Report: CS776A Assignment 1

> *Submitted on : 03/02/2022, by Gurbaaz Singh Nandra (190349)*

Table of Contents

1. Models

The architecture of the MLP model is a 512 sized input layer, followed by a single hidden layer with 64 neurons and `ReLu` activation function, and finally an output layer with 10 neurons and a `softmax` activation function (since it is a multiclass classification problem, `softma` is an ideal choice). The loss function is `cross-entropy` (since it is a classification problem).

In the code, the model has been represented by a python `class` named `MLP`, which takes in different parameters like size of input features, hidden layer features and output labels, learning rate etc to instantiate. For training, first of all an object of class `MLP`, eg, `mlp` is created. The weights get initialised too. Then its method `train()` is called, which requires input features `X`, hot encoded labels `y` and number of `epochs`. In each iteration in an epoch, class methods `forward()` and `backprop()` are called to update the model parameters. Finally, predictions are run using `mlp.predict()` which takes in input features `X_test` and outputs the predicted labels from `0-9`.

2. Hyperparameters

1. **Learning rate**: Different learning rates have been tried and experimented, where a lr of `1e-2` was a bit too fast, and `1e-4` a bit too slow. Finally learning rate of `1e-3` proved to be the best tradeoff.
2. **Batch Size**: The models have been trained in an `SGD` fashion, hence the batch size is kept as `1`.
3. **Epochs**: It has been observed that there is no significant change in model performance after around 15 epochs (note that as SGD hasa been used, `50000` times parameter updates occur in each epoch), hence the model that has been finally evaluated on has been trained on `20` epochs.

3. Evaluation metrics

After each epoch, average loss of model has been computed using `cross-entropy` loss. As epochs increase, loss should ideally decrease gradually. After the training has been completed, model accuracy on training data has been evaluated. A good accuracy on training-set affirms that the model has learned from the given dataset. Although, a very high accuracy (>90%) is a sign of overfitting. Finally, the model accuracy on (unseen) test dataset has been evaluated. To calculate accuracy, model runs `forward()` on given input features and outputs a 10 length vector. Then using `np.argmax()`, the index corresponding to maximum value in the vector is reported as output label(`0-9`).

4. Instructions for running the models

i. Downloading and extracting CIFAR Dataset

1. Download the dataset using `wget` or `curl` (or manual download from the internet GUI)

   ```
   wget https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
   ```

2. Extract the content with `tar` using the following command

   ```
   tar zxvf cifar-10-python.tar.gz
   ```

This will generate a directory named `cifar-10-batches-py` containing data batches.

ii. Running the code

Make sure you have `python` and `pip` / `conda` (package managers for `python`) installed.

1. Install the required dependencies,

   ```
   pip install -r requirements.txt
   ```

2. Run the `dataset.py` script. This will run image transformations and generate augmented training set.

   ```
   python dataset.py
   ```

3. Post execution, you will see 3 pickled data batches in your current directory, named `unaugmented_dataset`, `augmented_dataset` and `test_dataset`. These contain labels and numpy image array (of size 3x32x32) data of normal training set(50000 samples), augmented training set(50000 samples) and test set(10000 samples) respectively. These will be loaded and used by our next script contaning the implementation, training and testing of our MLP model.

   ```
   python model.py
   ```

   If you want to use the provided model weights in `.npy` files and skip the training part, pass the `--no-train` flag.

   ```
   python model.py --no-train
   ```

5. Performance Comparison

After `20` epochs, for MLP model trained on unaugmented dataset, train-set accuracy was `83.26%` and test-set accuracy was `79.03%`. After `10` epochs, for MLP model trained on augmented dataset, train-set accuracy was `70.01%` and test-set accuracy was `78.36%`. Note that here epochs have been kept half for a fair comparison, since size of augmented dataset was twice that of unaugmented. But on `20` epochs, the model outperforms and gives a train-set accuracy of `84.9%` and test-set accuracy of `80.1%`. Augmented dataset do makes it slightly harder for model to overfit on trainset due to different image transformations done on samples, and it makes performance on test set relatively better as model has learned to fit over very different types of images due

to augmented images. Though here, the advantage of augmented dataset is not well profound and there is only a significant edge over the unaugmented dataset.

6. Derivation of gradients and update expressions

Please scroll below to see all the required derivations and calculations

Given, y = actual output, a_o = predicted output

Loss function

$$\mathcal{L}(y, a_o) = -\sum_i y_i a_{o_i} \quad \text{(Cross-entropy)}$$

To update weights and biases, we need to calculate partial derivatives of loss function wrt weights and biases.

$$\therefore \frac{\partial \mathcal{L}}{\partial w_o} = \frac{\partial \mathcal{L}}{\partial a_o} \times \frac{\partial a_o}{\partial z_o} \times \frac{\partial z_o}{\partial w_o}$$

w_o = weights in output layer.

a_o = softmax(z_o)

We know that $\frac{\partial \mathcal{L}}{\partial a_o} \times \frac{\partial a_o}{\partial z_o} = a_o - y$

$$\text{(I)} \implies \frac{\partial \mathcal{L}}{\partial w_o} = (a_o - y)\frac{\partial z_o}{\partial w_o} = (a_o - y) \times a_h$$

since $z_o = a_h \times w_o + b_o$ — (1)

Similarly, $\frac{\partial \mathcal{L}}{\partial b_o} = (a_o - y)\frac{\partial z_o}{\partial b_o} = (a_o - y) \times 1$ (II)

b_o = bias in output layer

$= a_o - y$ (From (1))

$$\text{(III)} \quad \frac{\partial \mathcal{L}}{\partial w_h} = \frac{\partial \mathcal{L}}{\partial a_h} \times \frac{\partial a_h}{\partial z_h} \times \frac{\partial z_h}{\partial w_h}$$

w_h = hidden layer weights

$a_h = \text{Relu}(z_h)$

$z_h = X \times w_h + b_h$ — (2)

Now $\frac{\partial z_h}{\partial w_h} = X$ (input)

$$\frac{\partial \mathcal{L}}{\partial a_h} = \frac{\partial \mathcal{L}}{\partial z_o} \times \frac{\partial z_o}{\partial a_h} = (a_o - y) w_o$$

$$\frac{\partial a_h}{\partial z_h} = \text{derivativeRelu}(z_h)$$

$$\therefore \implies \frac{\partial \mathcal{L}}{\partial w_h} = (a_o - y) w_o \times \text{derivativeRule}(z_h) \times \text{input}(= X)$$

(IV)

bh - bias in hidden layer

$$\frac{\partial e}{\partial bh} = \frac{\partial e}{\partial ah} \times \frac{\partial ah}{\partial zh} \times \frac{\partial zh}{\partial bh}$$

$$= \frac{\partial e}{\partial ah} \times \frac{\partial ah}{\partial zh} \quad (\because \frac{\partial zh}{\partial bh} = 1 \text{ (from (2))})$$

(These were already computed in III)

Finally, update the parameters as follows:-

$$wh = wh - lr \times \frac{\partial e}{\partial wh} \qquad (lr = \text{learning rate})$$

$$bh = bh - lr \times \frac{\partial e}{\partial bh}$$

$$wo = wo - lr \times \frac{\partial e}{\partial wo}$$

$$bo = bo - lr \times \frac{\partial e}{\partial bo}$$