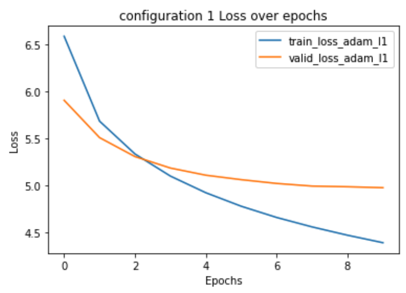
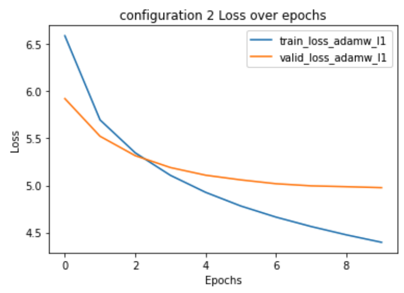
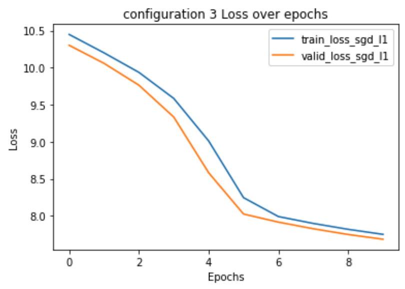
Problem 1

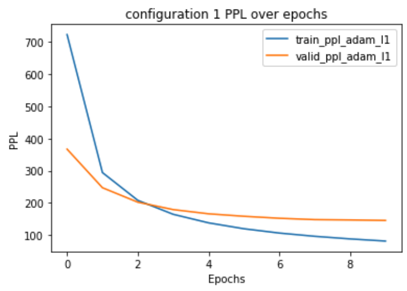
1. Question 3

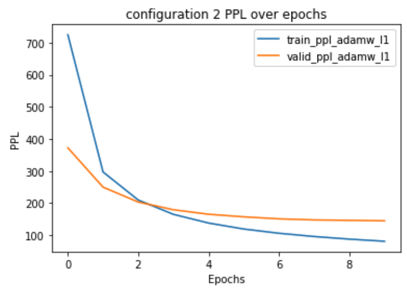
3. The configurations'' plots are as following:

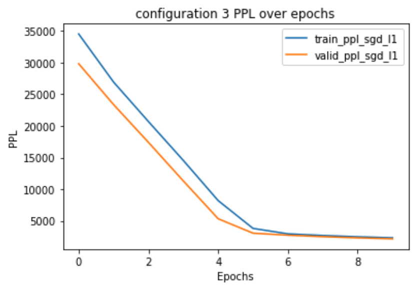


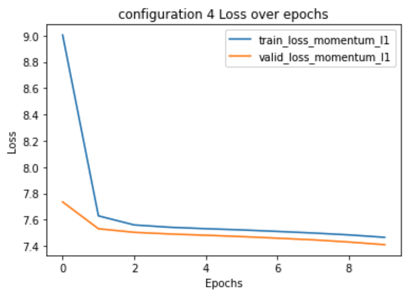


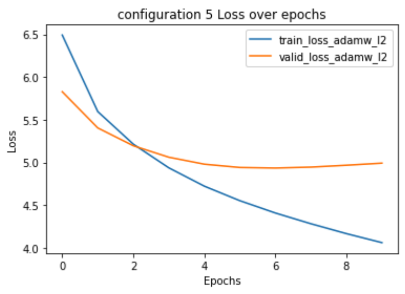


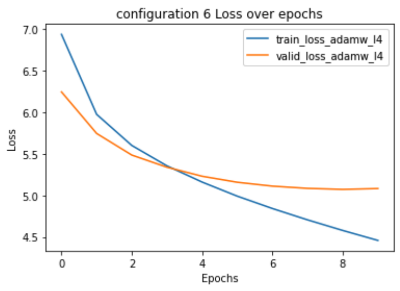


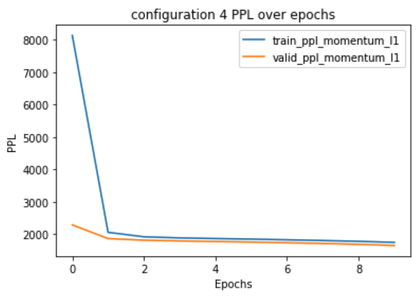


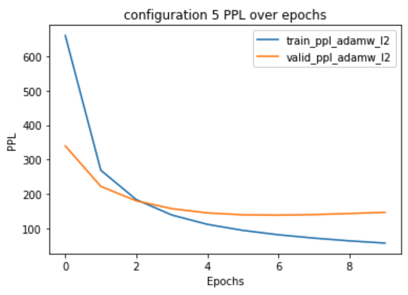


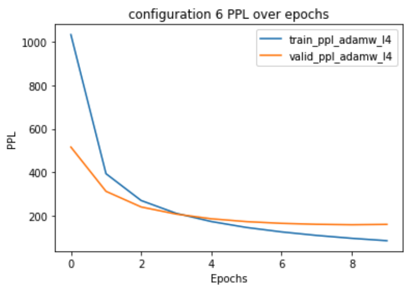












2. Question 4

- Configuration 3 has an average train time of 66.46 and a valid time of 3.08 per epoch. In total, we have 69.54 per epoch, which takes the smallest amount of time among all the configurations. Its hyper-parameters are the following:

    - batch_size=16, layers=4, optimizer='*sgd*', epochs=10, lr=0.001, momentum=0.9, weight decay=0.0005)

  However, while it is the fastest to train and valid, its performance is worse than configuration 1 with optimizer *adam* and the rest of the hypermeter as above, which only takes a slightly more time (69.90s in total) but achieves much better performance.

- Since the lower perplexity and loss indicate better generalization performance, the configuration 1 has achieved the lowest test perplexity (PPL) of 148.53 and a loss of 5.00 with the following hyperparameters and optimizer combinations.
    - batch size=16, layers=4, optimizer='*adam*', epochs=10, lr=0.001, momentum=0.9, weight decay=0.0005

Problem 2

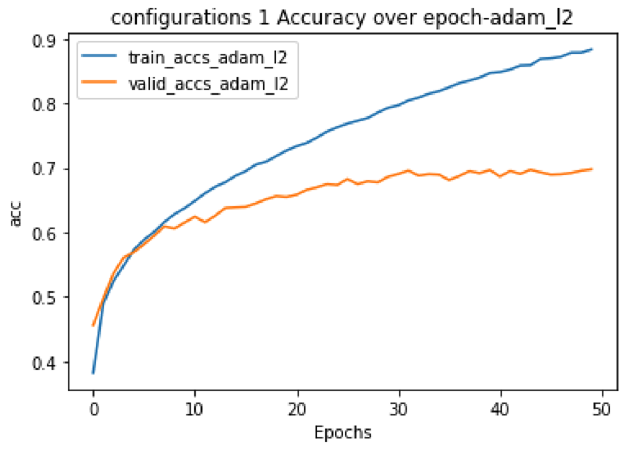
Question 5

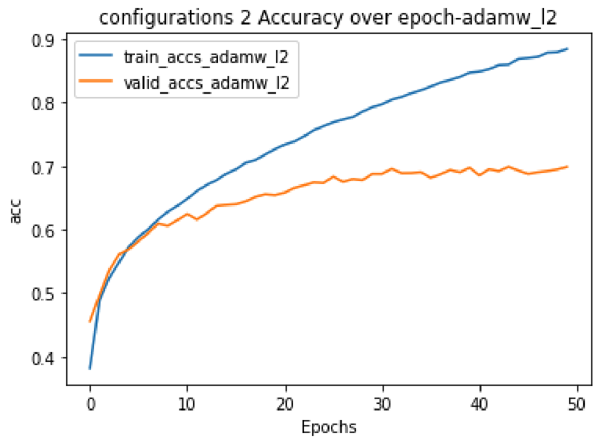
The number of learnable parameters are num_heads*head_size.

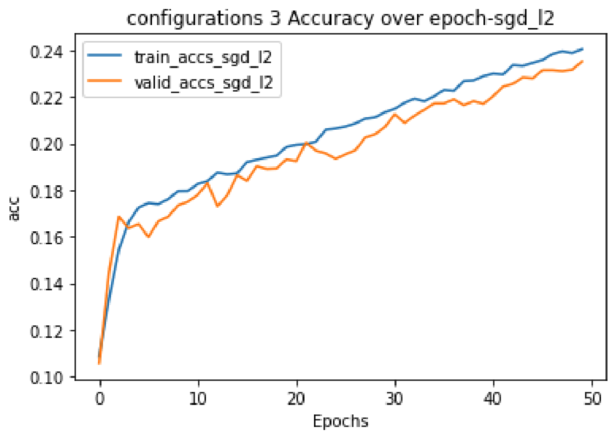
Problem 3

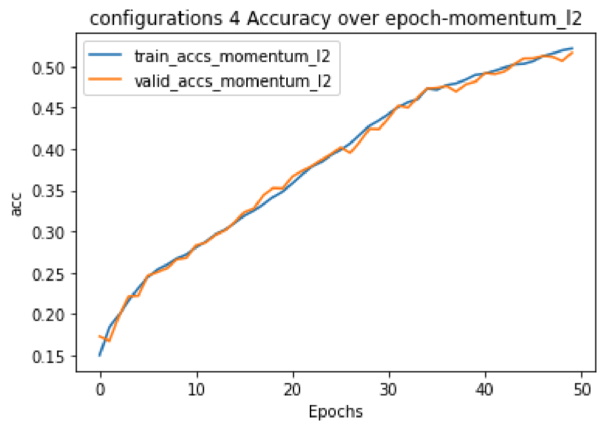
1.

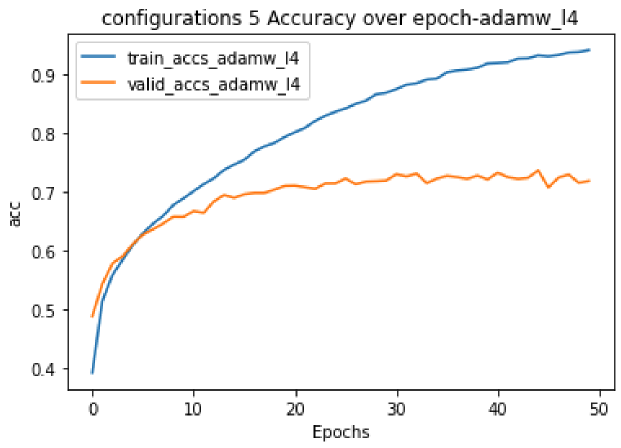
- Note that all the configurations are run with 50 epochs. The configurations' learning curves of accuracy over epochs are as follows:

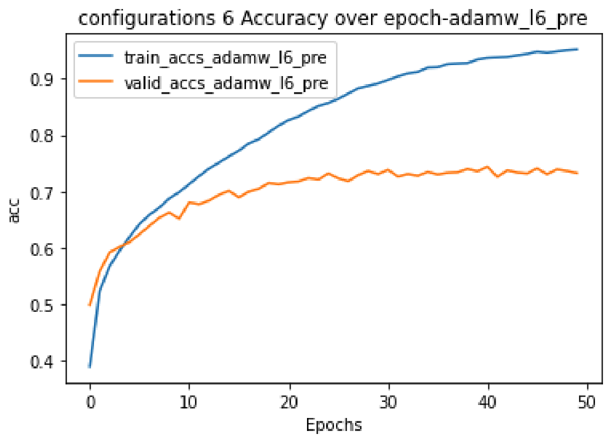


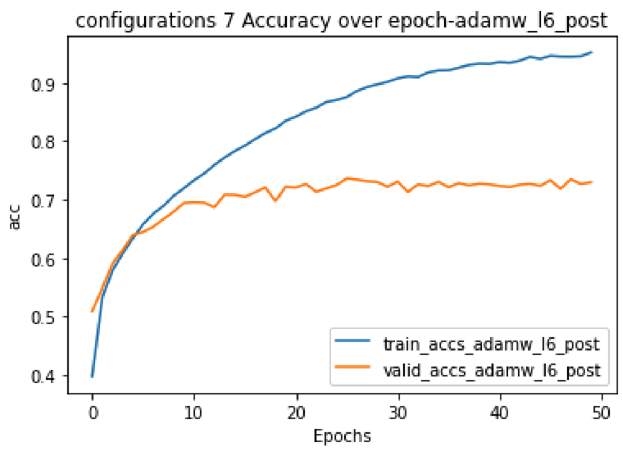




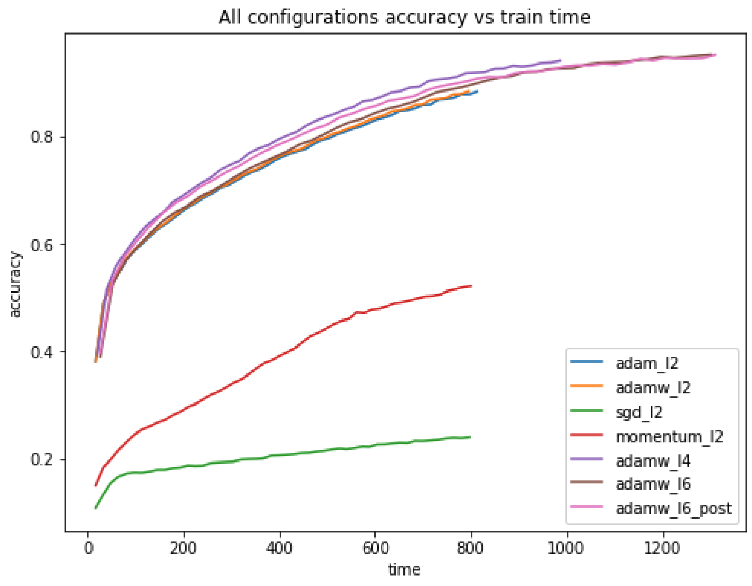


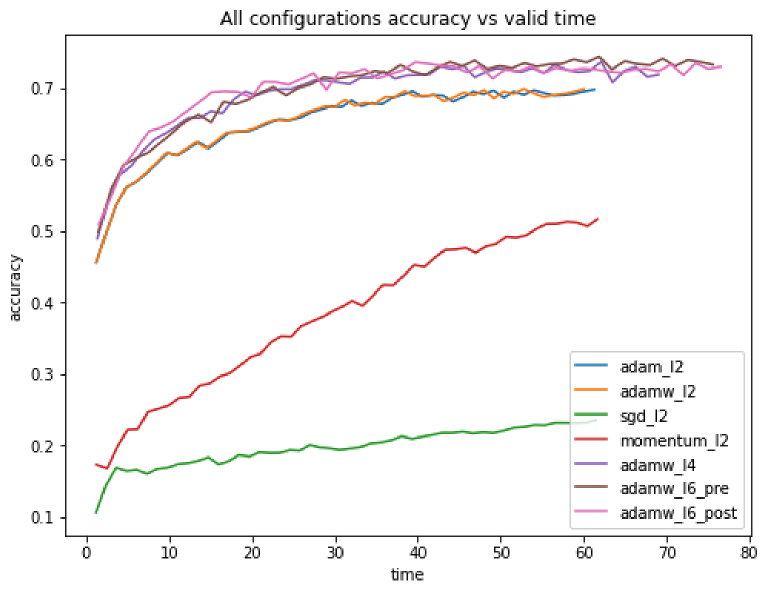






- The configurations' learning curves of accuracy over wall-clock time are as following:





2.

Table: Train and valid performance for each experiment summary

Architecture	**Number of layers**	**Optimizer**	**No. experiment**	**Best Train accuracy**	**Best Valid accuracy**
prenorm	2	adam	1	0.8836	0.6979
prenorm	2	adamw	2	0.8841	0.6988
prenorm	2	sgd	3	0.2405	0.2352
prenorm	2	momentum	4	0.5216	0.5164
prenorm	4	adamw	5	0.9407	0.7365
prenorm	6	adamw	6	**0.9518**	**0.7437**
Postnorm	6	adamw	7	**0.9517**	**0.7361**

3.

- If the most important concern is the wall-clock time, from the plots of the configurations' learning curves of accuracy over wall-clock time in Q1, we can see the short time used at 796.62s of training time and 61.52s of validation time from configuration 3, which are is an *sgd* optimizer with 2 layers. However, the performance of configuration 3 is the worst of all. With a similar amount of time(+14s) and using *adamw* or *adam* as optimizer and layer 2, we can get much better accuracy.
  The hyperparameters for both configurationsare:
    - "batch_size": 128,
    - "layers": 2,
    - "block": "prenorm",
    - "epochs": 50,
    - "optimizer": "adamw",
    - "lr": 0.0003,
    - "momentum": 0.9,
    - "weight_decay": 0.0005

- If the most important concern is the generalization performance, the performance of layer 6 with *adamw* optimizer and *postnorm* structure has the best accuracy of 73.27% in the test set. Hence, it should be used in this case.
  The hyperparameters for the configurations are:
    - "batch_size": 128,
    - "layers": 6,
    - "block": "postnorm",
    - "epochs": 50,
    - "optimizer": "adamw",
    - "lr": 0.0003,
    - "momentum": 0.9,

- "weight_decay": 0.0005

4. The observations are the followings:
   - We can see that the performance of the *sgd* is the worst among the first four experiments, the second worst is the one with momentum optimizer.
   - By introducing *momentum*, the performance improves faster than *sgd.* The *momentum* as an exponential moving average of past gradients, accelerates the process.
   - However, the performance increases more significantly with *adam* or *adamw* as an optimizer. It implies that incorporating *momentum* with the accumulation of squared gradients works to improve the performance.
   - In this experiment, *adam* and *adamw* have very close results. The impact of weight decay is not significant.

5. From the table above, we can see that *Postnorm* performs architecture slightly better than *Prenorm* architecture with the same optimizer and hyperparameters. But in the test set, the performance are 0.7288 of *Prenorm* vs 0.73269 of *Postnorm.* According to the result, there is no preference for which one works better than the other.
6.
   - No, there results are not as my expectation. Ever after training from 10 epochs(default) to 50 epochs, for all the configurations, the maximum valid accuracy is 74.37% with config 6 and the best test accuracy is 73.27% with config 7.
   - Even with five times the number of parameters to learn, the Vit performance is much lower (around 17% accuracy) when compared to the other CNN architectures in the table.
   - One possible reason is that ViTs yield excellent transfer performance only when they are pre-trained on larger datasets. The CIFAR10 dataset only has 60, 000 images. The amount of training for the VIT model is relatively small after splitting into train, valid, and test sets.

7. The increase in layers will increase the learnable parameters, which requires more GPU memory. The following table shows configurations, the corresponding learnable parameters, and GPU memory usage.

Architecture	**Number of layers**	**Optimizer**	**No. experiment**	**Learnable parameters**	**Average GPU memory usage over 10 epochs**
prenorm	2	adam	1	1086730	10%
prenorm	2	adamw	2	1086730	10%
prenorm	2	sgd	3	1086730	10%

prenorm	2	momentum	4	1086730	10%
prenorm	4	adamw	5	2140938	12%
prenorm	6	adamw	6	3195146	15%
Postnorm	6	adamw	7	3195146	15%

8. From the plots in Q1 (all the configuration accuracy over epochs), we can see there is no overfitting behaviour of the various models. There is no sign that a particular class of models overfits more easily than others. For configs 1, 2, 5, 6, 7, we can observe that the validation accuracy remains around 73% even though the training accuracy keeps increasing. For config 3,4, both might need more training epochs to observe whether they have overfitting behavior.

   In the case of overfitting behaviour, a practitioner can take action by gathering more training data, such as using data augmentation methods.