Homework 1 – Report

Efekan Çakmak – 2309797

Sanity Check

At the beginning, we have random weights in our Artificial Neural Network. Thus, when we put our images in this Network it will make random predictions.

Theoretic probability of labeling an image correctly without looking at an image, in CIFAR10 set, is %10 since there are 10 labels and they are uniformly distributed (1/p = 1/10).

Therefore, I expect the accuracy approximately at 0.10.

To predict loss, I tried to use Cross Entropy function by hand and by putting k = 10, label amount.

We have Cross Entropy Loss: CEL = $-\frac{1}{k} + \sum_{n=1}^{k} \left(\log\left(\frac{1}{k}\right)\right) = \log(k)$.

$$\log(10) = 2.3025$$

Thus, I expect the loss from this untrained ANN at 2.30.

After executed sanity.py, the python code, we can see that loss is equal to 2.3041, and accuracy is equal to 0.1002

Consequently, predicted and computed are too similar in the untrained ANN.

The reason is that my prediction is on randomly labeling without any computation or thinking.

In the beginning, untrained ANN has random weights for features. It means that it does not learn anything and give predictions based on randomized weights.

Validation Set from Training Set

During the training phase, we should be careful about when training halts since more and more training will yield 'memorization'. Therefore, we use a part of training set, namely validation set, in order to check loss of model at the end of each epoch.

I used the same set split with recitation code:

%80 of the training data is to be used for training,

%20 of the training data is to be used for validation.

As below,

```
train_set_length = int(0.8 * len(train_set))
val_set_length = len(train_set) - train_set_length
train_set, val_set = random_split(train_set, [train_set_length, val_set_length])
```

Hyper-Parameter Configurations

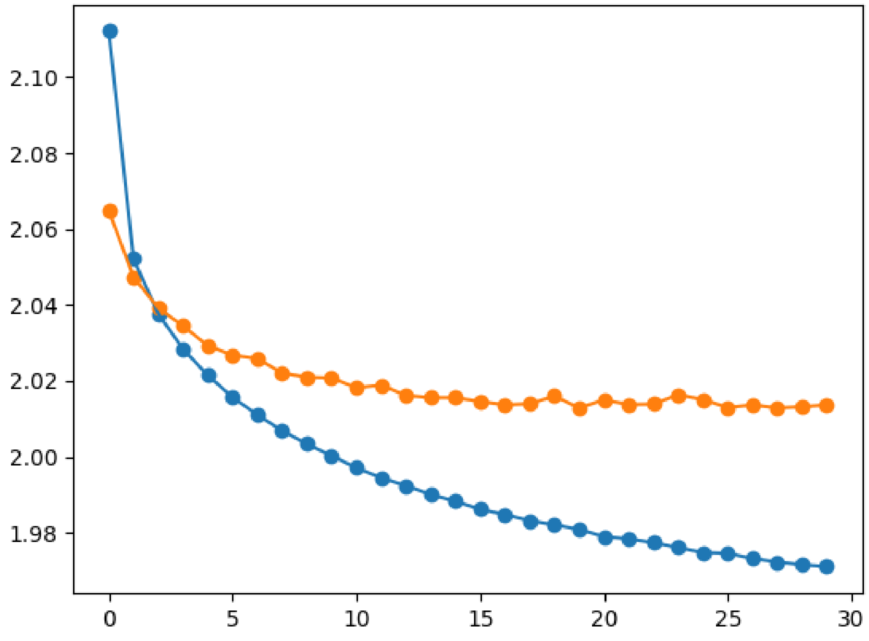
I used stable batch-size 64 in all models.
I select 30 for maximum epoch number. I do early-stop by looking average of validation loss.
And, all my configurations use 4096 as length of first layer, 2048 as length of the second layer.

Configuration Number	Learning Rate	Activation Functions	Number of Layers	Validation Accuracy	Validation Loss	Stop Epoch
1	0.001	-	1	27.880%	2.0556	6
2	0.001	relu	2	43.530%	1.8812	7
3	0.001	tanh	2	36.700%	1.9565	10
4	0.001	hardswish	2	43.130%	1.7738	7
5	0.001	relu->relu	3	43.920%	1.7581	7
6	0.001	relu->tanh	3	39.960%	1.8714	7
7	0.001	relu->hardswish	3	42.430%	1.7804	6
8	0.001	tanh->tanh	3	36.460%	1.9487	14
9	0.001	tanh->relu	3	43.900%	1.7637	8
10	0.001	hardswish->hardswish	3	42.240%	1.8981	6
11	0.001	hardswish->relu	3	42.300%	1.8828	6
12	0.001	hardswish->tanh	3	37.360%	1.8364	6
13	0.0002	-	1	28.800%	2.0233	23
14	0.0002	relu	2	45.420%	1.6435	9
15	0.0002	tanh	2	42.200%	1.7712	29
16	0.0002	hardswish	2	45.990%	1.6707	20
17	0.0002	relu->relu	3	46.700%	1.7107	7
18	0.0002	relu->tanh	3	47.750%	1.6409	7
19	0.0002	relu->hardswish	3	48.030%	1.6053	6
20	0.0002	tanh->tanh	3	44.910%	1.7416	18
21	0.0002	tanh->relu	3	45.520%	1.7171	7
22	0.0002	hardswish->hardswish	3	47.510%	1.6417	8
23	0.0002	hardswish->relu	3	47.050%	1.6421	7
24	0.0002	hardswish->tanh	3	46.900%	1.6752	15
25	0.0001	-	1	29.660%	2.0164	29
26	0.0001	relu	2	47.140%	1.6263	19
27	0.0001	tanh	2	41.870%	1.7045	29
28	0.0001	hardswish	2	45.640%	1.6505	29
29	0.0001	relu->relu	3	47.850%	1.6508	8
30	0.0001	relu->tanh	3	46.120%	1.6436	6
31	0.0001	relu->hardswish	3	47.280%	1.6371	8
32	0.0001	tanh->tanh	3	46.230%	1.6795	27
33	0.0001	tanh->relu	3	47.020%	1.6194	10
34	0.0001	hardswish->hardswish	3	46.650%	1.6487	14
35	0.0001	hardswish->relu	3	47.330%	1.6396	12
36	0.0001	hardswish->tanh	3	47.110%	1.6480	22

BEST K-LAYER NETWORKS

In all of the graphs, blue dots are representing test losses while orange ones show validation losses. In addition, Y-Axis indicates loss whereas X-Axis indicates epoch.

Around 1-Layer Networks, the best performing configuration is 25th configuration.
This configuration uses
Learning rate : 0.0001
Epochs : 29
And results in
Validation Accuracy = 29.660%
Validation Loss = 2.0164



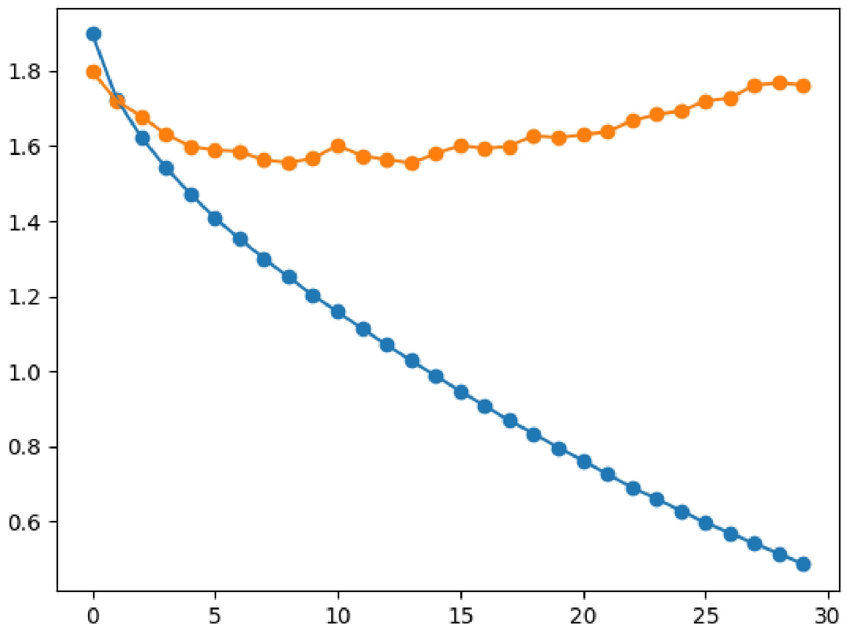
After epoch 15, validation loss seems like oscillating and does not change significantly. Actually, my classifier should have stopped the training at epoch 15. Continuing caused to overfitting.

Around 2-Layer Networks, the best performing configuration is 26th configuration.
This configuration uses

Activation Function: relu
Learning rate: 0.0001
Epochs: 19

And results in

Validation Accuracy = 47.140%
Validation Loss = 1.6263



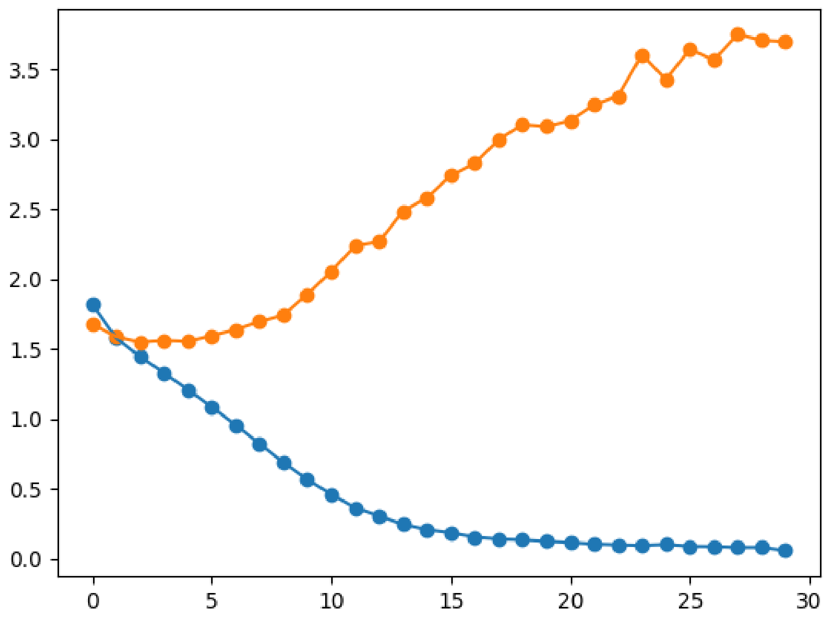
After epoch 16, validation loss increases slightly at each epoch. My classifier stopped at epoch 19. Stopping at epoch 19 is not optimal since there are ones that have less validation loss. However, my classifier detected significant increase on validation loss and stopped.

Around 3-Layer Networks, the best performing configuration is 19th configuration.
This configuration uses

Activation Function: relu for first layer, hardswish for second layer.
Learning rate: 0.0002
Epochs: 6

And results in

Validation Accuracy = 48.030%
Validation Loss = 1.6053



After epoch 5, validation loss significantly increases. My classifier stopped at epoch 6 so that we did not train unnecessarily and prevented from over-fitting.

- **What countermeasures did I take against overfitting?**
- **What method did I use to decide where to end the training procedure?**

I think that they are the same question because I need to end training procedure to avoid from overfitting.

I dealt with over-fitting by stopping training early.
Stop Condition:
I hold average of validation losses. When last computed loss is higher than average of old ones, I stop the training.

- **How may one understand when a network starts to overfit during training?**

Over-fitting results from increasing of validation loss as long as training continues.
When train error stay decreasing and test error increases, training becomes 'memorizing'. Thereby, we need to stop when over-fitting detected.

- **Test accuracies of best K-Layer networks:**

For configuration 25, the best 1-Layer network, test accuracy is 29.880%.
For configuration 26, the best 2-Layer network, test accuracy is 46.140%.
For configuration 19, the best 3-Layer network, test accuracy is 47.650%.

Discussion on accuracy metric

First, I understood that we certainly should not use train data to calculate accuracy. It lies us. We should score our models on new inputs.

In this data set, accuracy is suitable metric since labels are uniformly distributed. It is too important for balance issue.

If classes are not balanced, accuracy can mislead us. For example, there are 100 people and just one of them is infected by COVID. If we create classifier, just labeling all inputs as NO COVID, we will get %99 accuracy since there is unbalanced classes. In this case, we need to use another metrics like F1 score, confusion matrix etc.

In our case, since our classes are balanced, accuracy is enough to provide suitable results.

Learning Rate

1) Small Learning Rate

Pros:

Since out steps are small, we probably do not miss optimum points.

Cons:

Since out steps are small, we need more iterations and epochs to terminate. This can be observed from my result of configurations:

Average epochs of configurations with LR 0.001: 7.5
Average epochs of configurations with LR 0.0002: 13
Average epochs of configurations with LR 0.0001: 17.58

2) Big Learning Rate:

Pros:

Since out steps are big, we need less epochs or iterations. This can be observed from my result of configurations that I show above.

Cons:

Since our steps are big. we can miss optimum points.

Batch Size

There are three types of gradient decent.

1. Batch Gradient Decent: In this type, the batch size is equal to train set size. It means that we update weights after all train set is processed.
2. Stochastic Gradient Decent: In this type, the batch size is equal to one. It means that we update weights after one sample is processed.
3. Mini-Batch Gradient Decent: This type is between Batch and Stochastic Gradient Decent. Simply Batch size is between 1 and training set size.

I did not use different batch size on my classifiers, thereby I write theoretical responses.

3) Small Batch Size:

Pros:

It converges faster when comparing with big batch size in equal epochs.
It can escape local minima more possibly.

Cons:

We need to shuffle samples in order to achieve actual ‘randomness’.
It often gives results “close” to optimal, does not gives the optimal.

4) Big Batch Size:

Pros:

More deterministic.
It converges faster when comparing with big batch size in equal epochs.
We do not have to shuffle samples since we will cover them all

Cons:

It may get into local minima.
It converges slower when comparing with small batch size in equal epochs.