BLG 506E COMPUTER VISION

ASSIGNMENT 3
Two Layer Neural Networks

Student No: 504221511

Name & Surname: Enis Teper

1. **Download assignment 1 from Stanford's CS231n: http://cs231n.github.io/assignments2019/assignment1/ You will be following two_layer_net.ipynb notebook.**

Since I have downloaded the code base from previous homework, I did not download it again. The code base eliminates the workload of writing code from scratch for required algorithms. It just leaves empty most important cases to let us improve ourselves.

2. **Implement the first part of TwoLayerNet.loss in cs231n/classifiers/neural_net.py where you perform forward pass using weights and biases and compute the scores. Then compare your scores within the cell.**

With the forward pass which means calculating output data with respect to our input, weights and biases outputs are calculated. At first step of forward step inputs are multiplied with first weights and biases are added to their results. In order to achieve non-linearity ReLU activation function is implemented. Non-linearity is an important concept for classifiers. Since real world is complex non-linear activation functions help us to gain more information from the input. Afterwards, the result is multiplied with following weights and biases of second weights are added. Since weights and biases are just initialized biases are zero and weights are values around 1e-4.

3. **In the same function, compute the softmax loss with regularization. Then compare to correct loss within the cell.**

At first to scale result loss num train is calculated by the input. In order to implement softmax function, exponential of each value is calculated. Then, in order to calculate probabilities of each class all exponential values are divided by the sum of exponentials. However, even all values are divided only correct classes will be considered. With the purpose of considering only correct classes index values from 0 to input number is generated. Afterwards, loss is calculated by the function below. All probabilities of correct classes are fed into -log() to finalize the loss. $Loss = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{s_{yi}}}{\sum_j e^j}\right) + \sum_j W_j^2 + \sum_k W_k^2$

Here W_j and W_k represents weight layers of our neural network. Also, implemented regularization method is L2 regularization.

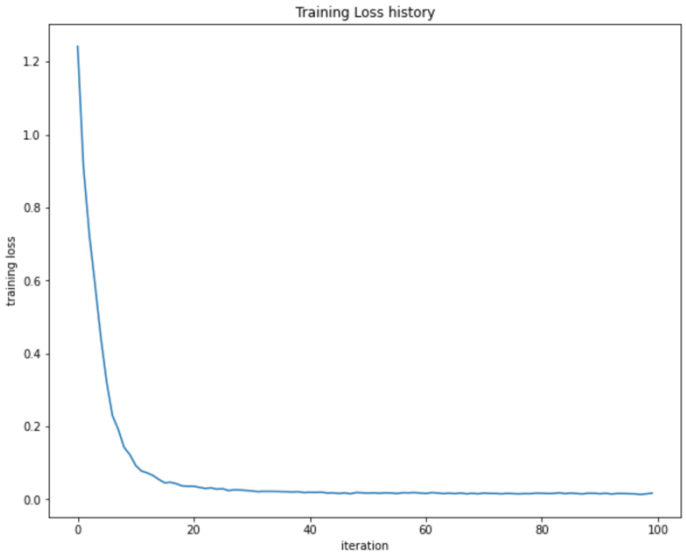
4. **Complete the rest of the function where you compute the gradients with respect to W1, W2, b1 and b2. Then check the relative errors within the cell.**

In order to calculate gradients backpropogation method is used. With this method we take derivative of each input with respect to output. To calculate weights that are not on the latest layer chain rule is used. At first latest layer gradient values are calculated by subtracting 1 from all correct classes. Then, transpose of first pass is multiplied with results and scaled by train number. Meanwhile, sum of all values of each row is assigned to relevant bias value and scaled by train number. Lastly, derivative of regularization is added to weight gradients and results are stored.

With the aim of calcualting gradients of weight_1 and bias_1 class probabilities are multiplied with the transpose of weights of the latest layer. Then, to prevent the the value below zero for that scores as 0 at forward pass, 0 assignment is done. Then, finally gradients of weight_1 and bias_1 are calculated and scaled. Finally, derivative of regularization is added to weight gradients and results are stored.

5. **Implement the TwoLayerNet.train function. You will create random minibatches and update network parameters within the training loop.**

Random batches are generated by selecting inputs randomly with respect to batch size. Thanks to this method we do not have to load whole batch to the system at once. From the forward passes outputs are calculated and from backpropogation weights and biases are updated. As can be seen at Figure 1 our loss value decreases at each iteration until the local or global minimum.



6. **Now implement the TwoLayerNet.predict function. You should run a forward pass and obtain predicted labels using scores.**

Similar as the section of the loss calculation forwad pass is done by:

1. Multiplying inputs with weights of first layer

2. Adding biases of first layer
3. Using ReLU activation function to implement non-linearity into classifier
4. Multiplying results of first layer with second layer weights
5. Adding biases of second layer
6. Obtaining indexes of highest values from each row for prediction of each row.

Only difference is 6th row which is obtaining indexes of highet values from each row step. Since we are not calculating gradients and not implementing backpropogation we only need index of highestt value of each row. Even though, implementing softmax function won't change the result of our prediction only scalar value would be change to probability.

7. Congrats, you have AI awoken! First test it on toy data. Then load CIFAR10 dataset, and unleash it!

Cifar-10 dataset is the dataset that we have used on previous homework while implementing linear classifiers. With respect to results seems like our neural network is not achieving better than linear classifiers.

8. Wait, not a good accuracy for an AI, right? Diagnose it by plotting loss and accuracy trends, and visualize the weights.

Even though our loss value does not lower a lot with respect to initial loss value it seems our model learns a bit. Thus, it achieves 0.28 accuracy which is low when compared to linear classifiers. The plot is given at Figure 1 and from the plot train and validation accuracy increases simultaneously. This may be the cause of our network is too basic for this dataset and classes.

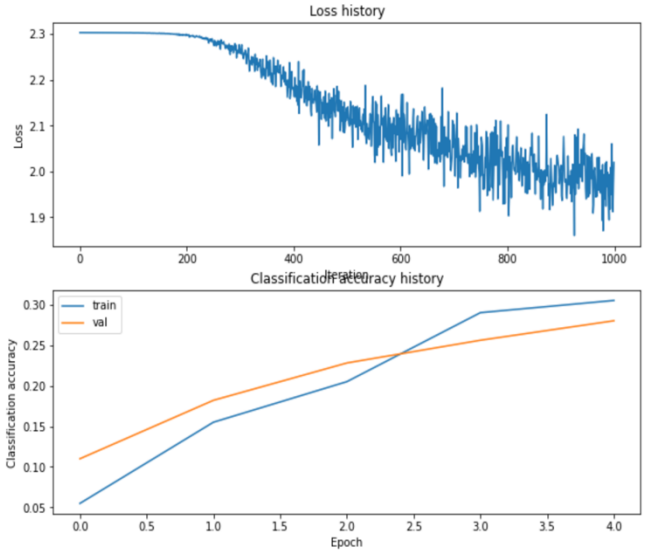


Figure 1: Loss and Accuracy Plot

Even though there might seem to be no overfitting from visualized weights we understand that our model overfitted on car template with respect to other classes on first weights. The templates are given at Figure 2.

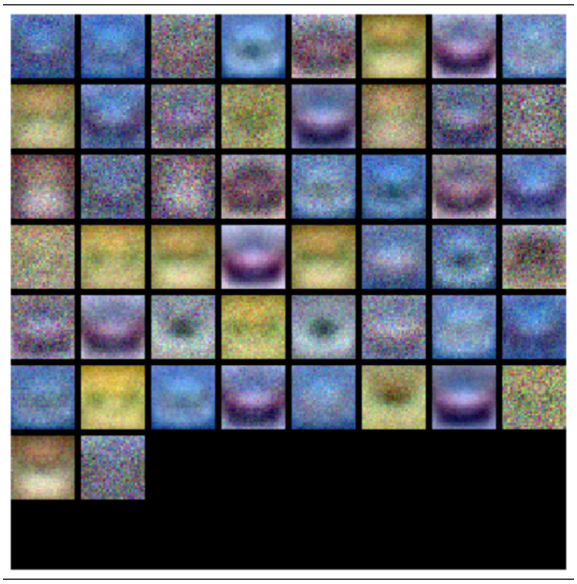


Figure 2: Templates of First Weights

9. Some things prevent your model learning the inputs. Release your AI by tuning hyperparameters on the validation set to acquire better accuracy. Explain what you have done clearly. In real world people do this for months but luckily we have a small dataset.

Hyperparameter tuning may consume a lot of time with respect to combination number of parameters. To achieve best accuracy I have assigned different parameters to the model. Then results are evaluated and best model is selected. The list of parameters that I have tried all combinations is given below:

1. Learning Rate → [1e-3, 1e-4]
2. Learning Rate Decay → [0.98, 0.95]
3. Regression parameter → [1, 0.01]
4. Iteration Number → [1000, 2000]
5. Hidden Size → [100, 150]

Since the combination grows exponentionally with respect to input parameter combination only 2 parameter is selected for each hyperparameter. Batch size may also be implemented here but adding 2 more possibilities means multiplying combinations with 2 and because of iteration number is high my computational power may not good enough to compute all these possibilities in proper time. By trying all these combinations best validation accuracy (0.512) is obtained with:

1. Learning Rate → 1e-3
2. Learning Rate Decay → 0.95
3. Regularization → 1e-2
4. Iteration Number → 2000
5. Hidden Size → 150

10. Visualize the weights of your final model and get accuracy on the test set. Answer the inline question.

By visualizing weights we can see more variant templates have been generated. Test result of our model is 0.51 which is too similar with validation accuracy. This may also express why our model is better than the recent one. Templates of weights are given at Figure 3.

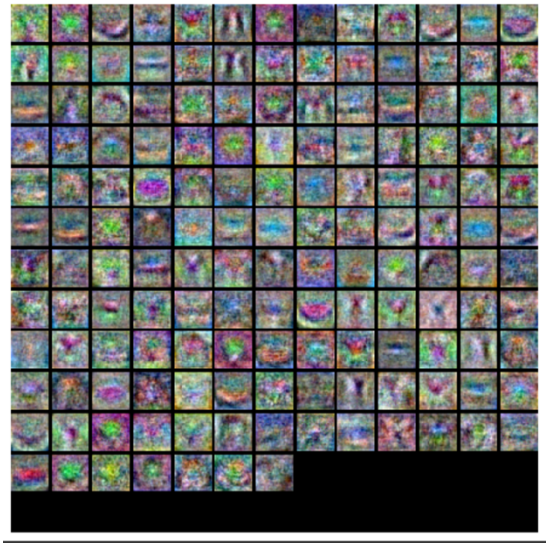


Figure 3: Weight Visualization

****Inline Question** Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.**

1. **Train on a larger dataset.**
2. **Add more hidden units.**
3. **Increase the regularization strength.**
4. **None of the above.**

Your Answer: 1 and 3

Your Explanation: If the training accuracy is high and test accuracy is low this means our model overfits on training data. To avoid it we may feed more data to our model. Thus, variation of input data will increase and model will not be able to memorize all of them. Adding more hidden units means to making model is more complex which is also mean model will have more capacity to memorize input data. For this reason, it is not suggested to adding more hidden units when model overfits the data. Additionally, regression parameter may be increased to spread out the values of weights and not let one to overwhelm others. Thus, we consider all pixels more equally with respect to overfitted weights.