COMS E6998 Introduction to Deep Learning and LLM based Generative AI Systems - HW1

Kevin Tang (kt2942)

September 25, 2024

1 Problem 1

1. Assume the equation for the true relationship is $y(x) = f(x) + \epsilon$, and the equation for the predicted value of y is $\hat{y} = g(x)$. From these two equations, we can write the Mean Squared Error between the true and predicted values as:

$$MSE = \frac{1}{t}\sum_{i=1}^{t}(f(x_i) + \epsilon - g(x_i))^2 = E[(f(x) + \epsilon - g(x))^2]$$

In the above equation, rewrite the notation of Mean Squared Error using expected values, and assume that x is an independent variable. This expression is our starting point to derive the bias-variance decomposition. To get started, we first look at the error term ϵ. This term has the following properties:

$$E[\epsilon] = 0, var(\epsilon) = E[(\epsilon - E[\epsilon])^2] = E[\epsilon^2] = \sigma_\epsilon^2$$

Also, note that the bias and variance is defined as follows:

$$Bias = E[g(x)] - f(x)$$

$$Variance = E[(g(x) - E[g(x)])^2]$$

With these being said, we can derive the bias-variance decomposition from the expression of MSE:

$$\begin{aligned} E[(f(x) + \epsilon - g(x))^2] &= E[(f(x) - g(x))^2] + 2E[(f(x) - g(x))\epsilon] + E[\epsilon^2] \\ &= E[(f(x) - g(x))^2] + 2E[(f(x) - g(x))]E[\epsilon] + E[\epsilon^2] \\ &= E[(f(x) - g(x))^2] + \sigma_\epsilon^2 \end{aligned}$$

Now, we further expand the $E[(f(x) - g(x))^2]$ term:

$$\begin{aligned} E[(f(x) - g(x))^2] &= E[((f(x) - E[g(x)]) - (g(x) - E[g(x)]))^2] \\ &= E[(f(x) - E[g(x)])^2] - 2E[(f(x) - E[g(x)])(g(x) - E[g(x)])] + E[(g(x) - E[g(x)])^2] \\ &= E[(E[g(x)] - f(x))^2] - 2E[f(x) - E[g(x)]]E[(g(x) - E(g(x)))] + E[(g(x) - E[g(x)])^2] \end{aligned}$$

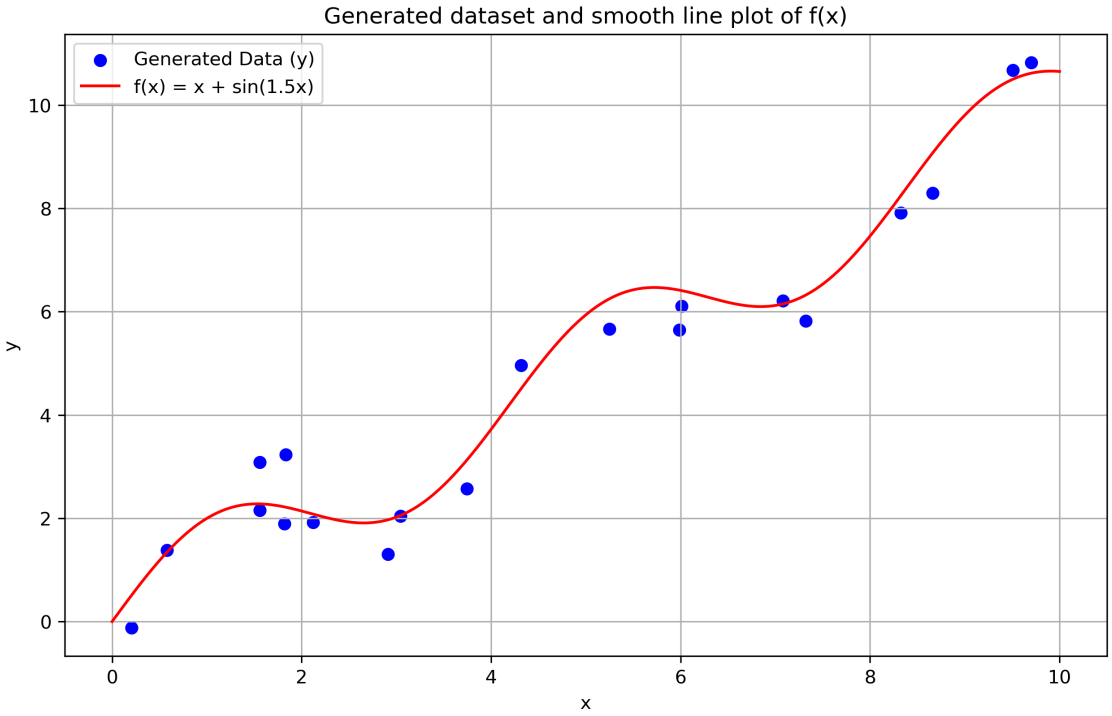
Because $E[(g(x) - E(g(x)))] = 0$, and referring to the equations of bias and variance we have:

$$E[(f(x) - g(x))^2] = Bias^2 + Variance$$

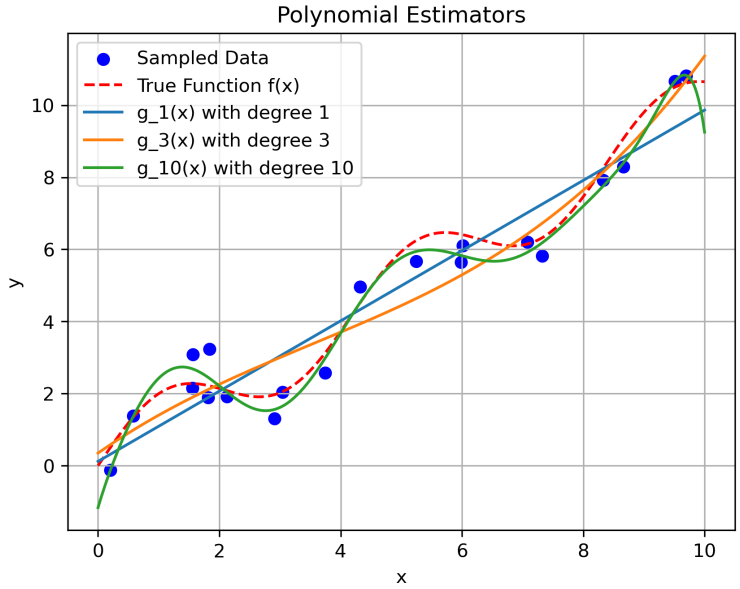
Combining it into the expression of MSE, we now have:

$$\begin{aligned} E[MSE] &= Bias^2 + Variance + \sigma_\epsilon^2 \\ &= Bias^2 + Variance + Noise \end{aligned}$$

2. The generated figure is shown below. All x values are sampled from 0 to 10.

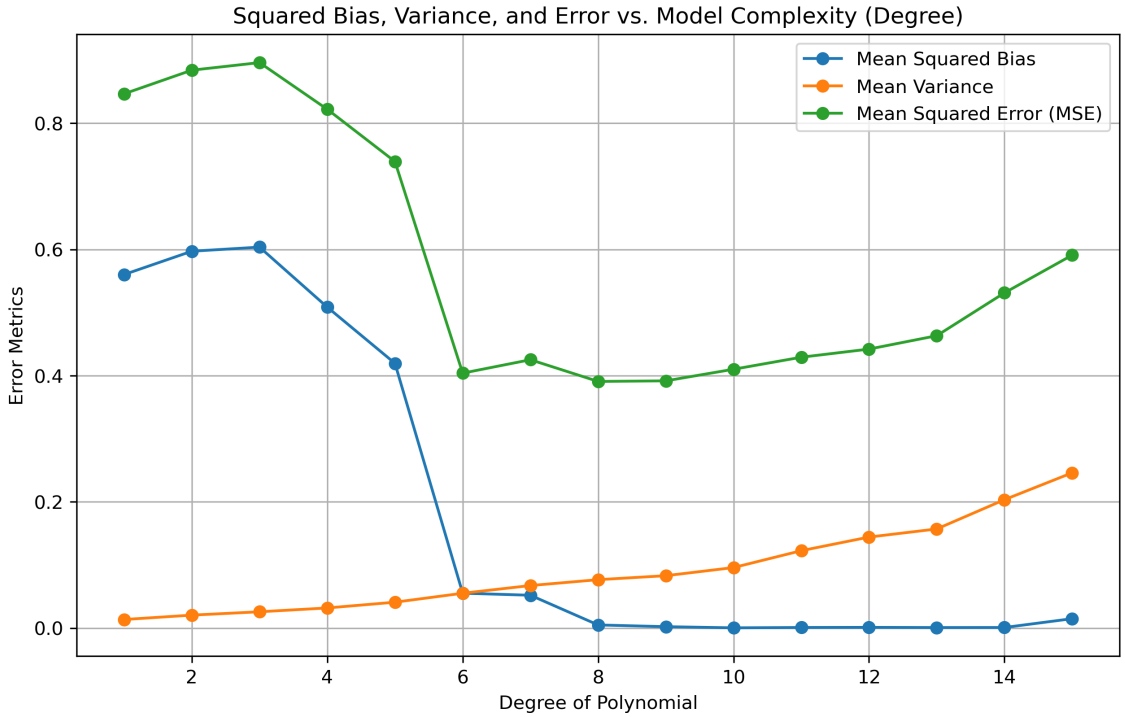


3. The following graph demonstrates the three polynomial estimators $g_1(x)$, $g_3(x)$, and $g_{10}(x)$, along with the sampled data $y(x)$ and the true model $f(x)$:



Compared to the true model, the estimator with $degree = 1$ is clearly underfitting, while the estimator with $degree = 10$ is overfitting. The estimator with $degree = 3$ is still not an ideal representation of the true distribution, even though it is slightly less underfitted than $g_1(x)$.

4. By conducting the experiment mentioned in this question and using the first suggested method to perform data generation, we obtained the following graph representing the trade-off between bias and variance with model complexity:



Based on the graph, model with $degree = 6$ seems to be the best one, as it's keeping both bias and variance relatively low and also has a relatively low MSE.

5. After carrying out the experiment, we obtained the following data:

   Original model squared bias = 0.0004488301878092365, Regularized model squared bias = 0.03226956721264739

   Original model variance = 0.26168794050197275, Regularized model variance = 0.2616879375174942

   Original model MSE = 0.00010082123339512403, Regularized model MSE = 0.032735006180081794

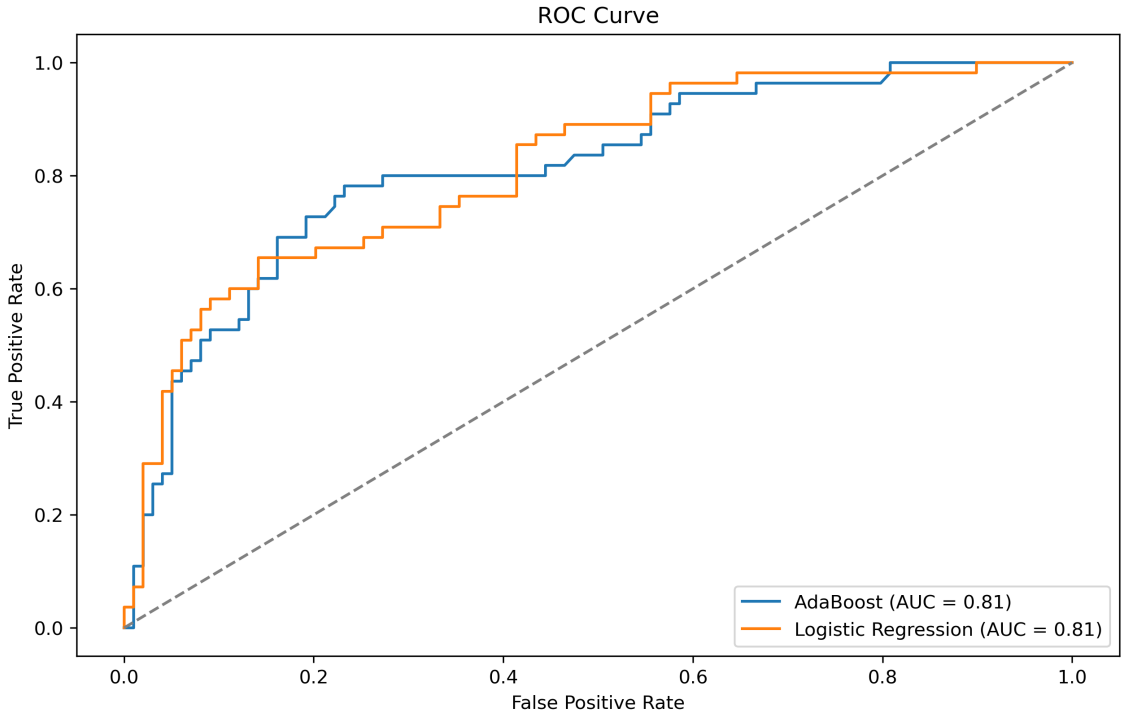
   We can see that the original model has a lower bias compared to the regularized model. This is because L2 regularization reduces the magnitude of model parameters, therefore simplifies the model and increases its bias. Because this regularized model is now more simplified, it cannot represent the distribution as well as the original model and therefore received a higher MSE.

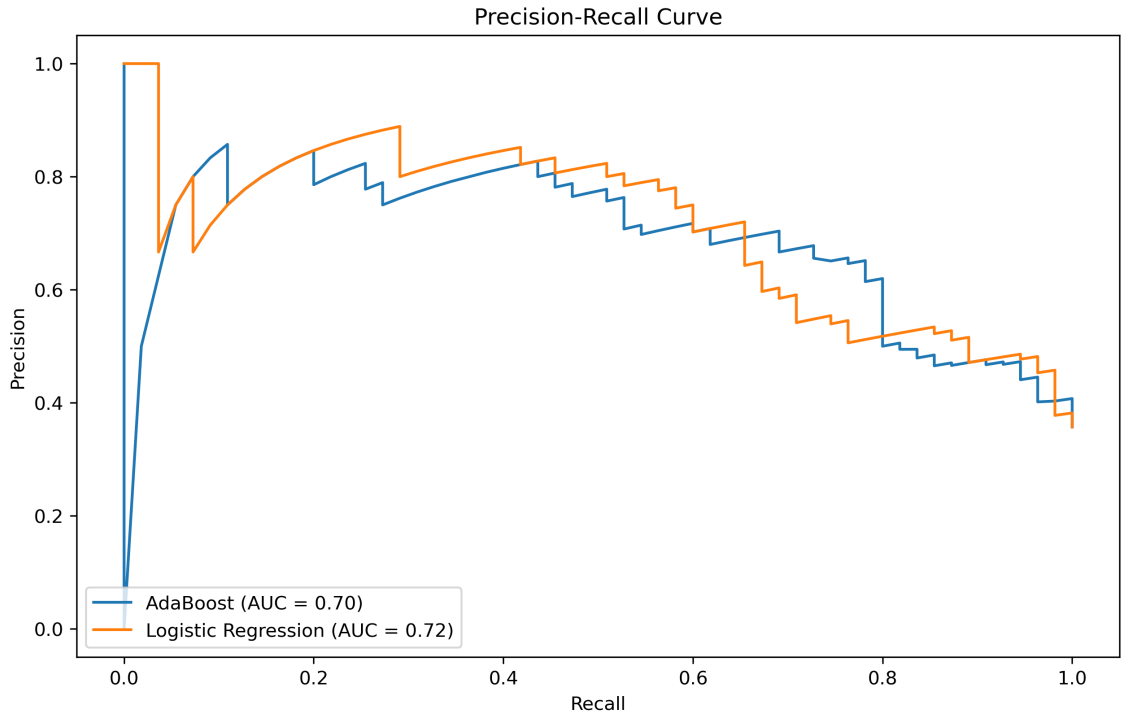
2 Problem 2

1. True negative matter for the ROC curve, because the False Positive Rate is calculated as $FP = \frac{FP}{FP+TN}$, which will be impacted by the number of true negatives in the confusion matrix. However, this value does not impact the Percision-Recall curve, as it does not show up in either of the calculation of percision or recall.

   According to the ICML 2006 paper, each point on the ROC curve corresponds to a specific confusion matrix. Also, each point on the PR curve should also correspond to only one confusion matrix. This is because that even though the number of True Negatives is not specified by percision and recall, when the total amount of samples is fixed, knowing three other values (TP, FP, and FN) will give us the true negative amount. With the help of this correspondence, we and further establish one-to-one relationship between points on the ROC curve and points on the PR curve.

2. We used the diabetes dataset from Open ML (included in submission). The ROC and PR for the two models are shown below:





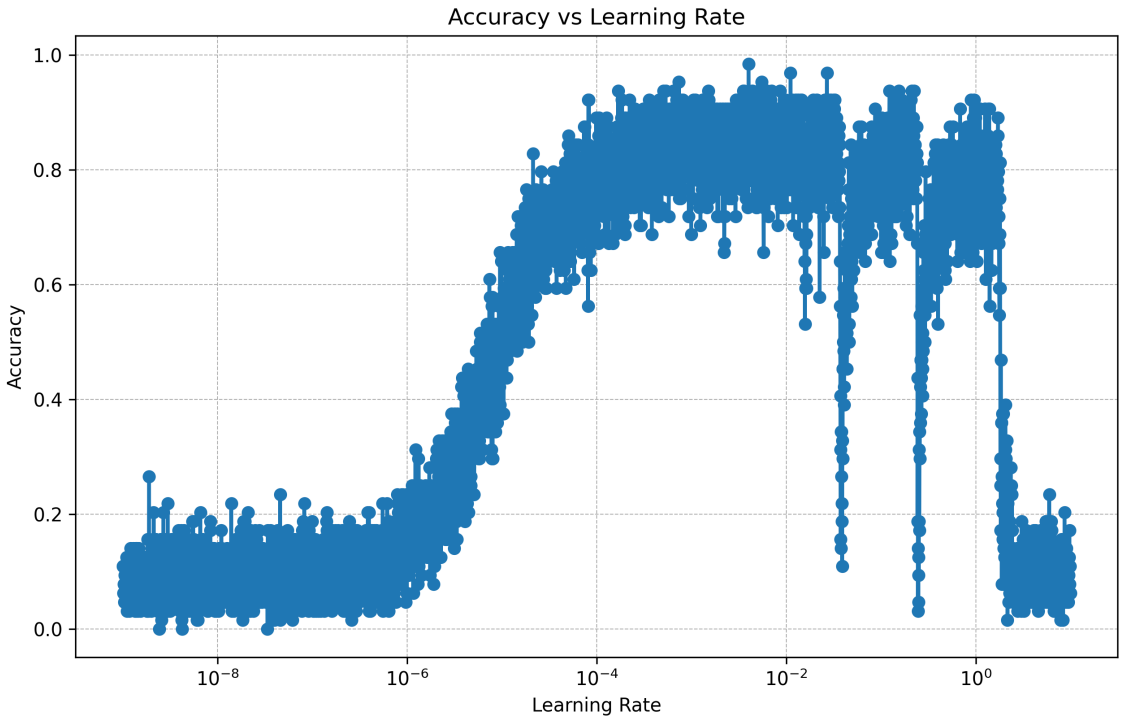
For an all-positive classifier, it will lie at point (1, 1) in the ROC figure. This is because an all-positive classifier classifies all samples as positive. Therefore, True Positive Rate will be 1 because all actual positive samples are predicted as positive. At the same time, False Positive Rate will also be one because all negative samples are classified as positive.

For this specific dataset we are using, there are 99 negative samples and 55 positive samples in the test set. With the all positive classifier, we will get $TP = 55, FP = 99, TN = FN = 0$. With these values, we will get $recall = \frac{TP}{TP+FN} = 1$, and $percision = \frac{TP}{TP+FP} = 0.36$. Therefore, it will locate at point (1, 0.36) in the Percision-Recall figure.

3. The AUROC and AUPR metrics were calculated and shown in the previous question. For AdaBoost model, $AUROC = 0.81$ and $AUPR = 0.7$. For the Logistic Regression model, $AUROC = 0.81$ and $AUPR = 0.72$. We further calculated the AUPRG for AdaBoost as $AUPRG = 0.3798403064460848$. For Logistic Regression, $AUPRG = 0.37224450918910745$. I agree with the paper that we should use PR-Gain curves rather than PR curves, because it allows us to inherently compare our models against the baseline performance and is more indicative in terms of showing how good each classifier is performing.

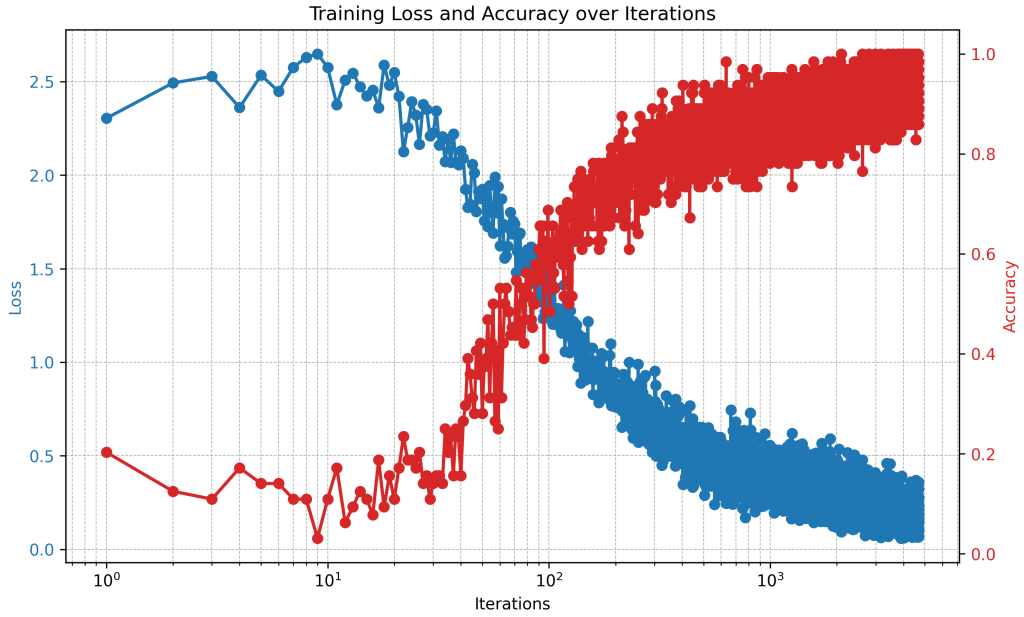
3 Problem 3

1. The following graph shows the change of test accuracy along different learning rates, similar to Figure 3 in the Smith paper.

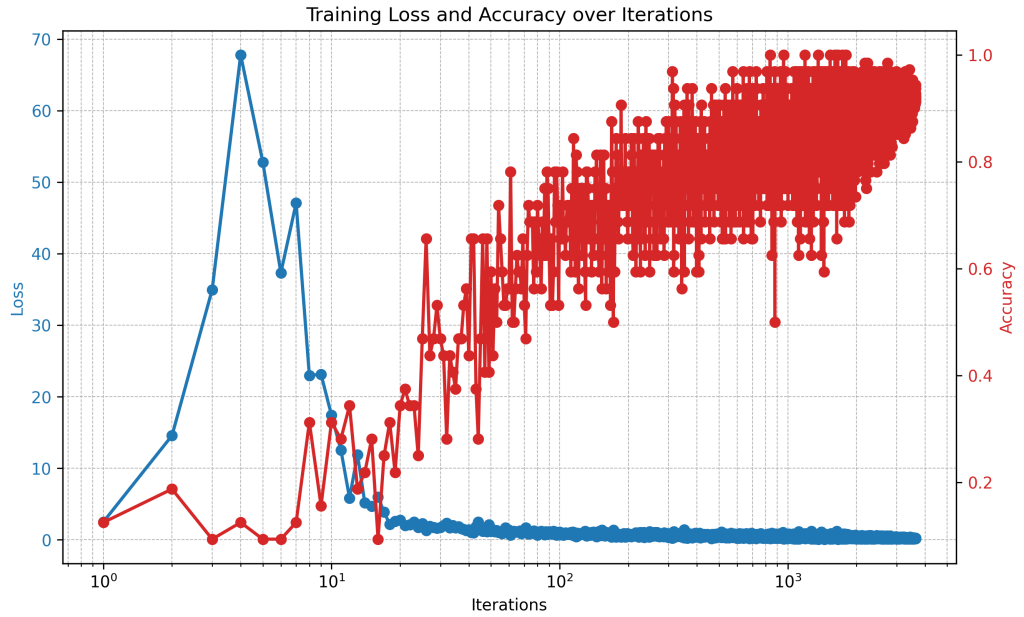


Based on the graph, we can choose $lr_{min} = 10^{-6}$ and $lr_{max} = 10^{-4}$

2. The train loss and accuracy curves over the iterations are shown in the following graph



3. The training loss and accuracy over iterations for this experiment is shown in the following plot:



By observing the two plots, we ca observe that both models were able to reach a very high training accuracy. The decrease in loss is faster than when using cyclical learning rate policy.

4 Problem 4

1. The completed tables is shown in Table 1:

2. (a) The general ideal behide the Inception module is to use fitters of different sizes to better capture the features are multiple scales and improve the network's ability of handling different spatial patterns in the input data. Also, the inception module is more efficient in terms of computation

Layer	Number of Activations (Memory)	Parameters (Compute)
Input	$224 \times 224 \times 3 = 150K$	0
CONV3-64	$224 \times 224 \times 64 = 3.2M$	$(3 \times 3 \times 3) \times 64 = 1,728$
CONV3-64	$224 \times 224 \times 64 = 3.2M$	$(3 \times 3 \times 64) \times 64 = 36,864$
POOL2	$112 \times 112 \times 64 = 800K$	0
CONV3-128	$112 \times 112 \times 128 = 1.6M$	$(3 \times 3 \times 64) \times 128 = 73,728$
CONV3-128	$112 \times 112 \times 128 = 1.6M$	$(3 \times 3 \times 128) \times 128 = 147,456$
POOL2	$56 \times 56 \times 128 = 400K$	0
CONV3-256	$56 \times 56 \times 256 = 800K$	$(3 \times 3 \times 128) \times 256 = 294,912$
CONV3-256	$56 \times 56 \times 256 = 800K$	$(3 \times 3 \times 256) \times 256 = 589,824$
CONV3-256	$56 \times 56 \times 256 = 800K$	$(3 \times 3 \times 256) \times 256 = 589,824$
CONV3-256	$56 \times 56 \times 256 = 800K$	$(3 \times 3 \times 256) \times 256 = 589,824$
POOL2	$28 \times 28 \times 256 = 200K$	0
CONV3-512	$28 \times 28 \times 512 = 400K$	$(3 \times 3 \times 256) \times 512 = 1,179,648$
CONV3-512	$28 \times 28 \times 512 = 400K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$28 \times 28 \times 512 = 400K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$28 \times 28 \times 512 = 400K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2	$14 \times 14 \times 512 = 100K$	0
CONV3-512	$14 \times 14 \times 512 = 100K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$14 \times 14 \times 512 = 100K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$14 \times 14 \times 512 = 100K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$14 \times 14 \times 512 = 100K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2	$7 \times 7 \times 512 = 25K$	0
FC	4096	$4096 \times 7 \times 7 \times 512 = 1.0276 * 10^8$
FC	4096	$4096 \times 4096 = 16,777,216$
FC	1000	$4096 \times 1000 = 4,096,000$
TOTAL	$16.48M$	$1.44 * 10^8$

Table 1: VGG19 memory and weights

because it has less parameters compared to directly stacked CNN layers, such as those used in VGG19.

(b) For the naive version of the Inception module, the output dimension size would be $32 \times 32 \times 672$, as none of the convolution layers or max pooling layer is reducing the side of the input. For the version with dimension reductions, the output dimension would be $32 \times 32 \times 480$. This is because the 1×1 convolution layer after the max pooling layer greatly helps reducing the dimension of the output.

(c) For the naive version of the Inception Module, it performs the following convolution operations:

$$[1 \times 1 conv, 128] : 32 * 32 * 128 * 1 * 1 * 256 = 33.5M$$
$$[3 \times 3 conv, 192] : 32 * 32 * 192 * 3 * 3 * 256 = 452M$$
$$[5 \times 5 conv, 96] : 32 * 32 * 96 * 5 * 5 * 256 = 629M$$

This results in a total of 1114.5 M Convolution Operations for the naive version. For the version with dimension reductions, it performs the following convolution operations:

$$[1 \times 1conv, 128] : 32 * 32 * 128 * 1 * 1 * 256 = 33.6M$$
$$[1 \times 1conv, 128] : 32 * 32 * 128 * 1 * 1 * 256 = 33.6M$$
$$[1 \times 1conv, 32] : 32 * 32 * 32 * 1 * 1 * 256 = 8M$$
$$[3 \times 3conv, 192] : 32 * 32 * 192 * 3 * 3 * 128 = 226M$$
$$[5 \times 5conv, 96] : 32 * 32 * 96 * 5 * 5 * 32 = 78.6M$$
$$[1 \times 1conv, 64] : 32 * 32 * 64 * 1 * 1 * 256 = 17M$$

This results in a total of 397 M convolution operations.

(d) In the naive architecture, the 3×3 and 5×5 convolution layers resulted in too many convolution operations, because they are operating on the original depth of the input with dimension 256. The dimensionality reduction greatly helps reducing the total convolution operations in those layers by adding a 1×1 depthwise convolution first to reduce the depth dimension of the input that will be sent to the 3×3 and 5×5 layers. Such an operation is very effective, resulting in over 60% reduction of the total convolution operations.

5 Problem 5

Assume the initial weight is w_0, the staleness of the weight updates are calculated as the following:

$$w_1 = w_0 + \lambda g[L_1, 1], \text{staleness of } g[L_1, 1] = 0$$
$$w_2 = w_1 + \lambda g[L_1, 2], \text{staleness of } g[L_1, 2] = 0$$
$$w_3 = w_2 + \lambda g[L_2, 1], \text{staleness of } g[L_2, 1] = 2$$
$$w_4 = w_3 + \lambda g[L_1, 3], \text{staleness of } g[L_1, 3] = 1$$
$$w_5 = w_4 + \lambda g[L_1, 4], \text{staleness of } g[L_1, 4] = 0$$
$$w_6 = w_5 + \lambda g[L_2, 2], \text{staleness of } g[L_2, 2] = 2$$

6 Code Submission

GitHub Repository Link: https://github.com/KevinTang2318/COMS6998_015_HW1