Question 1.2.1

Which class does LanguageModelingDataset inherit from?

It inheirits from torch.utils.data.Dataset

Question 1.2.2

What does the function lm collate fn do? Explain the structure of the data that results when it is called.

It pads tokens to sentences that are shorter than the longest sentence in the batch. After padding the batch of samples will be composed of sentences of the same length on the specified device.

Question 1.2.3

Looking the notebook block [6], (with comment "Print out an example of the data") what does this tell you about the relationship between the input (X) and output (Y) that is sent the model for training?

Y is the latter part of X which is used as the target for the prediction task.

Question 1.2.4

Given one such X,Y pair, how many different training examples does it produce?

3 since there's 4 words in the sentence.

Question 1.2.5

In the generate function in the file model.py what is the default method for how the generated word is chosen - i.e. based on the model output probabilities?

The default method is to not sample and take the most likely element.

Question 1.2.6

What are the two kinds of heads that model.py can put on to the transformer model? Show (reproduce) all the lines of code that implement this functionality and indicate which method(s) they come from.

Language model heads and classifier heads.

```
self.lm_head = nn.Linear(config.n_embd, config.vocab_size,
bias=False)
self.classifier_head = nn.Linear(config.n_embd,
config.n_classification_class, bias=True)



if not finetune_classify:
    # LM forward procedure
    logits = self.lm_head(x)
else:
    # Finetune classify procedure
    logits = self.classifier_head(x[:, -1, :])
```

They are from the __init__ and forward method from model.py

Question 1.2.7

How are the word embeddings initialized prior to training?

They are either copied from a pretrained model or randomly initialized depending on whether the type or the parameters are given.

Question 1.2.8

What is the name of the object that contains the positional embeddings?

pos_emb

Question 1.2.9

How are the positional embeddings initialized prior to training?

They are randomly initialized.

Question 1.2.10

Which module and method implement the skip connections in the transformer block? Give the line(s) of code that implement this code.

```
def forward(self, x):
    x = x + self.mlpf(self.ln_2(x))
```

It'd defined in the forward method.

Question 2.1

Report the value of the loss.

```
iter_dt 0.00ms; iter 0: train loss 10.81249
iter_dt 8.68ms; iter 100: train loss 5.96995
iter_dt 9.21ms; iter 200: train loss 2.49559
iter_dt 10.11ms; iter 300: train loss 1.49280
iter_dt 15.74ms; iter 400: train loss 0.83902
iter_dt 8.57ms; iter 500: train loss 0.78918
iter_dt 9.00ms; iter 600: train loss 0.83952
iter_dt 9.00ms; iter 700: train loss 0.70429
iter_dt 9.01ms; iter 800: train loss 0.64494
iter_dt 9.00ms; iter 900: train loss 0.59071
iter_dt 9.44ms; iter 1000: train loss 0.56029
iter_dt 7.98ms; iter 1100: train loss 0.76987
iter_dt 8.03ms; iter 1200: train loss 0.58646
iter_dt 9.02ms; iter 1300: train loss 0.61791
iter_dt 9.00ms; iter 1400: train loss 0.66156
iter_dt 7.98ms; iter 1500: train loss 0.68874
iter_dt 8.00ms; iter 1600: train loss 0.69681
iter_dt 9.00ms; iter 1700: train loss 0.62078
iter_dt 8.03ms; iter 1800: train loss 0.58298
iter_dt 9.27ms; iter 1900: train loss 0.59302
iter_dt 7.99ms; iter 2000: train loss 0.59085
iter_dt 8.04ms; iter 2100: train loss 0.60756
iter_dt 9.11ms; iter 2200: train loss 0.64772
iter_dt 8.00ms; iter 2300: train loss 0.57708
iter_dt 8.00ms; iter 2400: train loss 0.62975
iter_dt 9.02ms; iter 2500: train loss 0.65382
iter_dt 7.99ms; iter 2600: train loss 0.64882
iter_dt 28.44ms; iter 2700: train loss 0.76117
iter_dt 22.10ms; iter 2800: train loss 0.63148
iter_dt 25.01ms; iter 2900: train loss 0.68538
```

Question 2.2

What is the output for each? Why does the the latter parts of the generation not make sense?

'He and I hold the dog.. cat. cat and dog'

'She rubs a cat and dog. cat. cat. cat'

It doesn’t make sense as the model was trained on the small corpus where there’s no latter part.

Question 2.3

Show the output along with these probabilities for the two examples, and then one of your own choosing.

```
He 1.0
 and 1.0
 I 1.0
 can 0.7332900166511536
 hold 0.6262842416763306
 the 0.5019563436508179
 dog 0.6777902245521545
. 0.9997881054878235
 cat 0.8717184662818909
. 0.9997501969337463
 cat 0.7051288485527039
 and 0.7348537445068359
 cat 0.6792247891426086
```

```
She 1.0
 rub 1.0
s 1.0
 a 0.4766572415828705
 dog 0.6089667081832886
 and 0.5519123673439026
 cat 0.9998014569282532
. 0.9991673231124878
 dog 0.6277222633361816
 and 0.5781517028808594
 cat 0.997319757938385
. 0.999198853969574
 cat 0.8431316018104553
```

```
The 1.0
 cat 1.0
 can 1.0
 rub 0.6098027229309082
 the 0.5069823861122131
 dog 0.6066402792930603
. 0.9990702271461487
. 0.9826630353927612
 cat 0.6837152242660522
 and 0.5920320749282837
 cat 0.9993662238121033
 and 0.7476220726966858
 dog 0.9891071915626526
```

Question 2.4

Show the result in a table that gives all six words, along with their probabilities, in each column of the table. The number of columns in the table is the total number of generated words. For the first two words generated, explain if the probabilities in the table make sense, given the input corpus.

```
A 1.0
 cat 1.0
 can 1.0
```

6 rows × 10 columns pd.DataFrame

	0	1	2	3	4	5	6	7	8	9
1st	rub 0.52	dog 0.65	a 0.55	a 0.42	a 0.46	a 0.44	. 0.97	s 1.00	. 0.99	a 0.55
2nd	hold 0.48	cat 0.35	the 0.36	the 0.29	the 0.29	the 0.31	and 0.03	and 0.00	. 0.01	the 0.45
3rd	can 0.00	can 0.00	and 0.08	and 0.29	and 0.25	and 0.25	. 0.00	. 0.00	and 0.00	and 0.00
4th	dog 0.00	a 0.00	. 0.00	. 0.00	. 0.00	. 0.00	a 0.00	a 0.00	rub 0.00	holds 0.00
5th	cat 0.00	hold 0.00	s 0.00	cat 0.00	cat 0.00	holds 0.00	the 0.00	cat 0.00	dog 0.00	hold 0.00
6th	a 0.00	rub 0.00	holds 0.00	holds 0.00	holds 0.00	cat 0.00	rub 0.00	dog 0.00	holds 0.00	rub 0.00

Given the input corpus it makes sense since the only possible words after can are rub or hold and they are equally likely. However, dog or cat following rub does not make sense as a determiner is missing in between.

Question 3.2

Report the examples you used and the generation results, and comment on the quality of the sentences.

```python
# Example showing how the reloaded model still works
encoded_prompt = train_dataset.tokenizer("Inflation is caused by the").to(trainer.device)
generated_sequence,prob_seq = trainer.model.generate(encoded_prompt, trainer.device, temperature=0.6, max_new_tokens=10)
i=0
import pandas as pd
df=pd.DataFrame({'1st' : [],'2nd' : [],'3rd' : [],'4th' : [],'5th' : [],'6th' : []})
while i<len(prob_seq[0]):...

df.transpose()
```

Executed at 2023.10.17 23:46:58 in 84ms

```
 In flation  is  caused  by  the  death
  � ,
  F  date  were  to  upon
```

6 rows × 10 columns pd.DataFrame

	0	1	2	3	4	5	6	7	8
1st	death 0.35	\n 0.97	� 0.94	, 0.45	\n 0.96	F 0.63	date 1.00	were 1.00	to 0.92
2nd	ELL 0.15	have 0.02	� 0.05	. 0.42	� 0.02	. 0.25	year 0.00	for 0.00	not 0.01
3rd	man 0.11	ism 0.00	� 0.01	and 0.13	the 0.01	ER 0.04	re 0.00	. 0.00	fast 0.01
4th	own 0.06	. 0.00	0.00	to 0.00	date 0.00	TER 0.02	of 0.00	æ 0.00	generally 0.01
5th	REE 0.04	OF 0.00	ue 0.00	� 0.00	his 0.00	; 0.01	had 0.00	ing 0.00	necessarily 0.01

```python
# Example showing how the reloaded model still works
encoded_prompt = train_dataset.tokenizer("The federal government issued").to(trainer.device)
generated_sequence,prob_seq = trainer.model.generate(encoded_prompt, trainer.device, temperature=0.6, max_new_tokens=10)
i=0
import pandas as pd
df=pd.DataFrame({'1st' : [],'2nd' : [],'3rd' : [],'4th' : [],'5th' : [],'6th' : []})
while i<len(prob_seq[0]):...

df.transpose()
```

Executed at 2023.10.18 00:07:15 in 79ms

```
 The  federal  government  issued  in  State  Secretary
  eded  of  the  order  years .
```

6 rows × 10 columns pd.DataFrame

	0	1	2	3	4	5	6	7	8
1st	in 0.82	State 0.37	Secretary 0.52	\n 0.86	eded 0.97	of 1.00	the 0.97	order 0.56	years 0.
2nd	from 0.17	report 0.17	ins 0.33	. 0.04	- 0.03	. 0.00	\n 0.03	were 0.13	and 0.19
3rd	INT 0.00	position 0.11	that 0.04	] 0.03	some 0.00	is 0.00	one 0.00	was 0.05	stars 0.
4th	, 0.00	commission 0.05	officer 0.02	facing 0.02	af 0.00	— 0.00	it 0.00	ne 0.03	clock 0.
5th	of 0.00	States 0.04	of 0.02	� 0.01	with 0.00	being 0.00	but 0.00	\n 0.03	talents
6th	a 0.00	request 0.03	ance 0.01	re 0.00	ual 0.00	; 0.00	an 0.00	hundred 0.02	pieces

```python
# Example showing how the reloaded model still works
encoded_prompt = train_dataset.tokenizer("Spain and Portugal").to(trainer.device)
generated_sequence,prob_seq = trainer.model.generate(encoded_prompt, trainer.device, temperature=0.6, max_new_tokens=10)
i=0
import pandas as pd
df=pd.DataFrame({'1st' : [],'2nd' : [],'3rd' : [],'4th' : [],'5th' : [],'6th' : []})
while i<len(prob_seq[0]):...

df.transpose()
```

Executed at 2023.10.18 00:09:56 in 86ms

```
 Spain  and  Portugal  coins
   coins  on  laws ,  �  vere  of  were
```

6 rows × 10 columns pd.DataFrame

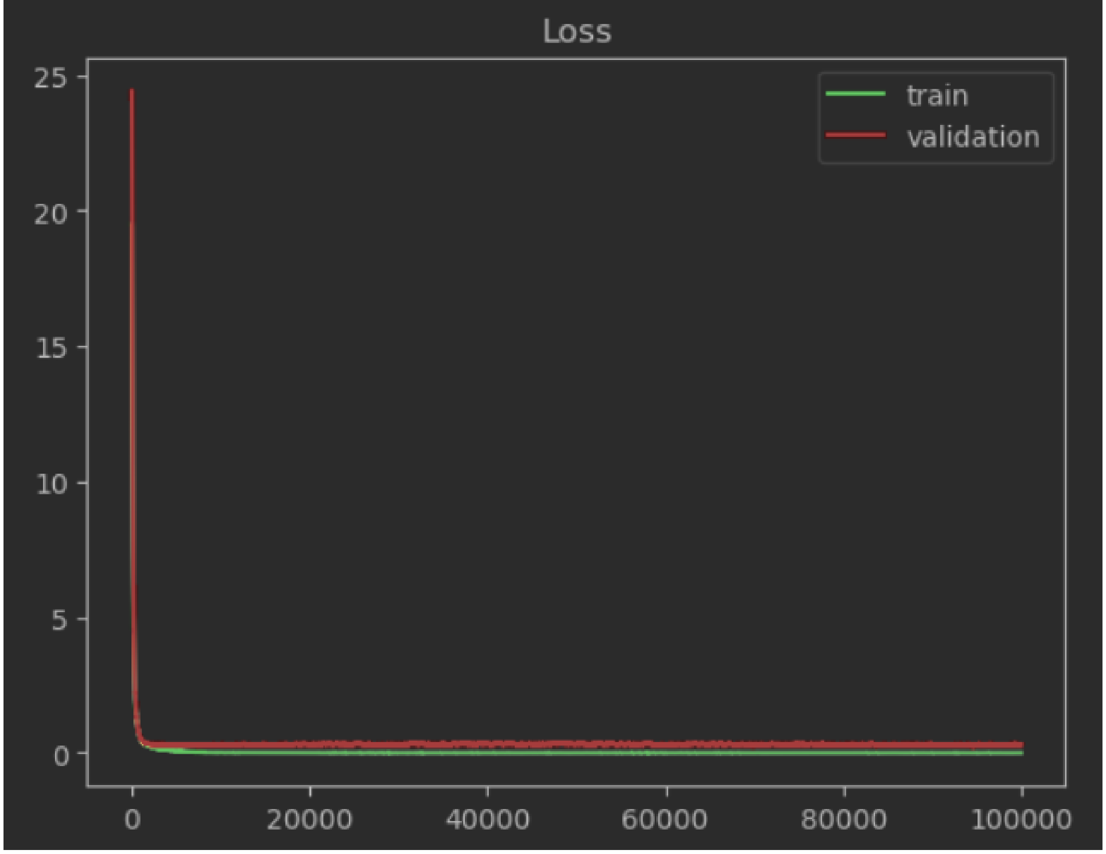
	0	1	2	3	4	5	6	7	8
1st	coins 0.45	\n 1.00	coins 0.43	on 0.93	laws 0.56	, 0.81	� 0.89	vere 0.09	of 1.00
2nd	to 0.11	in 0.00	metal 0.21	put 0.04	s 0.15	a 0.05	D 0.05	ule 0.07	the 0.00
3rd	has 0.11	and 0.00	coin 0.05	ready 0.01	under 0.13	., 0.03	C 0.02	OU 0.06	, 0.00
4th	for 0.08	at 0.00	gold 0.04	in 0.01	bottom 0.02	and 0.02	\n 0.02	Clerk 0.06	was 0.00
5th	a 0.06	, 0.00	metals 0.03	of 0.00	been 0.02	being 0.02	and 0.00	places 0.04	as 0.00
6th	out 0.05	the 0.00	are 0.02	more 0.00	_ 0.01	. 0.02	L 0.00	y 0.04	there 0.00

The results were gibberish, usually only the first word being predicted will make any sense.

Question 3.3

Report the training and validation curves for the fine-tuning, and the accuracy achieved on the validation dataset.

```
Final test accuracy: 0.7041666666666667
[0.00290389]
[0.29625264]
```



Question 4.2

Report the classification accuracy on the validation set. Comment on the performance of this model: is it better than the model you fine-tuned in the previous section?

[9600/9600 52:44, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	2.533500	1.490864	0.758333
2	1.413200	1.494483	0.825000
3	0.885100	1.170032	0.812500
4	0.528300	1.163084	0.854167
5	0.380000	1.290416	0.841667
6	0.147600	1.244972	0.833333
7	0.139200	1.334769	0.850000
8	0.090200	1.393531	0.837500
9	0.038800	1.370420	0.870833
10	0.019400	1.279875	0.879167

```
TrainOutput(global_step=9600, training_loss=0.5977858739693208, metrics={'train_runtime': 3165.1271, 'train_samples_per_second': 3.033, 'train_steps_per_second': 3.033, 'total_flos': 5016897611366400.0, 'train_loss': 0.5977858739693208, 'epoch': 10.0})
```

The training process was much slower and the validation accuracy got much better, but this is expected since the model used (gpt-2) has 100 times more parameters than gpt-2 nano.