Due Date: Monday, 14th November, 11pm ET

Problem 1

Problem 2

Implementing a GPT (Generative-Pretrained Transformer) (27pts) Implementing the attention mechanism (17pts):

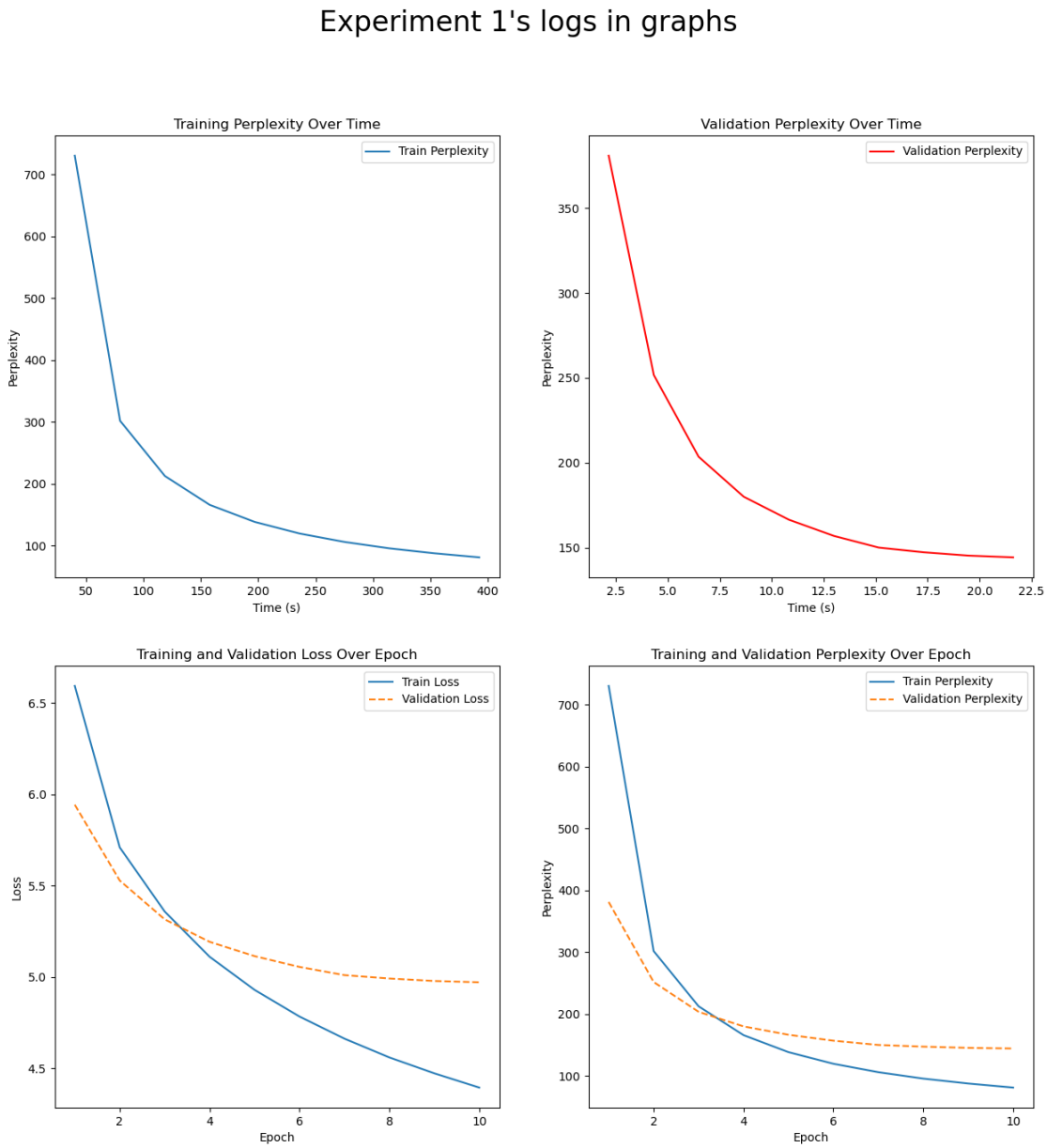
2. The linear layers in my `__init__()` are W_Q, W_K, W_V, and W_Y. Since they all have weight matrices of size ($hidden\_size$, $hidden\_size$) and bias of size ($hidden\_size$), with $hidden\_size = num\_heads * head\_size$.
   As a result, the weights are of size: $hidden\_size * hidden\_size$
   and the biases: $hidden\_size$. Since we have 4 linear layers, the final number of parameters is:

$$4((num\_heads * head\_size)^2 + (num\_heads * head\_size))$$

Problem 3

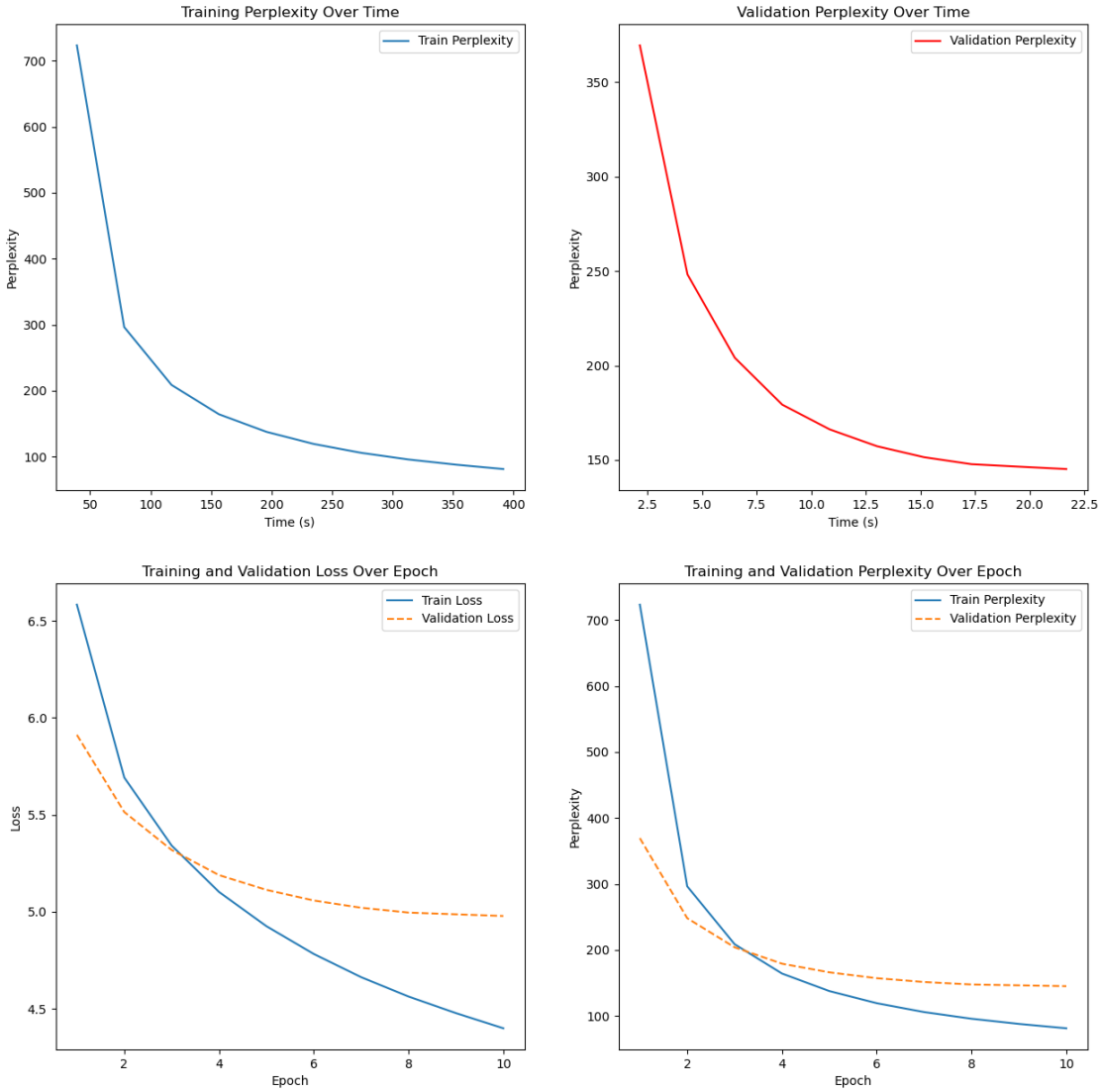
Training language models and model comparison (25pts)

1. Here are the graphs for the 12 experiments:



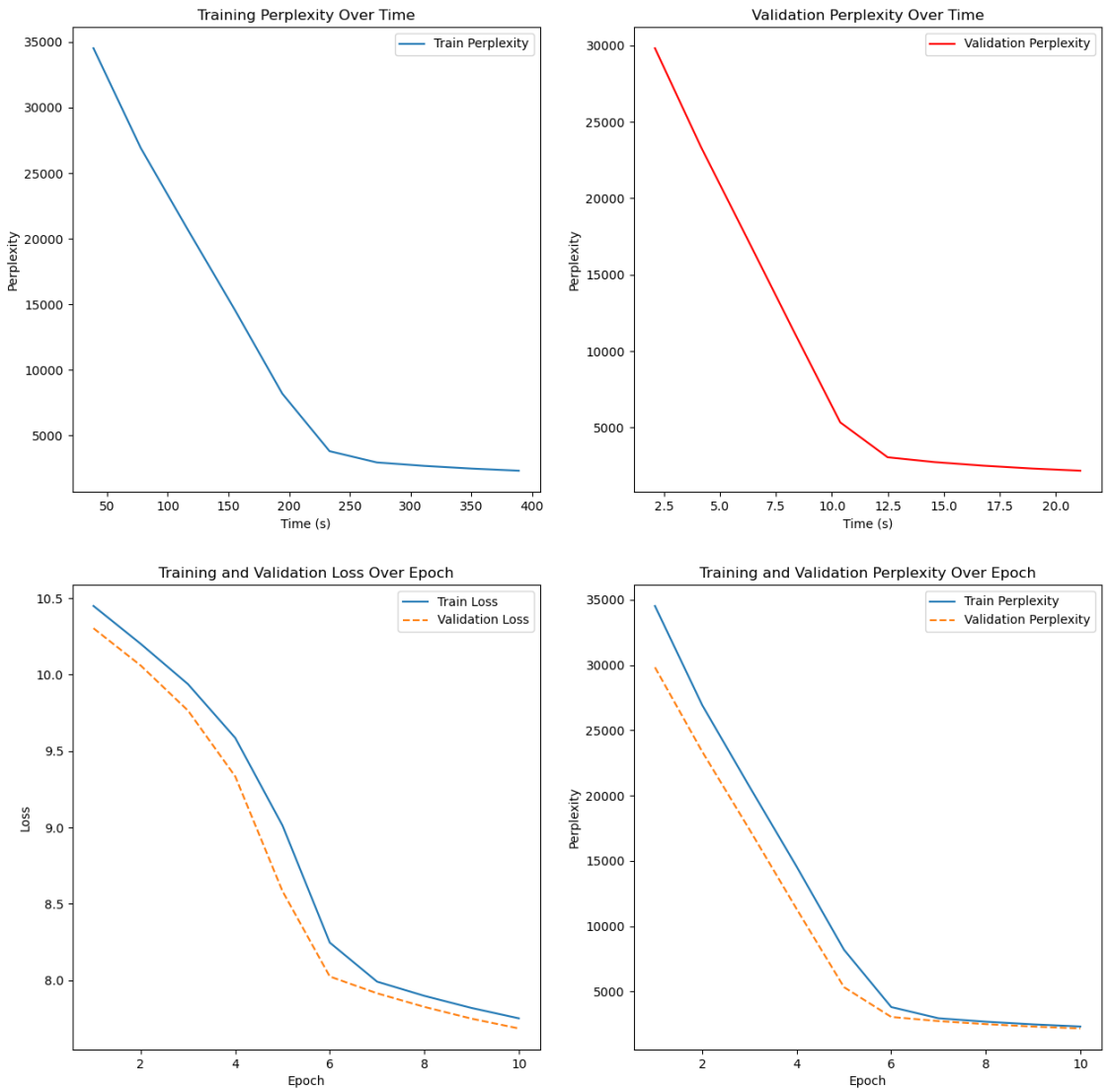
Model: lstm, Layers: 1, Optimizer: adam, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 2's logs in graphs



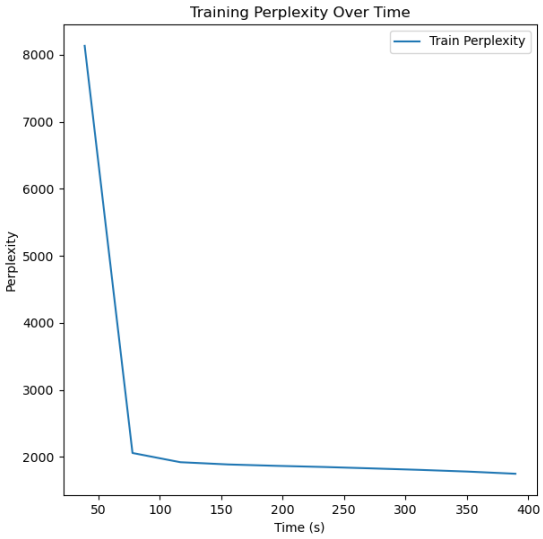
Model: lstm, Layers: 1, Optimizer: adamw, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

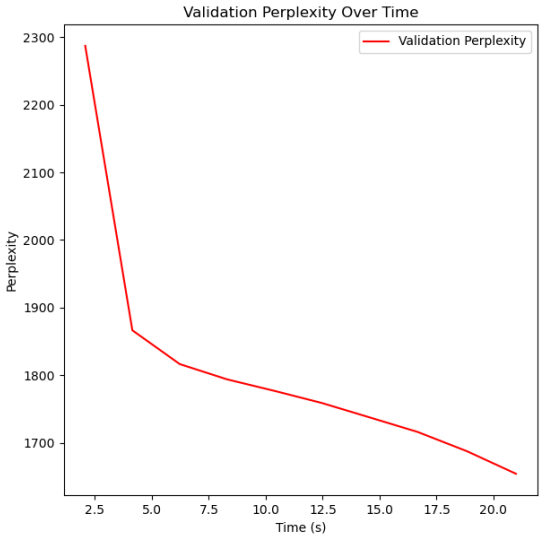
Experiment 3's logs in graphs

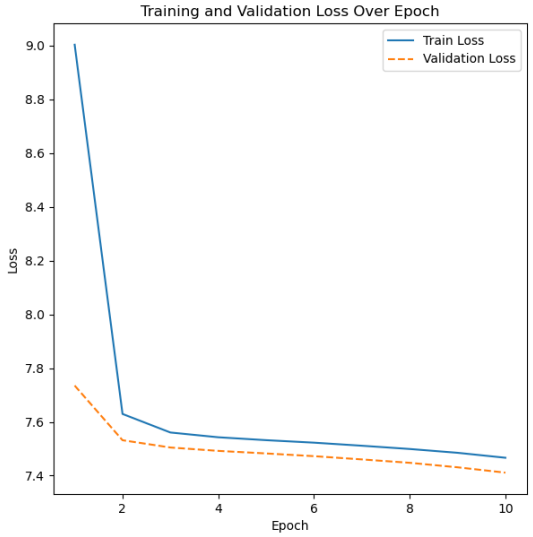


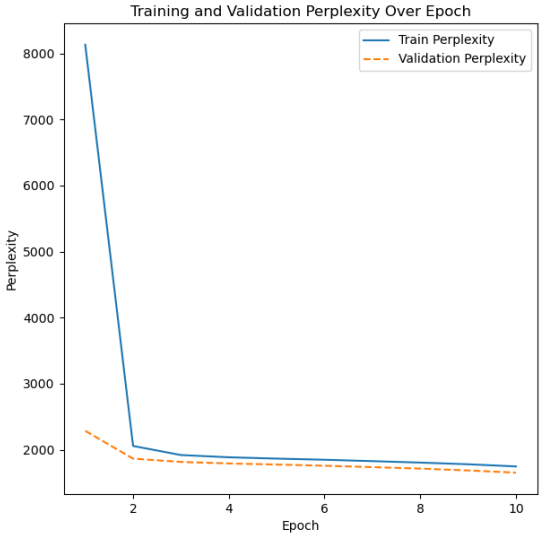
Model: lstm, Layers: 1, Optimizer: sgd, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 4's logs in graphs



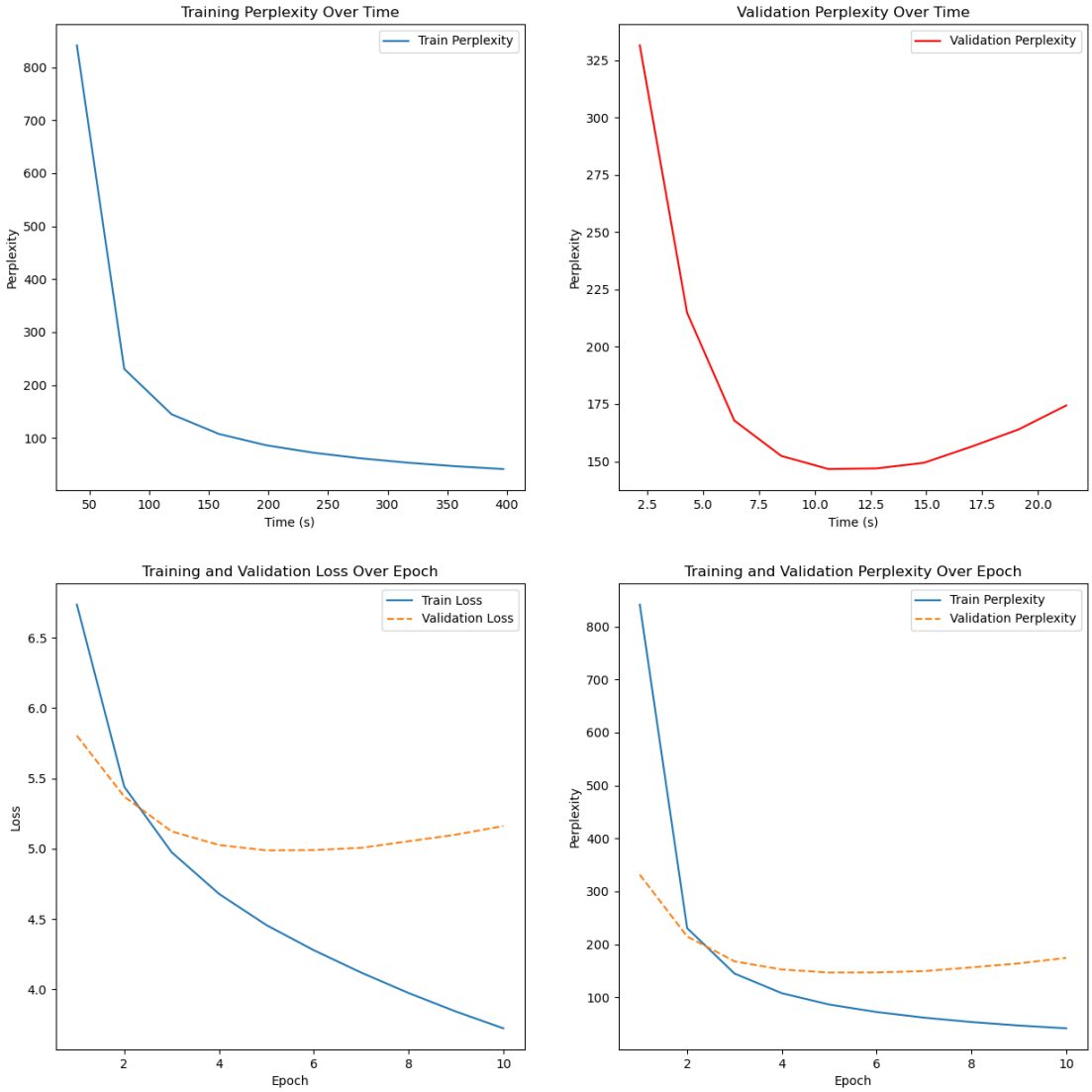






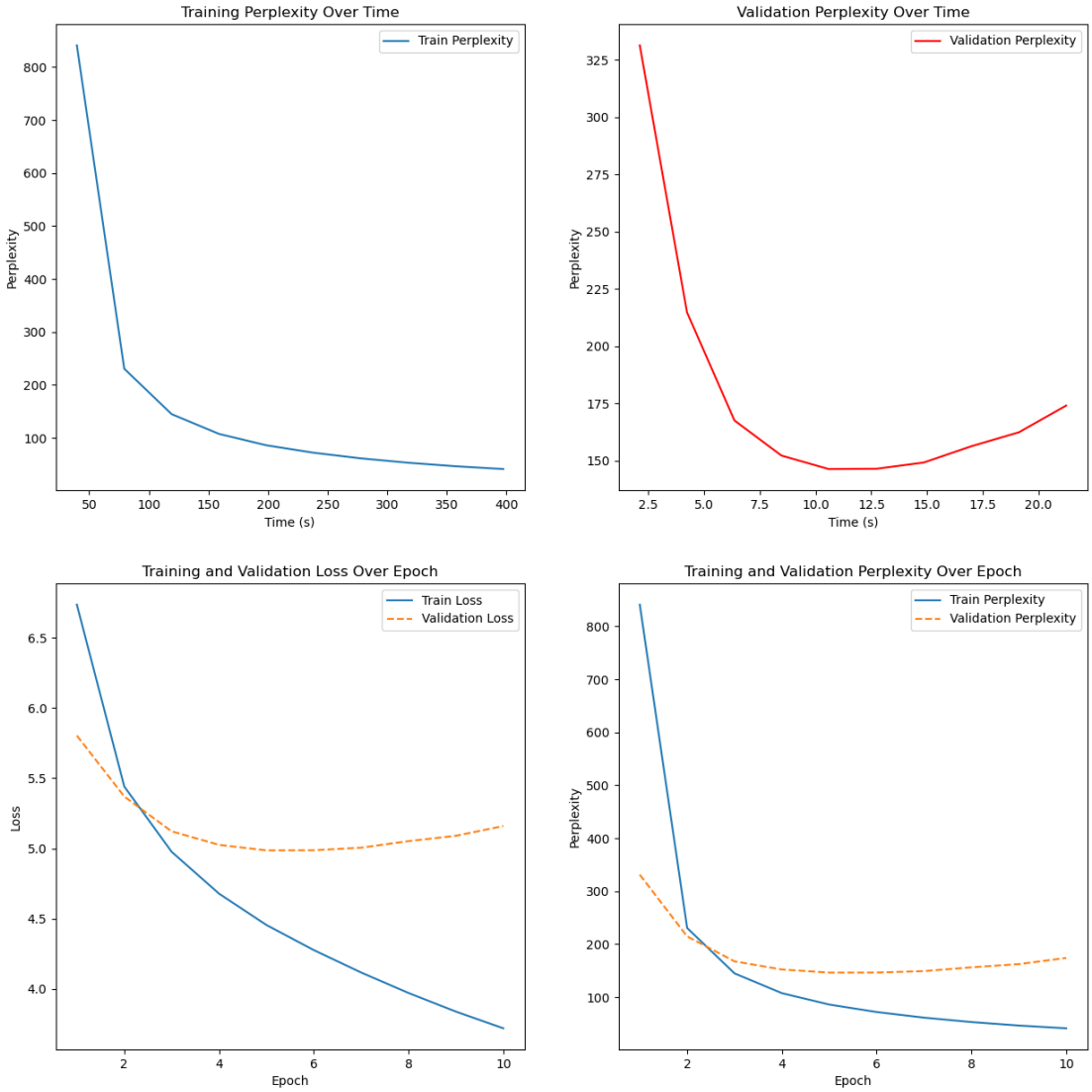
Model: lstm, Layers: 1, Optimizer: momentum, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 5's logs in graphs



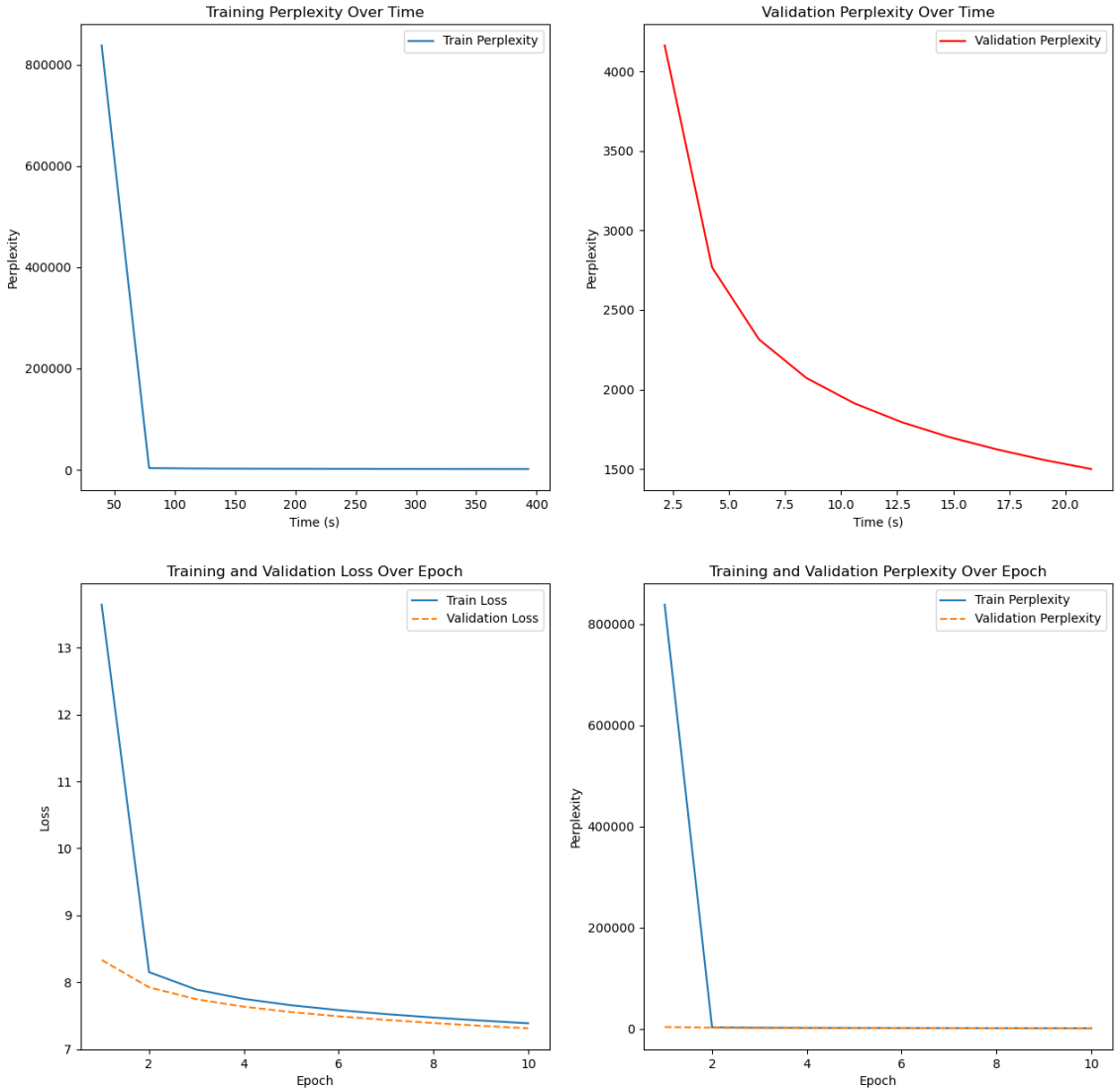
Model: gpt1, Layers: 1, Optimizer: adam, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 6's logs in graphs



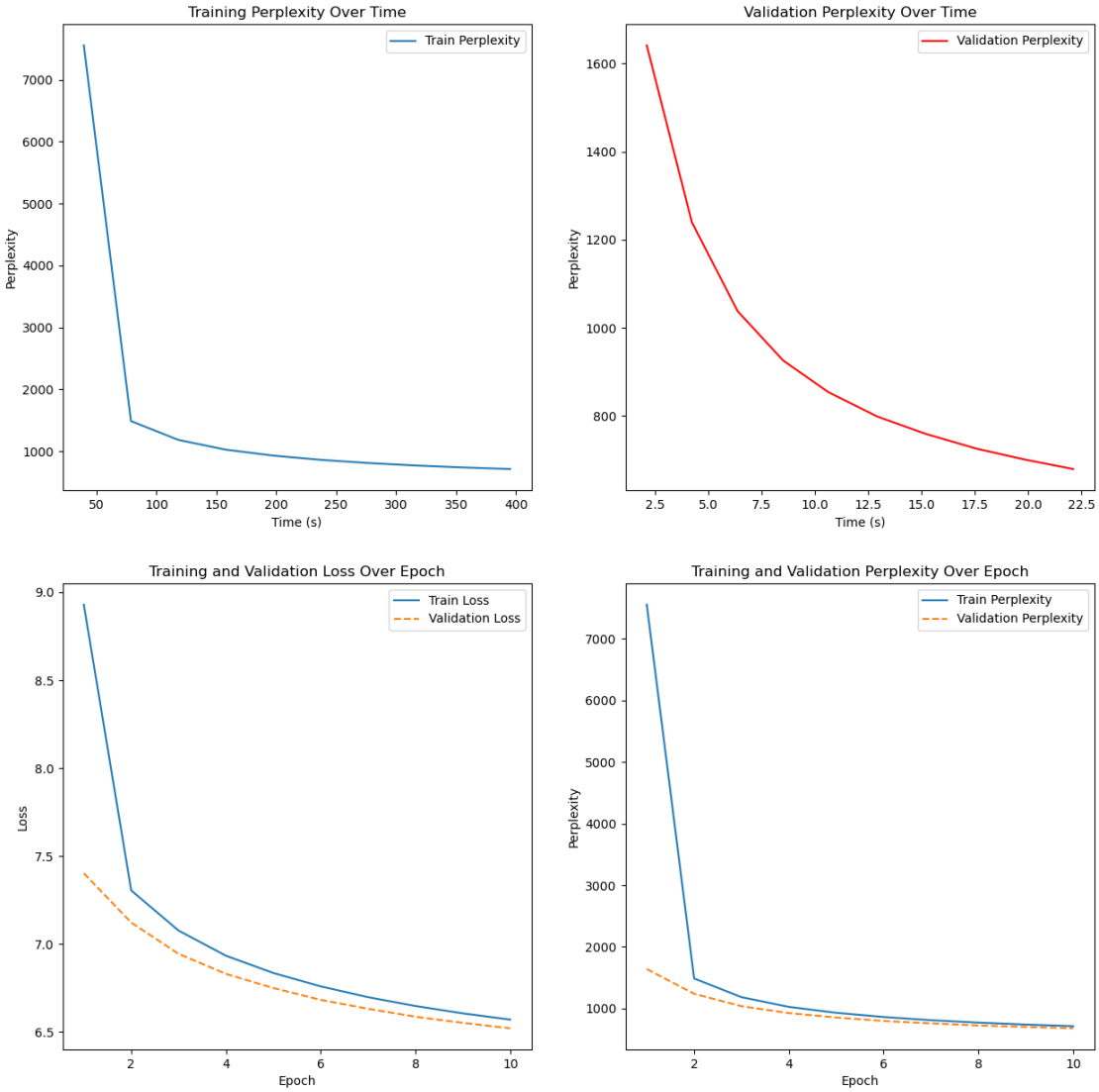
Model: gpt1, Layers: 1, Optimizer: adamw, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 7's logs in graphs



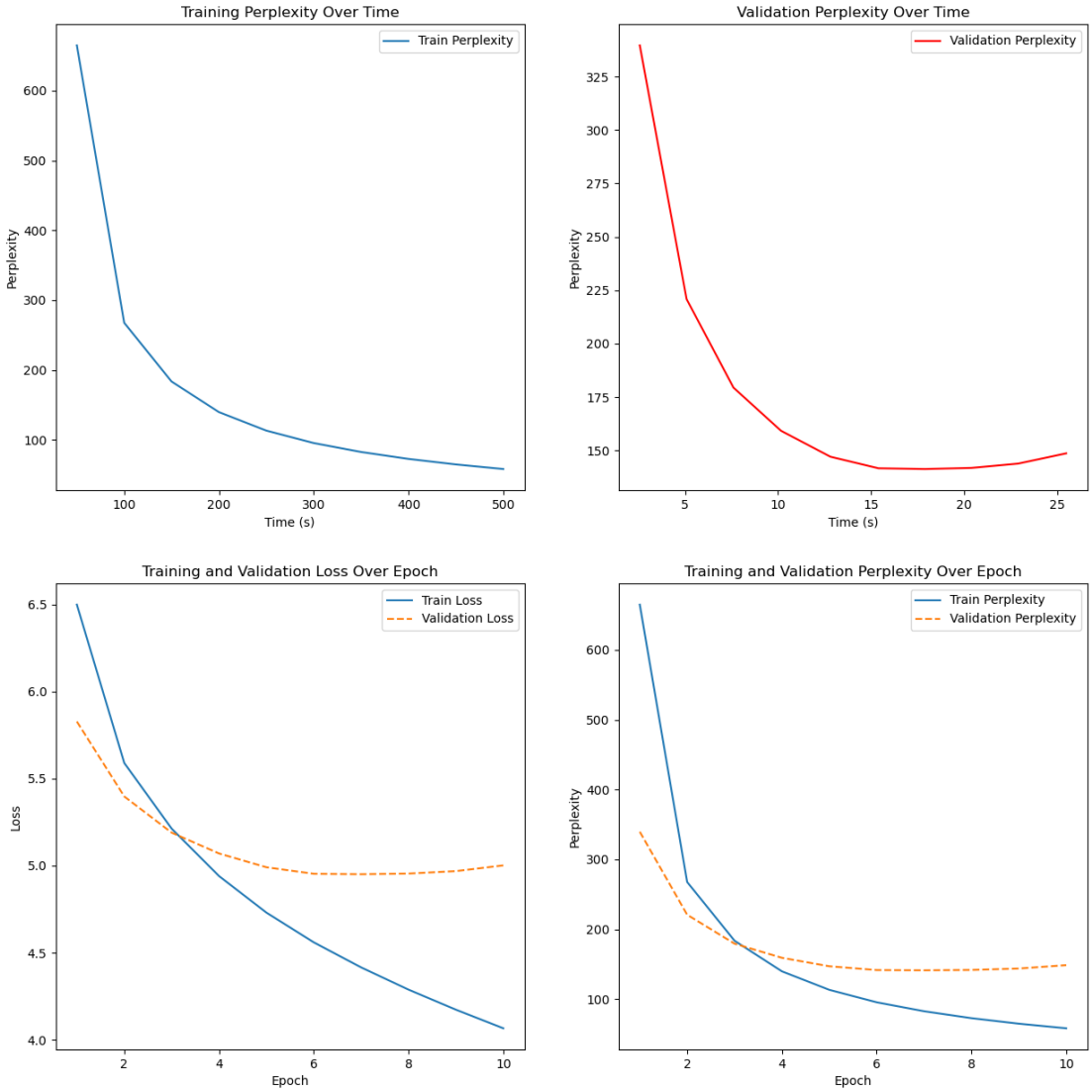
Model: gpt1, Layers: 1, Optimizer: sgd, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 8's logs in graphs



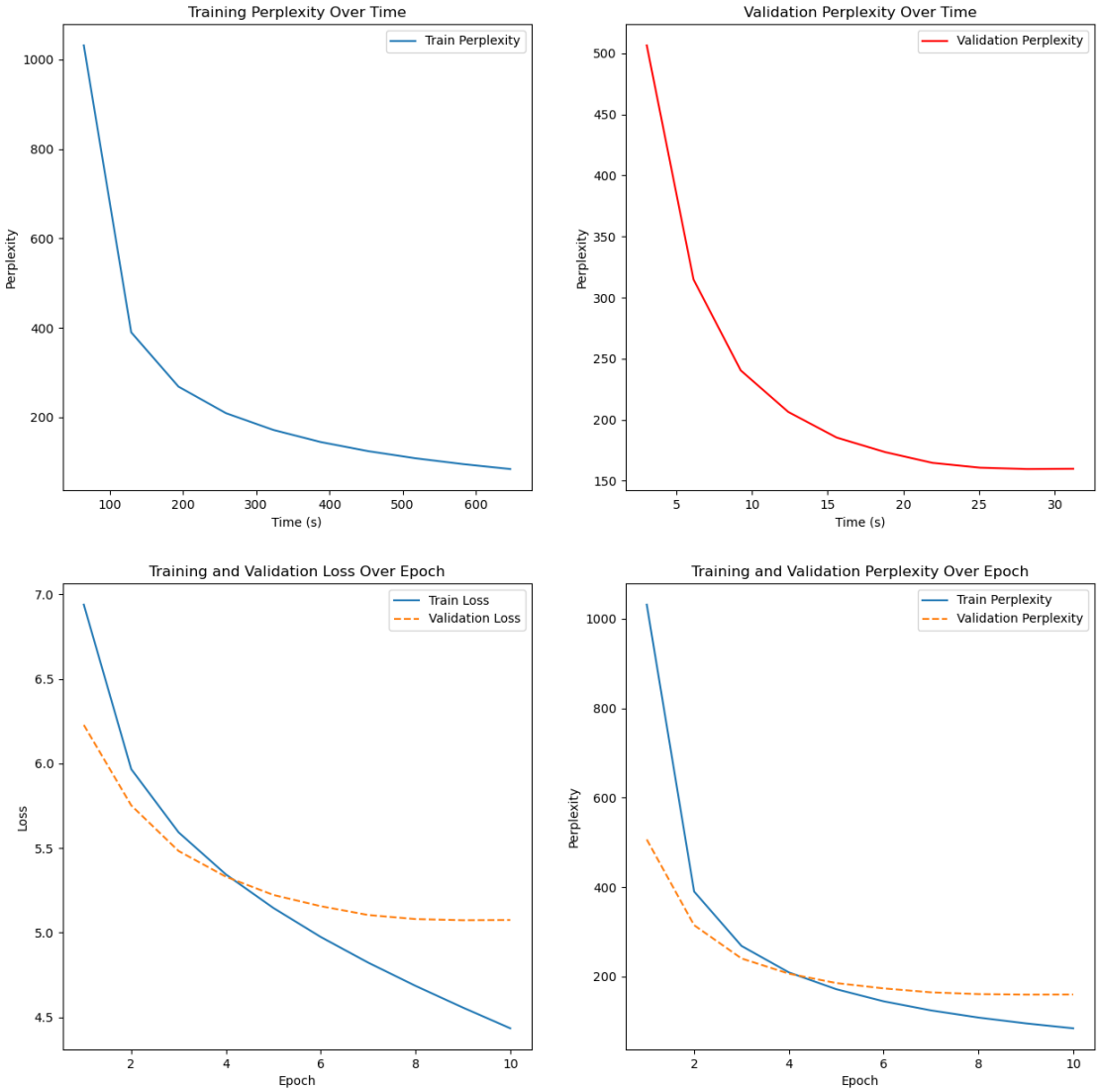
Model: gpt1, Layers: 1, Optimizer: momentum, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 9's logs in graphs



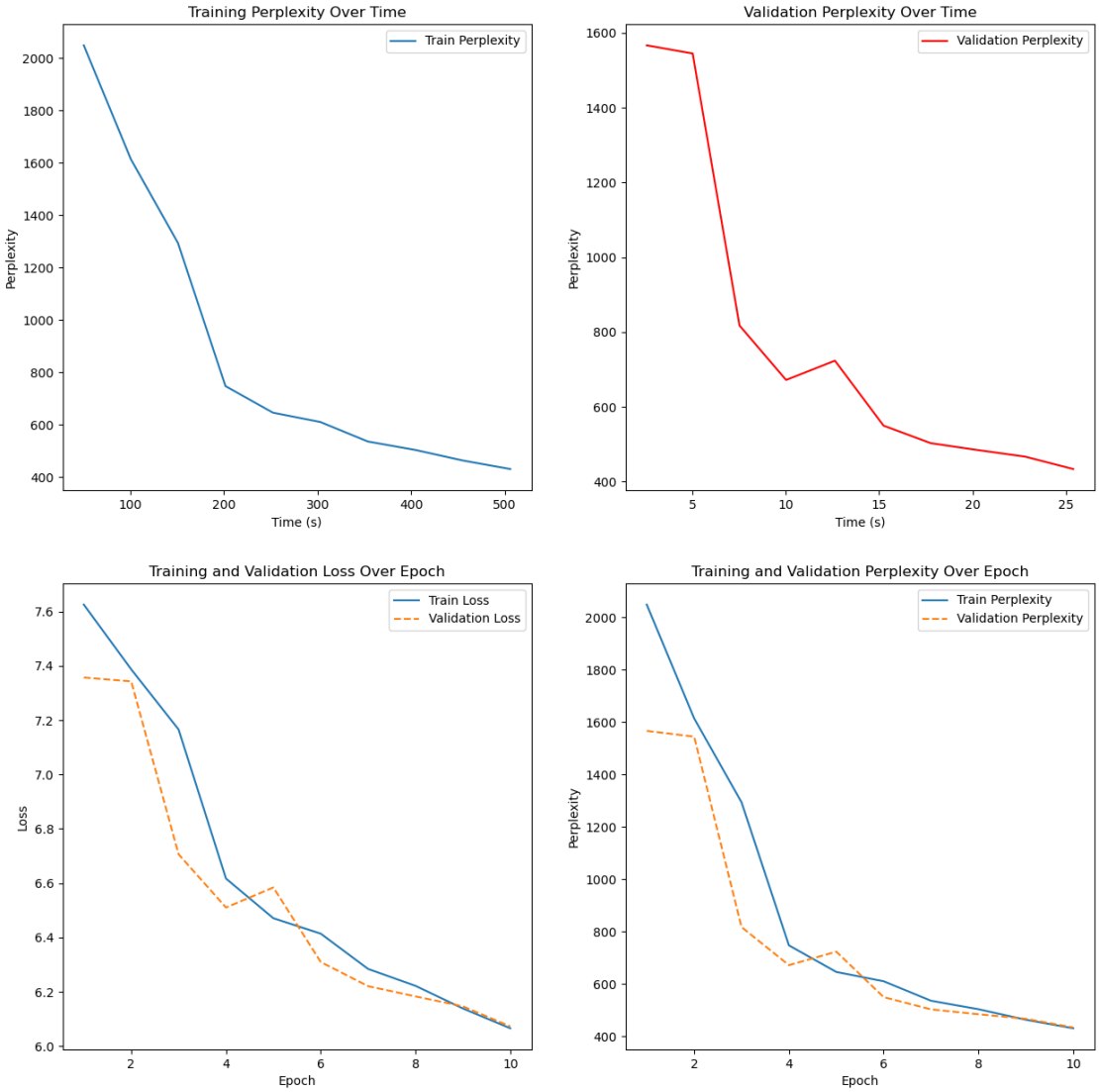
Model: lstm, Layers: 2, Optimizer: adamw, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 10's logs in graphs



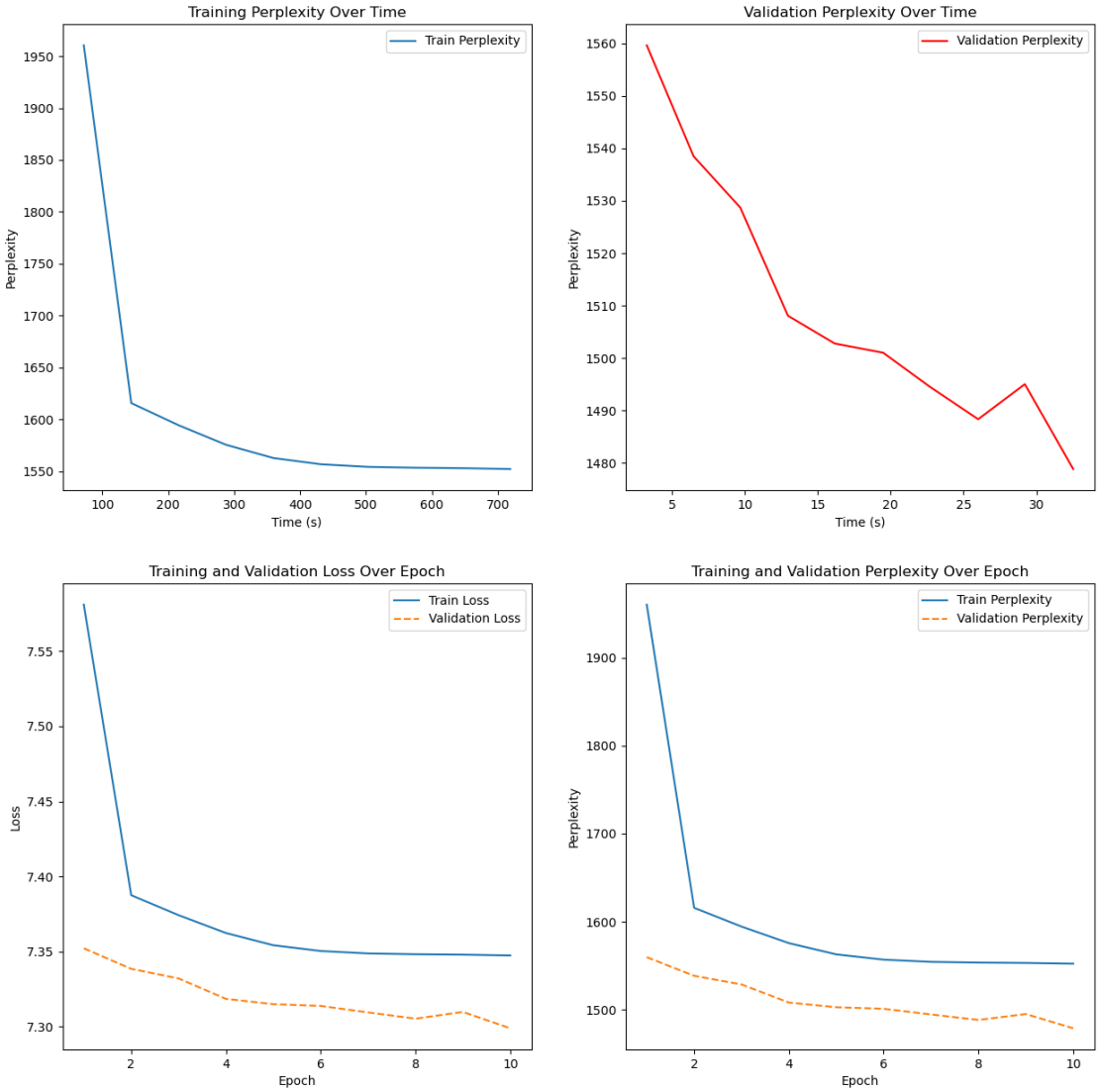
Model: lstm, Layers: 4, Optimizer: adamw, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 11's logs in graphs



Model: gpt1, Layers: 2, Optimizer: adamw, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

Experiment 12's logs in graphs



Model: gpt1, Layers: 4, Optimizer: adamw, Learning Rate: 0.001, Momentum: 0.9, Weight Decay: 0.0005, Batch Size: 16

2. Table:

Exp. num	Architecture	Layers	Optimizer	Val. PPL	Train PPL [at Best Epoch]
5	gpt1	1	adam	146.66129	86.21779 [4]
6	**gpt1**	**1**	**adamw**	**146.41322**	**86.1315** [4]
8	gpt1	1	momentum	679.82338	714.16387 [9]
7	gpt1	1	sgd	1500.89078	1617.57659 [9]
11	gpt1	2	adamw	433.99818	430.98058 [9]
12	gpt1	4	adamw	1478.83723	1552.26473 [9]
1	lstm	1	adam	144.2296	81.0613 [9]
2	lstm	1	adamw	145.1958	81.3694 [9]
4	lstm	1	momentum	1654.2252	1748.5392 [9]
3	lstm	1	sgd	2173.3796	2322.206 [9]
9	**lstm**	2	adamw	**141.3318**	**82.7597** [6]
10	lstm	4	adamw	159.6651	95.2801 [8]

Table 1: Summary of model performances across different architectures, layer depths, and optimizers. The table showcases the best validation perplexity (Val. PPL) achieved by each experiment configuration, alongside the training perplexity (Train PPL) at the epoch where the best validation perplexity was observed (in brackets). The best result for each architecture, which is measured by the lowest validation perplexity, is highlighted in bold. The results are rounded to the nearest 4 digits.

3. Here are the wall-clock training times of each experiment:

Exp	Architecture	Layers	Optimizer	Total Training Time (s)	Best Val PPL
3	lstm	1	sgd	389.04	2173.3796
4	lstm	1	momentum	389.46	1654.2252
2	lstm	1	adamw	391.52	145.1958
1	lstm	1	adam	392.5	144.2296
7	gpt1	1	sgd	393.01	1500.8908
8	gpt1	1	momentum	395.26	679.8234
5	gpt1	1	adam	396.97	146.6613
6	gpt1	1	adamw	397.35	146.4132
9	lstm	2	adamw	499.98	141.3318
11	gpt1	2	adamw	506.06	433.9982
10	lstm	4	adamw	647.0	159.6651
12	gpt1	4	adamw	718.8	1478.8372

Table 2: Experiments sorted by total training time alongside their best validation perplexity.

From what we can observe, the SGD optimizer appears to be the fastest, with Momentum trailing closely behind. Adam and AdamW optimizers require a slightly longer duration, especially as the number of layers increases.

We also notice that the training time increases with the number of layers. This is an expected observation due to the increased complexity and parameters in deeper architectures.
LSTMs generally train slightly faster than GPT1 models, with the gap widening as the number of layers increases. We notice around a 5s difference between the experiments of 1 layer that just change architecture.

If we want to prioritize wall-clock time, experiment 3 stands out with the shortest training duration of 389.04 seconds. However, we note it has the worst perplexity of 2173.3796, which indicates very poorer generalization. On the other hand, experiment 1 offers an excellent balance between training time and performance, requiring just an additional 3.5 seconds in training but achieving an impressive validation perplexity of 144.2296 which is only 3 off expirement 9 which is the best.

4. **Optimizer: Adam**
Average training time: 394.74 seconds
Average validation perplexity: 145.45
LSTM: 392.5s train time and 144.2296 perplexity
GPT1: 396.97s train time and 146.6613 perplexity

Optimizer: AdamW
Average training time: 394.44 seconds
Average validation perplexity: 145.80
LSTM: 391.52s train time and 145.1958 perplexity
GPT1: 397.35s train time and 146.4132 perplexity

Optimizer: SGD
Average training time: 391.02 seconds
Average validation perplexity: 1837.14
LSTM: 389.04s train time and 2173.3796 perplexity
GPT1: 393.01s train time and 1500.8908 perplexity

Optimizer: Momentum
Average training time: 392.36 seconds
Average validation perplexity: 1167.02
LSTM: 389.46s train time and 1654.2252 perplexity
GPT1: 395.26s train time and 679.8234 perplexity

Adam takes on average 394.74 seconds which is slightly longer than SGD and Momentum but nearly the same time as AdamW. The average validation perplexity for Adam is 145.45, which is considerably better than both SGD and momentum-based optimization. This highlights Adam's capability in terms of faster and better convergence to a minimum in complex models.
AdamW has an average training time of 394.44 seconds, which is very close to that of Adam, indicating that the addition of weight decay doesn't have a significant impact on training time. On the other hand, the average validation perplexity for AdamW of 145.80, is slightly higher than Adam, suggesting that weight decay might slightly regularize the model, poten-

tially preventing overfitting but not necessarily always leading to better generalization in our dataset.
SGD has the shortest training time of 391.02 seconds among all the optimizers. With an average validation perplexity of 1837.14, SGD performs significantly worse than the other optimizers. This could be due to the lack of adaptive learning rates, causing slower or poorer convergence, especially in complex models.
The average training time for Momentum is 392.36 seconds, which is just slightly longer than SGD but shorter than both Adam variants. Momentum significantly improves the validation perplexity to 1167.02 from SGD's 1837.14. This shows the benefits of using momentum during optimization, as it helps to smoother and potentially accelerate convergence.

Overall, we can clearly see that Adam provides a good balance between training time and model performance (as mentionned in the previous question too). Weight Decay (AdamW) has a minimal impact on training time but might help in regularizing the model. Finally, Momentum offers a compromise between SGD and Adam, providing faster training than Adam but better convergence than plain SGD.

5. Experiment 1 and 5 have a very different training times, with the GPT model being slower by 4.5 seconds. However, the difference in validation perplexities is tight, the GPT1 architecture achieves a slightly higher perplexity of 146.66 compared to the LSTM's 144.23. Although this points to a slightly better fit for the LSTM model on the validation data, the difference can be considered negligible, especially in light of the overall performance across the other experiments.

   An important observation is that Experiment 5 reaches its best perplexity at epoch 4, showing a rapid learning capability of the GPT1 model. However, a subsequent decline in performance after epoch 4 means a tendency towards overfitting, as we can see in the experiment's graph trends. In contrast, the LSTM in Experiment 1 demonstrates a more measured learning curve, achieving its optimal performance at the final 9th epoch without signs of overfitting. This gradual and steady improvement indicates a robust learning process, culminating in the best model fit towards the end of its training duration.

   Given this, if an early stopping strategy were applied to Experiment 5, the GPT1 model could potentially emerge as the more effective choice. By halting training at epoch 4, the GPT1 model would likely reach optimal performance in about half its total training time, making it more efficient while preventing the overfitting observed in its later epochs.

6. The transformer architecture, when trained with Adam and AdamW optimizers, achieves impressively low validation perplexities rapidly. This is in line with our expectations, especially with these optimizers because they are equipped to handle sparse gradients, which are often encountered in transformer architectures. However, transformers do not seem to perform as well with stochastic gradient descent and momentum when compared to the results.

   It's interesting to observe that increasing the layer depth in configurations 11 and 12 actually worsens the perplexity. Both experiment 11 and 12 could not compete in terms of perplexity to their counter part, the single layer in experiment 6. This shows that adding depth to the

model does not linearly translate to enhanced performance and this could be a manifestation of overfitting. However, examining the training progress graphs for experiments 11 and 12, we see that these models may require extended training periods likely due to the increased complexity and capacity the deeper models, necessitating more epochs to adequately learn from the data.

We expect transformers to perform exceptionally well, and it shows, especially with Adam and AdamW optimizers. Hwever, while transformers have indeed demonstrated state-of-the-art performance, their effectiveness can be highly contingent on the specific dataset and hyper-parameter settings. Deeper layers typically require larger datasets to perform optimally and to prevent overfitting. Alternatively, the more complex models may also necessitate extended periods of training to fully capitalize on their depth. Furthermore, incorporating the correct optimizer is crucial for maximizing their potential. We see that when SGD and Momentum optimizers led to less impressive results compared to Adam and AdamW. Transformers, while powerful, still require careful tuning and consideration of their training context to achieve their best performance.

7. Table showing the different average GPU (16GB Nvidia V100) memory usages per model recorded by `nvidia-smi`:

Exp num	Architecture	Layers	Optimizer	Average GPU Memory Usage (MB)
3	lstm	1	sgd	3881.9
7	gpt1	1	sgd	4335.6
1	lstm	1	adam	4457.7
2	lstm	1	adamw	4484.2
4	lstm	1	momentum	4484.2
9	lstm	2	adamw	4553.6
10	lstm	4	adamw	4691.2
5	gpt1	1	adam	4716.0
6	gpt1	1	adamw	4716.0
8	gpt1	1	momentum	4716.0
11	gpt1	2	adamw	5052.4
12	gpt1	4	adamw	5705.2

Table 3: Ordered average GPU memory usage for each experiment.

We do observe a clear pattern in the GPU memory usage across the different configurations.

- SGD: Both experiments 3 and 7, which use SGD as their optimizer, consume the least amount of GPU memory.This can be attributed to the nature of the SGD optimizer, unlike adaptive optimizers like Adam or AdamW, SGD does not maintain additional running averages or moment estimates for each parameter, leading to a smaller memory footprint.
- LSTM: The experiments with the LSTM architecture (1, 2, 4, 9, 10), the GPU memory usage remains fairly consistent. A notable distinction emerges with the number of layers (as anticipated), the memory consumption increases with the addition of layers, as observed in experiments 9 and 10. Each additional layer in the LSTM introduces new weights, biases, and intermediate computations, resulting in an augmented memory demand. Furthermore, the inherently sequential nature of LSTM processing, which limits parallelization, contributes to its relatively lower memory footprint compared to more parallel architectures like transformers.
- GPT1: It consistently exhibits the highest memory usage. Specifically, experiments 5, 6, and 8 register identical memory usage, signifying that the choice of optimizer (whether Adam, AdamW, or Momentum) does not significantly influence memory consumption. However, when layer count escalates, as seen with experiments 11 and 12, there's a marked increase in memory. This could be due to the inherent complexity and dense connections within the transformer model, where each additional layer introduces a greater number of parameters, requiring more memory for storage and computation. Additionally, the self-attention mechanism in transformers, computes relationships across all elements in a sequence, contributing significantly to memory usage, especially as the model scales in depth.

Overall, the key factors influencing GPU memory usage are the model's architecture (with transformers typically requiring more than LSTMs), the depth or size of the model, and optimizer choice (batch size difference would've also influenced).

8. For this question, I am mainly looking at the graphs.
LSTM Models (Experiments 2, 9, 10)

In experiment 2 (LSTM with AdamW, 1 Layer) we see a good convergence of the training and validation perplexities. However, we do notice that the training loss surpasses the validation loss after the 3rd epoch, the same applies for their perplexities. This could indicate a start of our model sticking too much to our training data but at least the losses and validations aren't getting worse.

In experiment 9 (LSTM with AdamW, 2 Layers), the perplexity and validation loss start climbing back up after 20s of validation, it suggests the model has reached its best point at around 20s and afterward starts deteriorating in performance clearly showing further overfitting on the training data. Furthermore, as mentionned for experiment 2, the gap between the two losses is even bigger.

In experiment 10 (LSTM with AdamW, 4 Layers), the smaller gap between training and validation loss can be attributed to the increased capacity allowing the model to represent more complex functions. Howeber the validation loss and perplexity do plateau slightly earlier, maybe because of the model's ability to express more on the validaiton.
A larger model (more layers or neurons) has a greater capacity to fit complex data but also has a higher propensity to overfit. This seems evident when comparing Experiment 9 and 10. The 4-layer LSTM can fit the training data better and delays overfitting compared to the 2-layer LSTM.

Furthermore, when validation loss/perplexity increases after a certain epoch, it means the model starts fitting noise in the data beyond this point. Adam and AdamW have a very fast convergence, which can sometimes lead to overfitting, especially on the smaller models. As we can see in the LSTM experiments, momentum and SGD do not decreases as rapidly in the first 2 epochs compared to the ones with Adam and AdamW

GPT Models (Experiments 6, 11, 12)

In experiment 6 (GPT with AdamW, 1 Layer), we see a rapid convergence in just 2 epochs suggesting the model quickly fits on the training data. However, after approximately 10 seconds of validation wall-clock time (or at the 5th epoch), the validation loss and perplexity begin to increase, indicating clear overfitting. Additionaly, the training loss and perplexity outperform the validation loss and perplexity after the 2nd epoch, clearly showing that our model is sticking too much to our training data. The adaptive nature of the AdamW optimizer, coupled with the efficient GPT architecture—even with a single layer—could be responsible for this quick overfitting.

In experiment 11 (GPT with AdamW, 2 Layers), we notice minor fluctuations in validation loss and perplexity around 12 seconds of validation which could suggest that the model is beginning to overfit, but this behavior isn't as drastic as in experiment 6 (loss/perplexity

curve steadily climbing back up). Both training and validation loss decrease over epochs, which is a positive sign of learning. We notice the training loss is consistently lower than the validation loss, which is expected since the model is trained on the training data. More importantly, the validation loss is following the trend of the training loss without a significant gap between them, which suggests that the model is not overfitting. The same can be said for both perplexities, another indicatation that the model is generalizing well and is not overfitting.

In experiment 12 (GPT with AdamW, 4 Layers), the model shows a slower convergence after the 2nd epoch, which is expected due to the increased complexity from additional layers. The jagged but overall downward trend of the validation perplexity indicates learning is happening but with some degree of variance, which could be a consequence of the larger capacity of the model requiring more data or time to train effectively. It learns effectively from the training data while generalizing well to its unseen validation data. Notably, the validation perplexity and loss remained consistently lower than those of the training set throughout the training process. This tells us that it hasn't begun to overfit, as it performs even better on the validation set than on the training set. However, while it's loss and perplexity trends are downwards, showing steady learning, it does stagnate around the 6th epoch meaning it might need more training time. It seems it would be beneficial to continue training beyond epoch 10 to ensure this learning trend persists and that we have enough data for the larger model.

From what we observed, it appears that GPT models, especially those with fewer layers, tend to overfit quicker than LSTM models. The rapid convergence of GPT models in the early epochs and the subsequent increase in validation loss and perplexity suggest that these models might be capturing noise in the training data. In contrast, the LSTM models show a more gradual onset of overfitting, with the 4-layer LSTM delaying overfitting more effectively than the 2-layer model. Transformers are very powerful, even with a single layer, they can overfit easily, especially when combined with adaptive optimizers like AdamW. To avoid overfitting, we could do the following:

- early stopping
- regularization (like dropout or L2 regularization)
- learning rate scheduling which will reduce the learning rate upon a plateauing of validation performance
- simply adjusting the optimizer hyperparameters, layer sizes, or other relevant parameters.
- increase the size and diversity of the training dataset
- cross-validation.