rate = 2e-4
```

```
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    eval_iters = 500
    n_embd = 172
    n_head = 2
    n_layer = 2
    dropout = 0.1
```

```
for iter in range(max_iters):
        # every once in a while evaluate the loss on train and val sets
        if iter % eval_interval == 0 or iter == max_iters - 1:
            losses = estimate_loss()
            # Append losses during training
            with open(log_file, 'a') as f:
                f.write(f"step {iter}: train loss {losses['train']:.4f}, val loss {losses['val']:.4f}
```

### I(d)

```
nn.Linear(n_embd, 4 * n_embd, bias=False),
nn.ReLU(),
nn.Linear(4 * n_embd, n_embd, bias=False),
nn.Dropout(dropout),
```

```
nn.Linear(n_embd, 4 * n_embd, bias=True),
nn.ReLU(),
nn.Linear(4 * n_embd, n_embd, bias=True),
nn.Dropout(dropout),
```

### II(a)

```
#!/bin/bash

for i in {1..3}; do
    python gpt.py --path input_childSpeech_trainingSet.txt --parameters $i --model-path Models/model_
done

LowestLost=100000000
bestIndex=-1
for i in {1..3}; do
    loss=$(tail -n 1 Logs/results_input_childSpeech_trainingSet_params$i.log | awk '{print $NF}')
    if (( $(awk "BEGIN {print ($loss < $LowestLost)}") )); then
        LowestLost=$loss
        bestIndex=$i
        BestModel=Models/model_childSpeech_params$i
    fi
done

python gpt.py --path input_childSpeech_testSet.txt --parameters $bestIndex --model-path $BestModel --
python gpt.py --path input_shakespeare.txt --parameters $bestIndex --model-path $BestModel --no-train

echo "Testing complex model"
python gpt.py --path input_childSpeech_trainingSet.txt --parameters 4 --model-path Models/model_child
```