Due Date: March 22nd 2021, 11pm EST

Problem 1

Implementing an LSTM (9pts) The code for this part is submitted to the Gradescope. The code for all problems can be found here.

Problem 2

Implementing a GPT (Generative-Pretrained Transformer) (27pts) The code for this part is submitted to the Gradescope. The code for all problems can be found here.

Problem 2.5

Module `MultiHeadedAttention` has 4 sets of learnable parameters: $\{\texttt{Query}, \texttt{Key}, \texttt{Value}, \texttt{Output}\}$. Each set corresponds to a weight matrix and a bias vector thus our learnable parameters are: $\{W_Q, W_K, W_V, W_O, b_Q, b_K, b_V, b_O\}$. What are their dimensions? $\{W_Q, W_K, W_V\}$ are multiplied by the concatenated output of the previous layer, and should provide the concatenated Query, Key, Value to be split and fed to different heads of the current layer, so their input dimension and output dimension are both the same as embedding size which is itself $\texttt{num\_heads} \times \texttt{head\_size}$. Now if we call the concatenated output of the previous layer X, and $d = \texttt{num\_heads} \times \texttt{head\_size}$ then dimensions of X would be $X^{1\times d}$, if it's multiplied by either of $\{W_Q, W_K, W_V\}$, then

$$X^{1\times d} W_Q^{d\times d} = X_Q^{1\times d}$$

So the bias that's gonna be added to X_Q should be of dimension $b_Q^{1\times d}$.
The same goes for output weight matrix and its bias as well. W_O is multiplied by concatenated outputs of the current layer and should result in a concatenated vector for the next layer so it has dimensions of $W_O^{d\times d}$, and its bias $b_O^{d\times d}$. So in summary these are our learnable parameters with their dimensions: $\{W_Q^{d\times d}, W_K^{d\times d}, W_V^{d\times d}, W_O^{d\times d}, b_Q^{1\times d}, b_K^{1\times d}, b_V^{1\times d}, b_O^{1\times d}\}$. Hence the number of learnable parameters for this module would be:

$$n = 4 \times d^2 + 4 \times d = 4d(d+1), \qquad d = \texttt{num\_heads} \times \texttt{head\_size}$$

In our problem, $\texttt{num\_heads} = 12, \texttt{head\_size} = 64$, therefore $n = 4 \times 768 \times 769 = 2\,362\,368$

Problem 3

Training language models and model comparison (25pts)

1. For plots please refer to figures 1,2,3,4,5,6,7,8. Now let's analyze the results (we'll use these discussions for subsequent questions). Figures with odd index correspond to plots in which the x-axis is *epochs*, and figures with even index correspond to plots in which the x-axis is *wall-clock time*.

    - 1,2: Figure 1 shows that for *LSTM with a single layer*, Adam and AdamW perform much better than momentum and SGD, and they perform pretty much the same. AdamW is only slightly better. Although loss might be misleading that momentum and SGD are

maybe learning something but only slightly worse than Adam and AdamW, but looking at Perplexity (PPL) values, it is clear that with such high values of PPL, momentum and SGD are doing a terrible job of learning, thus we should confidently discard them. Figure 2 shows a similar pattern as a function of *wall-clock time*, and since they require about the same compute so we expect them to learn in the same duration, the reason that momentum takes longer is not due to the model. I have run the codes on a cluster and due to scheduling, this specific experiment took longer but it's not indicative of its training being longer.

- 3,4: Figure 3 shows that for *GPT-1 with a single layer*, again, Adam and AdamW perform much better than momentum and SGD, and they perform pretty much the same. Again, AdamW is only slightly better. Looking at Perplexity (PPL) values, the advantage of Adam and AdamW becomes much clearer. Though SGD is still bad just like its performance with LSTM, momentum's effect with GPT-1 has improved and it has achieved a lower PPL. Figure 4 shows a similar pattern as a function of *wall-clock time*, and again since all GPT-1 networks are similar, they require about the same compute so we expect them to learn in the same duration.
- 5,6: These figures are more interesting. Figure 5, the plot for loss shows that when optimized by the best optimizer (AdamW), among architectures of similar number of layers, LSTM does a better job of learning which is a bit counter-intuitive. But this will be resolved to some extent when considering all architectures (later). Also we should note that increasing the number of GPT-1 layers is not helping, and in fact it deteriorates the performance. The same goes with the plot for PPL. Figure 6 also shows that LSTM achieves a lower loss with less time required for optimizing over 10 epochs.
- 7,8: After analyzing all the plots and results, I decided to plot the best performing architectures on the same plot to assess them more carefully. This is what I've shown in figures 7,8. Figure 7,8 show that when all models are optimized by AdamW, indeed GPT-1 achieves lower loss and PPL slightly faster than the best performing LSTM (2 layer LSTM), and in particular much faster than LSTM with 4 layers which is a bit of a surprise given the huge number of parameters in a GPT-1. Again we see that AdamW and Adam perform quite the same, this is not contradicting to the theoretical part where we showed that Adam with L2 penalty is flawed and regularizes in an imbalanced way. The reason is that Adam was used here with weight decay (observable in the template codes). So we should expect this behavior.
  But there's a disadvantage to GPT-1, it shows overfitting, as the loss and PPL slowly increase as we go forward in time. So this undesired behavior might act hostile to using it when concerned with generalization. But on the other hand, LSTMs manifest a rather stable validation loss and PPL thus they might better suit the settings in which we're concerned with generalization (Question 3).

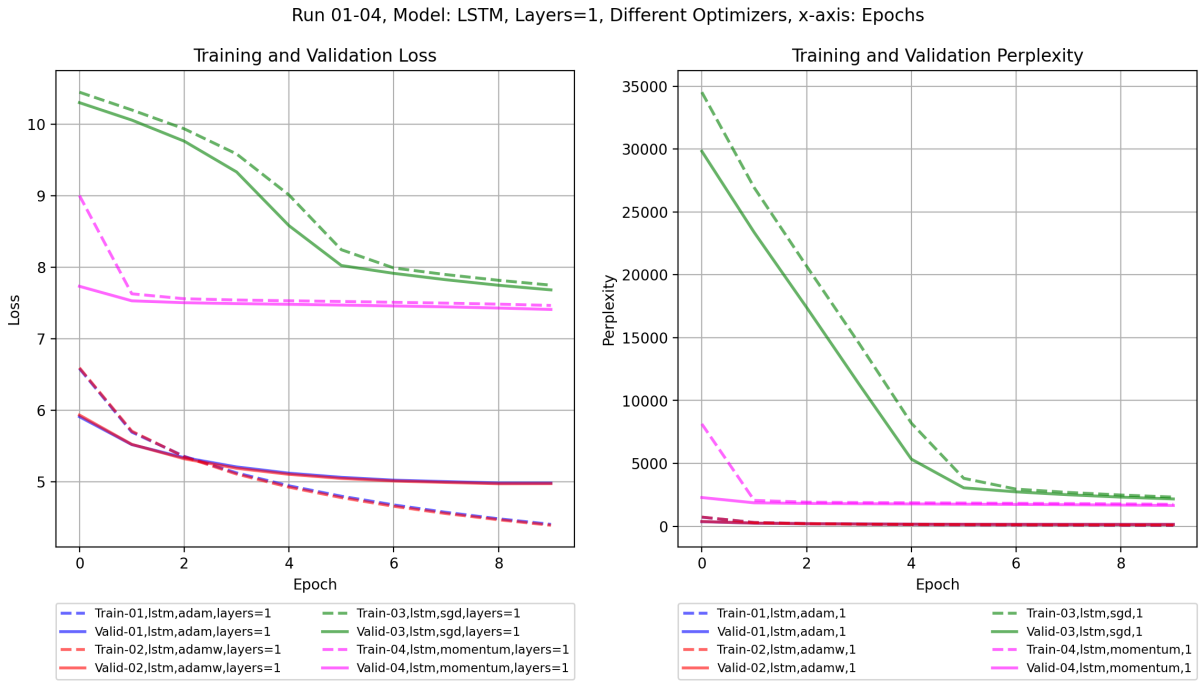

Figure 1: Loss and perplexity as a function of **epochs** for training and validation set trained on **LSTM** with a single layer.

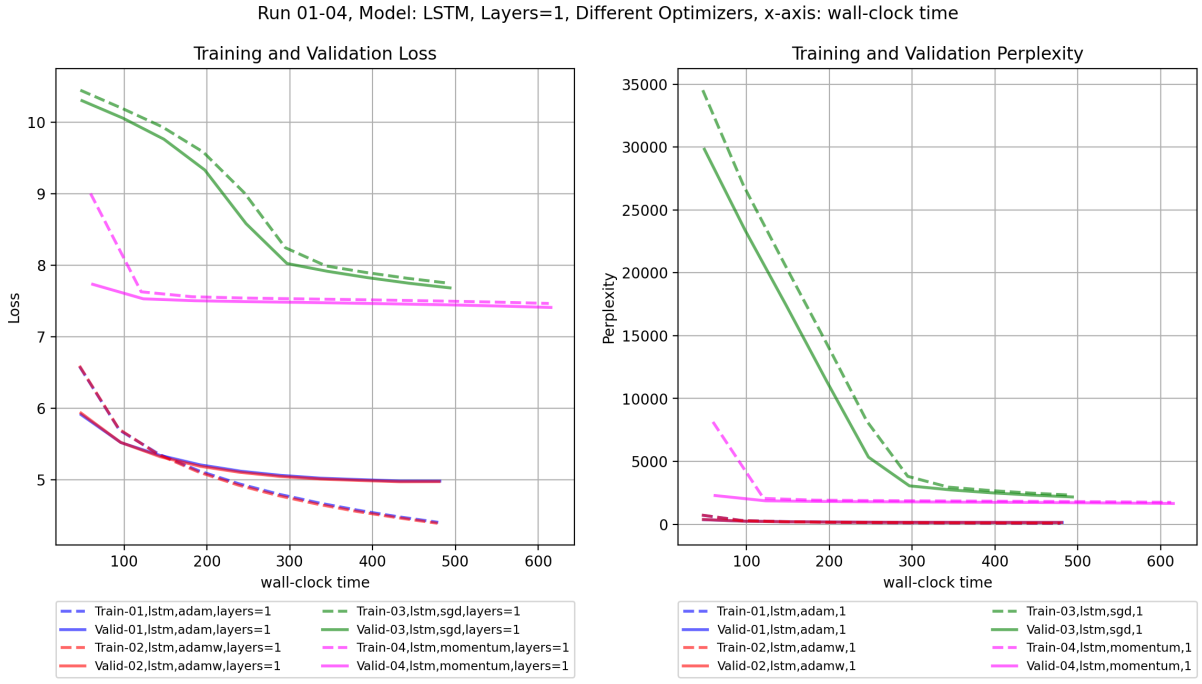

Figure 2: Loss and perplexity as a function of **wall-clock time** for training and validation set trained on **LSTM** with a single layer.

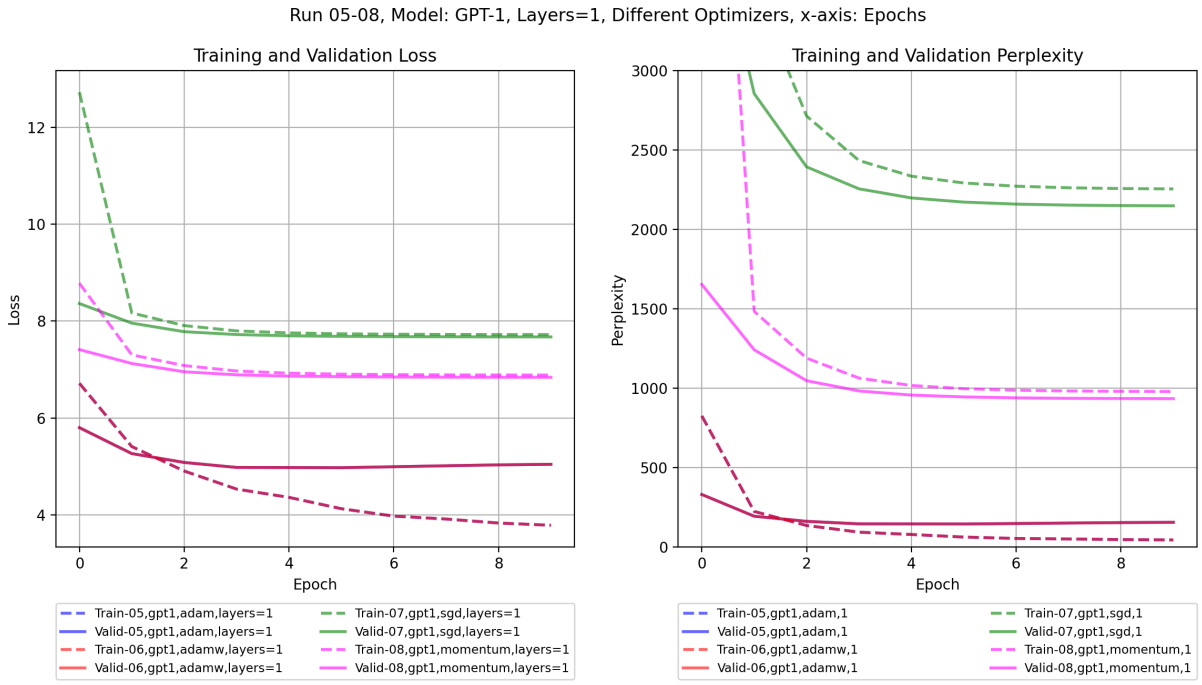

Figure 3: Loss and perplexity as a function of **epochs** for training and validation set trained on **GPT-1** with a single layer.

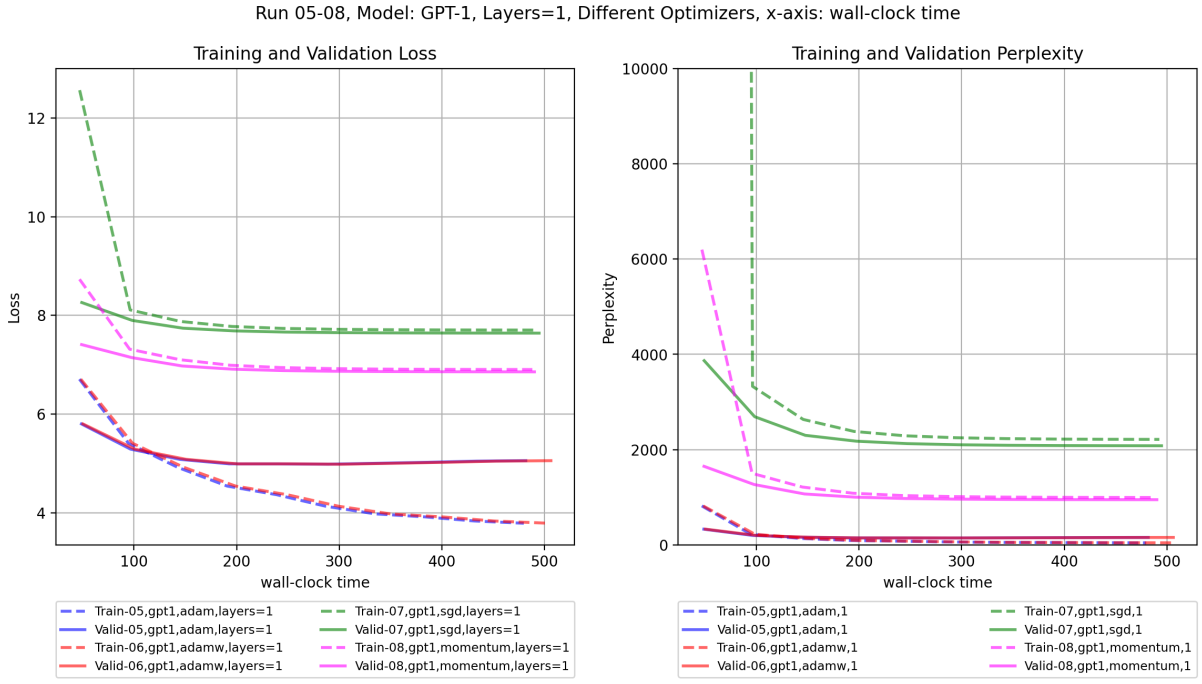

Figure 4: Loss and perplexity as a function of **wall-clock time** for training and validation set trained on **GPT-1** with a single layer.

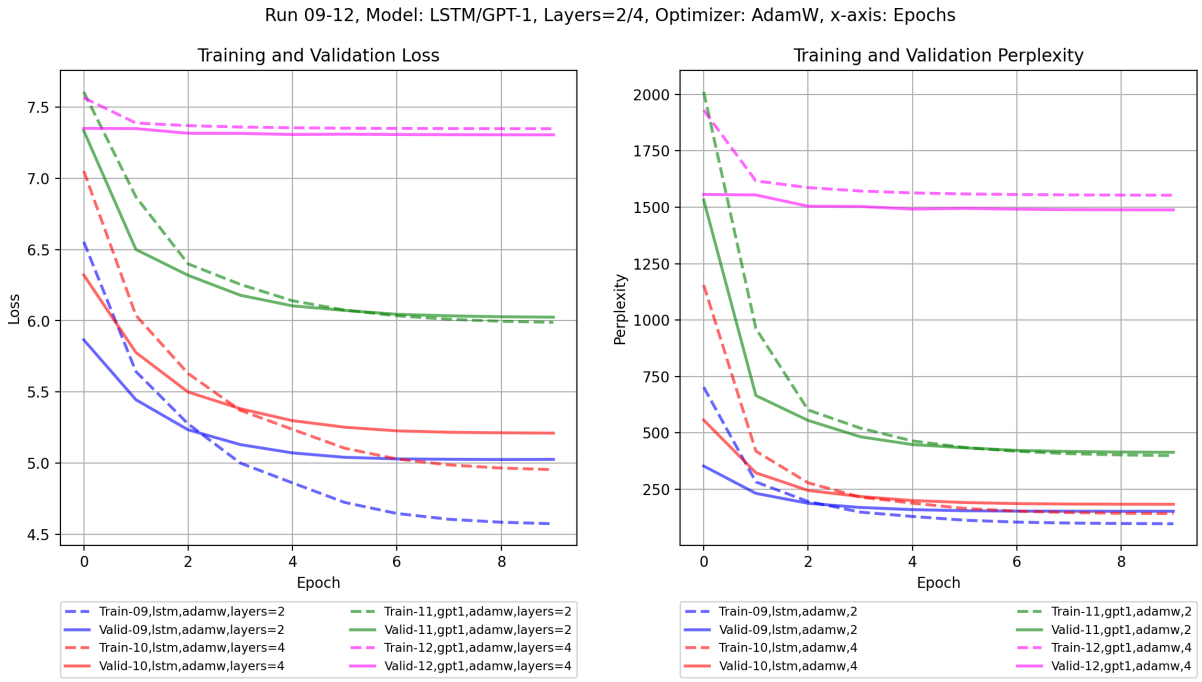

Figure 5: Loss and perplexity as a function of **epochs** for training and validation set trained on **LSTM or GPT-1** with layers $\in \{2, 4\}$.

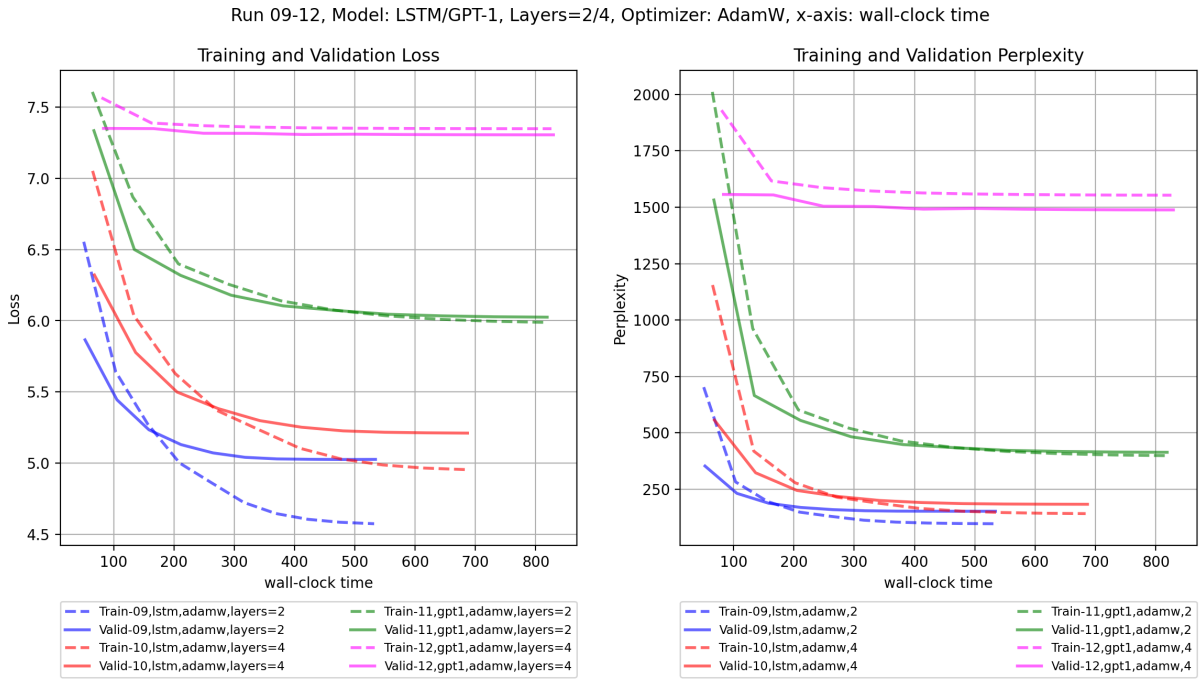

Figure 6: Loss and perplexity as a function of **wall-clock time** for training and validation set trained on **LSTM or GPT-1** with layers $\in \{2, 4\}$.

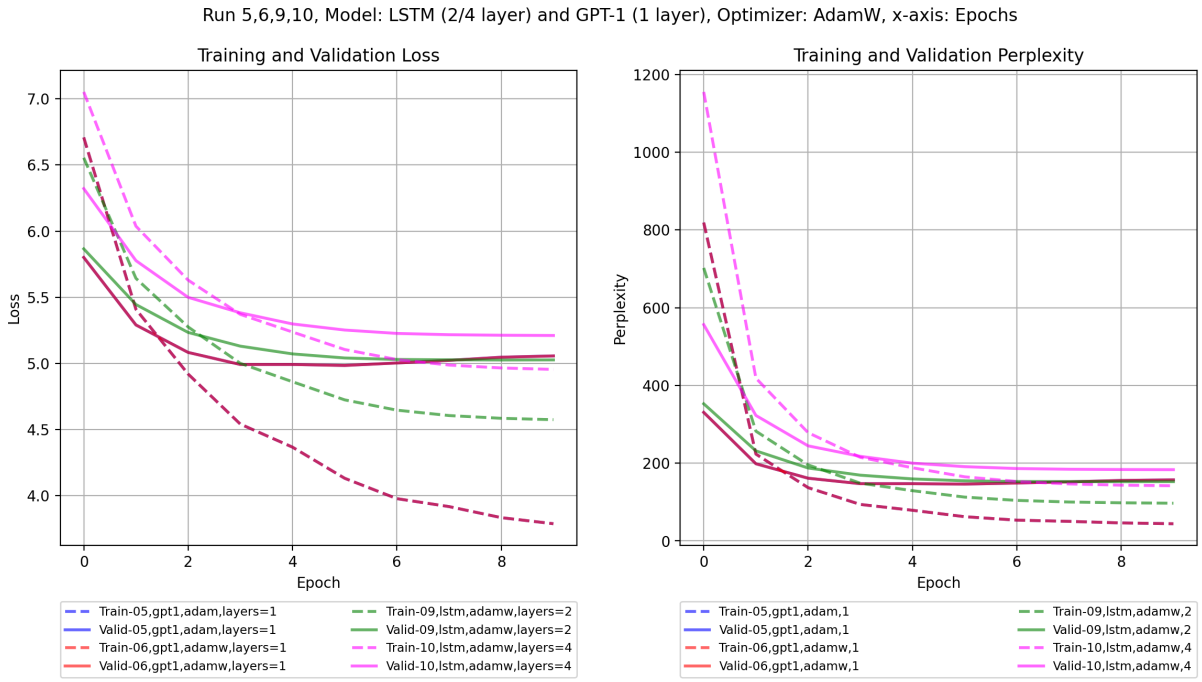

Figure 7: Loss and perplexity as a function of **epochs** for training and validation set trained on best of combinations.

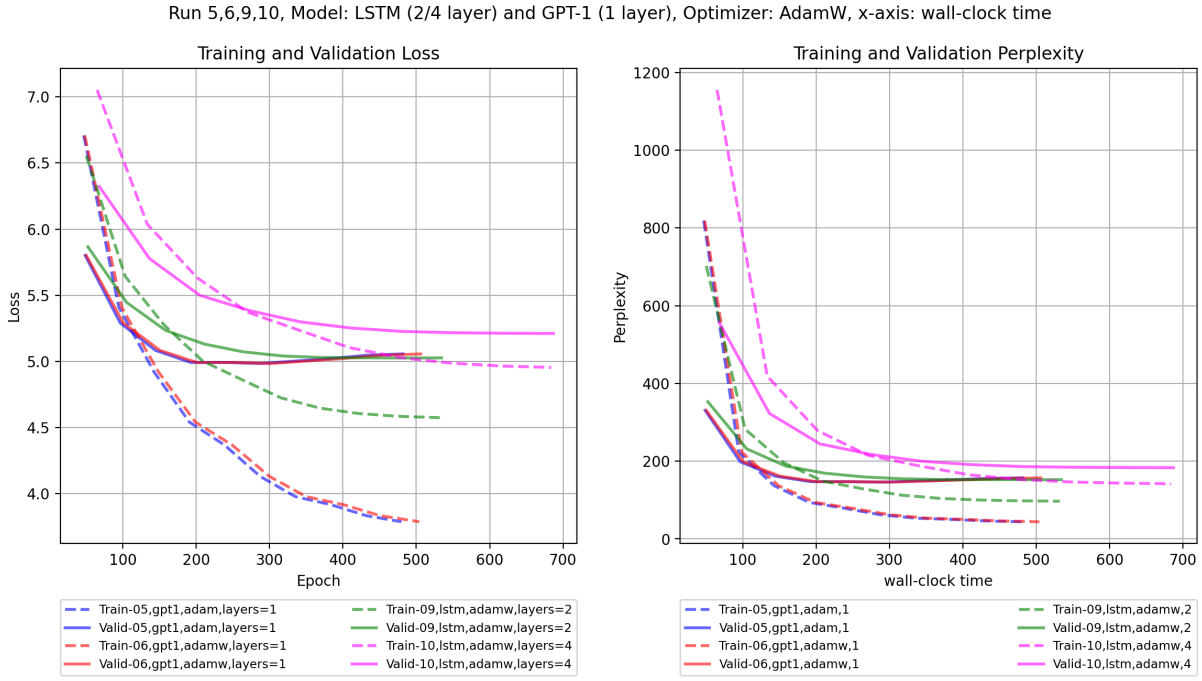

Figure 8: Loss and perplexity as a function of **wall-clock time** for training and validation set trained on best of combinations.

2. Please refer to tables 1, 2.

Exp. ID	Architecture	Layers	Optimizer	Validation PPL	Training PPL
1	LSTM	1	Adam	145.81	81.86
2	LSTM	1	AdamW	144.46	87.28
3	LSTM	1	SGD	2174.07	2322.51
4	LSTM	1	Momentum	1653.52	1748.55
5	GPT-1	1	Adam	144.47	61.95
6	GPT-1	1	AdamW	**144.44**	**61.96**
7	GPT-1	1	SGD	2081.28	2214.83
8	GPT-1	1	Momentum	933.77	978.67
9	LSTM	2	AdamW	**138.54**	**82.52**
10	LSTM	4	AdamW	157.70	96.21
11	GPT-1	2	AdamW	413.68	398.86
12	GPT-1	4	AdamW	1488.04	1553.22

Table 1: Comparing all 12 experiments (sorted by experiment ID). The best architecture is **LSTM with 2 layers and optimized by AdamW**. The second best architecture is **GPT-1 with 1 layer and optimized by AdamW**. Exp. ID is experiment numbers as in the runner script. The minimum value of perplexity (PPL) over 10 epochs on the validation set is retrieved, and constitutes the column *Validation PPL*. The PPL for corresponding epoch (with minimum Validation PPL) on the training set makes for the other column *Training PPL*.

Exp. ID	Architecture	Layers	Optimizer	Validation PPL	Training PPL
1	LSTM	1	Adam	145.81	81.86
2	LSTM	1	AdamW	144.46	87.28
3	LSTM	1	SGD	2174.07	2322.51
4	LSTM	1	Momentum	1653.52	1748.55
9	LSTM	2	AdamW	**138.54**	**82.52**
10	LSTM	4	AdamW	157.70	96.21
5	GPT-1	1	Adam	144.47	61.95
6	GPT-1	1	AdamW	**144.44**	**61.96**
7	GPT-1	1	SGD	2081.28	2214.83
8	GPT-1	1	Momentum	933.77	978.67
11	GPT-1	2	AdamW	413.68	398.86
12	GPT-1	4	AdamW	1488.04	1553.22

Table 2: Sorting all experiments by architecture, layer, and optimizer. The best architecture is **LSTM with 2 layers and optimized by AdamW**. The second best architecture is **GPT-1 with 1 layer and optimized by AdamW**. Exp. ID is experiment numbers as in the runner script. The minimum value of perplexity (PPL) over 10 epochs on the validation set is retrieved, and constitutes the column *Validation PPL*. The PPL for corresponding epoch (with minimum Validation PPL) on the training set makes for the other column *Training PPL*.

3. In terms of wall-clock time, **GPT-1 with a single layer optimized by AdamW** yields a slightly worse performance of 144.44 compared to 138.54 for the the best validation PPL

achieved by **LSTMs (AdamW) with two layers** but is faster than that (479 seconds compared to 532 seconds). So for wall-clock time sensitive settings, from these experiments we may conclude that **GPT-1 with a single layer optimized by AdamW** is better. However, as was mentioned earlier in answer to question 1, figures 7,8 show overfitting for GPT-1, thus we may adopt **LSTM with 2 layers optimized by AdamW** to achieve better generalization and avoid overfitting.

4. As we discussed it for figures 1,2 when answering question 1, SGD alone performs very poorly and is not capable of finding a good solution. Although according to the trends there's the chance that after a very long training, SGD could find a low loss, but due to nature of the problem SGD is less likely able to do so and requires other techniques to reasonably converge to the optimal solution. Momentum has helped learning but it still plateaus early on during the training. I tried using a learning rate scheduler but it didn't help momentum get out of that plateau. It is still an improvement though. Weight decay when combined with Adam and taken into account in the proper place (according to AdamW which we analyzed in theoretical part of the assignment) yields the best performance, and thus shows for language modeling, momentum is not enough, and we should descent adaptively for each weight. AdamW allows for L2 penalty to help even more and regularize the objective to learn slightly better solutions. The exact same observation goes for configurations 5-8 and there exists a completely similar pattern 3,4.

5. According to my observations, experiment 5 had a slightly better performance but they are really close and the improvement is pretty marginal. On the other hand experiment 5 shows overfitting behavior which turns our attention to LSTM for a better generalization. So purely looking at validation PPLs one might pick GPT-1 (exp. 5), but being more careful with generalization, one would probably adopt LSTM.

6. No absolutely they're not. It's a relief though that the best performing GPT-1 architecture could slightly beat LSTM. But I expected much more significant improvements. However I started to think that probably transformers should be much larger to manifest their power, otherwise it seems they have a very hard time beating small LSTMs. However LSTMs are known to not scale so well when sequences are much longer and architectures are deeper and that's mainly due to the sequential nature of computation with LSTMs causing very long-term dependencies. So my conjecture would be that in order to observe the weakness of LSTMs and advantages of transformers we need to to compare deeper LSTMs and transformers with much longer sequences with long-term dependencies to see the importance of attention and computing based on the whole sequence (transformer) where sequential computation and backpropagation (LSTM) breaks.

7. Please refer to table 3. The observations pretty much match our expectations.
   For experiments 1-4, we are using a single layer LSTM with only optimizers being changed, thus we don't expect much variance as the number of parameters used in training are almost identical, and we see that they use the same amount of GPU memory as well.
   In experiments 5-8, we're training a single layer transformer architecture, so the number of parameters is larger than an LSTM and we expect an increase in memory usage, but it's a single layer transformer so it's reasonable that the increase is not too much (about 300 Mb). Experiment 9 is a 2 layer LSTM, so more parameters than a single layer LSTM, and thus an

increase in memory usage is expected. Note that this increase is less than the increase we had when going from 1 layer LSTM to GPT-1. Then experiment 10 is a 4 layer LSTM so it should consume more memory than a 2 layer LSTM (exp. 9) and it's doing so, so observation is not contradictory.
There's a larger gap when going to experiments 11,12 as they correspond to a 2 layer and 4 layer GPT-1 respectively so it's totally predictable that memory usage jumps by about 600 Mb and 1 Gb respectively as GPT-1 consists of huge layers and keeping track of them during computations and updates clearly consumes more memory.
So GPU memory footprints show a feasible pattern of increase and decrease given the size of their architectures.

Exp. ID	Memory Used (Mb)
1	4193
2	4193
3	4193
4	4193
5	4489
6	4489
7	4489
8	4489
09	4318
10	4389
11	4916
12	5791

Table 3: Average steady state GPU memory used for different experiments.

8. Please refer to figure 9. From that figure it seems that overfitting is more severe with GPT-1, although we cannot observe that behavior with larger models (as they are not trained well to convergence). But increase on the figure to the left (LSTMs) seems to be tiny, but it's *persistent* and happens for all of them. My guess would be that since GPT-1 has larger number of parameters, when trained to convergence it has a higher chance of being overfitted compared to LSTM. Validation performance seems more robust for LSTMs. Also we see that GPT-1 with a single layer can continuously decrease the training loss to lower values compared to LSTM which again is a sign of overfitting when validation loss is not improving. So I would conclude that GPT-1 is more prone to overfitting. One more observation is that most of the times overfitting happens with Adam and its variant AdamW consistent with what we learned in class that Adam can cause overfitting. This observation is not strong though, since momentum and SGD have not converged to a *good* solution and this won't allow much room for an accurate judgment.
To avoid overfitting, there are steps one can take before starting and during training. Avoiding severely over-parametrized architectures before training is a good practice. During training a learning rate scheduler could be helpful to get out of plateaus and achieving global and generalizable solutions. Also one obvious approach is early stopping, i.e. monitoring the stopping criterion and stopping the training on the plateau before it increases. We're using AdamW so it means we have L2 penalty but maybe it's the case that it's not enough to

improve generalization. So one might use the validation set to assess this hypothesis as well. In this case, given the good performance when the loss is at minimum, it seems that early stopping would be the most effective as it seems unlikely that with such hyperparameters and architectures we obtain a lower validation loss.

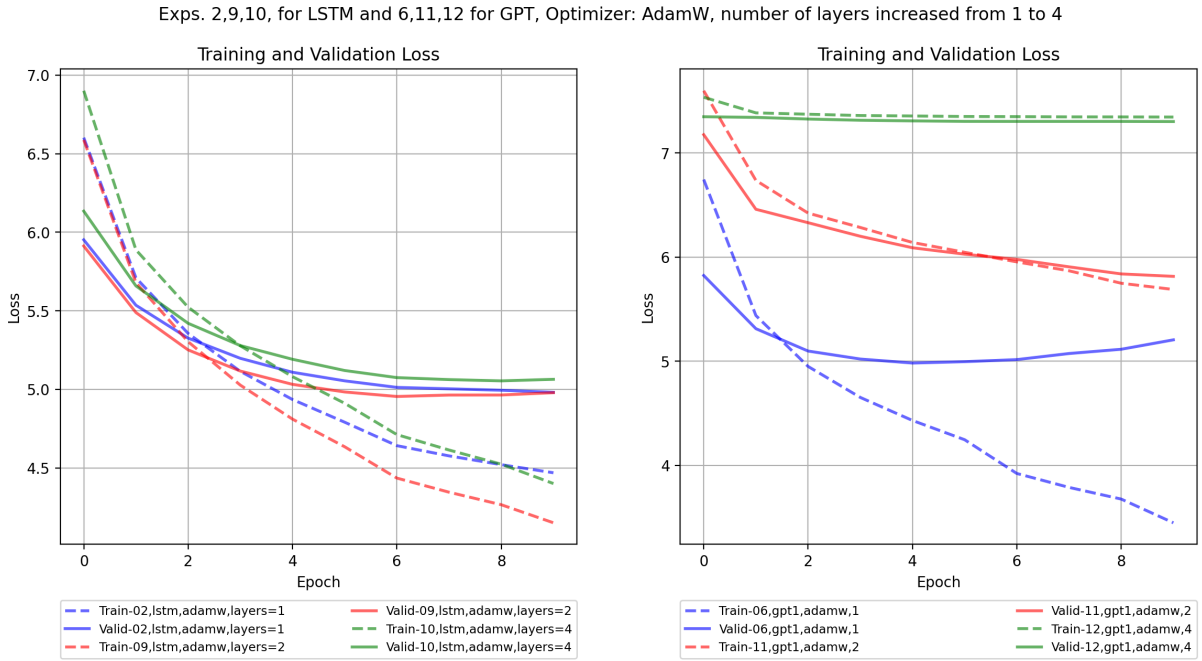

Figure 9: Loss for training and validation set trained for experiments mentioned in problem 3.8 exploring overfitting behavior.