Assignment 3 Bonus

DD2424 Deep Learning in Data Science

Márk Antal Csizmadia
macs@kth.se

2 June 2021

1 Introduction

This short report describes my solution to Assignment 3 Bonus, the implementation of which may be found in the script uploaded alongside with the report. I chose Option 1 (of the 2 available options) that is an assignment based on implementing batch normalisation to fully connected networks. The aim of the Assignment 3 Bonus was to optimize the performance of the k-layer network implemented in Assignment 3. The tricks/avenues I explored to help bump up performance were as follows:

1. Did a more exhaustive random search to find good values for the amount of regularization.
2. Did a more thorough search to find a good network architecture with the question in mind whether making the network deeper improves performance.
3. Applied dropout to my training as a means of regularization.
4. Augmented my training data by applying small random geometric and photometric jitter to the original training data.
5. Implemented and used the AdaGrad optimizer to more efficiently train networks.

2 Implementation

In this section, a brief description of my implementation is included. Note that the numerical gradient checks were successfully carried in Assignment 3 and its description can be found in my report and code for Assignment 3.

2.1 Neural network blocks

In my implementation, I built on top of my code from Assignment 2. At the core of my implementation is the `Dense` layer (fully-connected layer) that can have activations such as `SoftmaxActivation`, and `ReLUActivation`. The learnable (or trainable) parameters of the `Dense` layer can be initialized with the `XavierInitializer` and the `NormalInitializer`, and can be regularized with the `L2Regularizer`. Further to the `Dense` layer, batch normalization was implemented in the `BatchNormalization` layer. A `Model` comprises any number of layers, its loss function is the `CategoricalCrossEntropyLoss`. The trainable parameters are optimized with the mini-batch gradient descent algorithm via the `AdaGrad` optimizer and the learning rate schedule of the optimizer is set to be `LRCyclingSchedule` [1]. The `Model` is first compiled with the loss function, some metrics such as the `AccuracyMetrics`, and the optimizer, and then it is fit to the data with the `Model.fit` method. After each hidden layer, a `Dropout` layer is used as an additional means of regularization.

My code can be used to build a `Model` with k-layers. In this assignment, I considered the following architectures with varying number of nodes (last layer is at the end of the node number sequences):

- 9-layer: (3072, 50, 30, 20, 20, 10, 10, 10, 10, 10),

- 8-layer: (3072, 50, 30, 20, 20, 10, 10, 10, 10),
- 7-layer: (3072, 50, 30, 20, 20, 10, 10, 10),
- 6-layer: (3072, 50, 30, 20, 20, 10, 10),
- 5-layer: (3072, 50, 30, 20, 20, 10),
- 4-layer: (3072, 50, 30, 20, 10),
- 3-layer: (3072, 50, 30, 10)

Note that in the list above, the number of nodes in the input layer are always 3072 due to the size of the flattened CIFAR-10 images. The last layer always has 10 nodes due to the 10 classes in the CIFAR-10 dataset.

All of the `Dense` layers' trainable parameters are initialized with the `XavierInitializer` and regularized with the `L2Regularizer`. The variants of the aforementioned networks also contain a `BatchNormalization` layer after each hidden layer. Note that there is no batch normalization after the last layer which outputs the scores, or predictions of the network. The hidden `Dense` layers have `ReLUActivation` activations, and the last `Dense` layer (classifier) has `SoftmaxActivation` activation. The `Model` is then compiled with the `CategoricalCrossEntropyLoss` loss function, the `AccuracyMetrics` performance metric, and the `AdaGrad` optimizer that uses the `LRCyclingSchedule` learning rate schedule.

The mini-batches of the training data are augmented during the execution of the mini-batch gradient descent algorithm. Augmentation includes the following techniques:

- Horizontal flipping of 20% of the image batches
- Small Gaussian blur with random standard deviation between 0 and 0.5, with only blurring about 50% of the images in the batch

The augmentation is implemented in the `aug_f` function wit the aid of the *imgaug* library [2] and is passed to the `Model.fit` method as an argument.

A hyper-parameter search was conducted to find good values for the amount of regularization and to find a good network architecture with the question in mind whether making the network deeper improves performance. The hyper-parameter search was a random grid-based Bayesian search, which was implemented with the aid of the *hyperopt* library [3]. The `objective` function builds and trains a model with one of the architectures described before using the `dims` parameter. The model's layers use the L2 regularization strength `reg_rate`. The hyper-parameter search is implemented with the `objective` method and the `run_trials` methods.

3 Results

In generating the results discussed below, the whole CIFAR-10 data set was used. For training the networks, the training and validation data sets included `data_batch_1`, `data_batch_2`, `data_batch_3`, `data_batch_4`, and `data_batch_5`. Where not specified otherwise, the validation set was 5000 randomly sampled image and the training data set was the 45000 remaining image. At test-time, the `test_batch` set of 10000 images was used. Please note that the accuracy measures are given as real numbers between 0.0 and 1.0, where 0.0 mean 0.0% and 1.0 means 100.0%.

The *default parameter settings* are as follows:

- The k-layer network has k `Dense` layers with one of the aformentioned architectures. A `BatchNormalization` layer is attached after each hidden layer.
- The learnabele parameters of each `Dense` layer are initialized with the the `XavierInitializer` with a mean of 0 and a standard deviation of $1/\sqrt{in\_dim}$ where `in_dim` is the number of hidden nodes in the previous layer.
- The base and the maximum learning rate (`eta_min` and `eta_max`) of the `LRCyclingSchedule` are 1e-5 and 1e-1, respectively.
- The step-size (half-cycle) of the `LRCyclingSchedule` is $n\_s = 5 * 45,000/n\_batch$. Since $n\_batch = 100$ throughout the assignment, $n\_s = 2250$ update steps. Equivalently, one cycle is two step-size, that is 4500 update steps, which equals to 10 epochs. The networks are trained for 2 cycles, that is, for 20 epochs for each iteration of the hyperparameter search. The network trained with the best L2 regularization rate `lambda` and network architecture from the search is trained on the union of the training and the validation datasets for 4 cycles, that is, for 40 epochs, with a step size of 2500.
- The `gamma` and `beta` learnable parameter matrices of the `BatchNormalization` layer are initialized to ones and zeros, respectively. The `momentum` hyperparameter and `epsilon` of each batch normalization layer are set to 0.9 and 1e-5, respectively.
- The `Dropout` layers keep the connections with a probability of $p = 0.9$.

The results of the hyper-parameter search are shown in Table 1.

trial	val_accuracy	lambda	n_layers
0	0.4636	$1.47{\cdot}10^{-3}$	7
1	0.4896	$3.81{\cdot}10^{-2}$	3
2	0.4126	$3.6{\cdot}10^{-2}$	7
3	0.3924	$4.97{\cdot}10^{-2}$	9
4	0.466	$5.1{\cdot}10^{-2}$	5
5	0.3922	$7.39{\cdot}10^{-2}$	8
6	0.4666	$4.76{\cdot}10^{-2}$	4
7	0.382	$3.89{\cdot}10^{-2}$	8
8	0.4818	$9.02{\cdot}10^{-3}$	6
9	0.4034	$7.91{\cdot}10^{-2}$	7
10	0.4772	$5.66{\cdot}10^{-2}$	3
11	0.406	$4.97{\cdot}10^{-2}$	7
12	0.4488	$6.98{\cdot}10^{-2}$	6
13	0.4578	$1.58{\cdot}10^{-2}$	8
14	0.4032	$5.04{\cdot}10^{-2}$	7
15	0.506	$1.4{\cdot}10^{-2}$	3
16	0.3452	$8.1{\cdot}10^{-2}$	9
17	0.454	$3.84{\cdot}10^{-2}$	7
18	0.475	$7.94{\cdot}10^{-2}$	3
19	0.4878	$1.94{\cdot}10^{-2}$	5
20	0.4956	$2.42{\cdot}10^{-2}$	3

Table 1: Results of the hyperparameter search for good L2 regularization strengths, `lambda`, and a good network architecture

The `n_layers` column refers to the network architectures described earlier in the report. As shown in Table 1, the best `lambda` and network architecture were found in trial 15 and are 1.4×10^{-2} and 3-layers, respectively. It was found that deeper architectures perform more poorly, as exemplified by, for instance, trial 3, 7, and 16. This could be remedied by adjusting the learning rate and other hyperparameters, but this is beyond the scope of the assignment.

The best model found in the aforementioned hyperparameter search was re-trained for 4 cycles on the union of the training and the validation datasets (i.e.: 40 epochs with a modified cycling schedule step size of 2500, due to merging the training and the validation sets). Figure 1 show the results of training the final network.

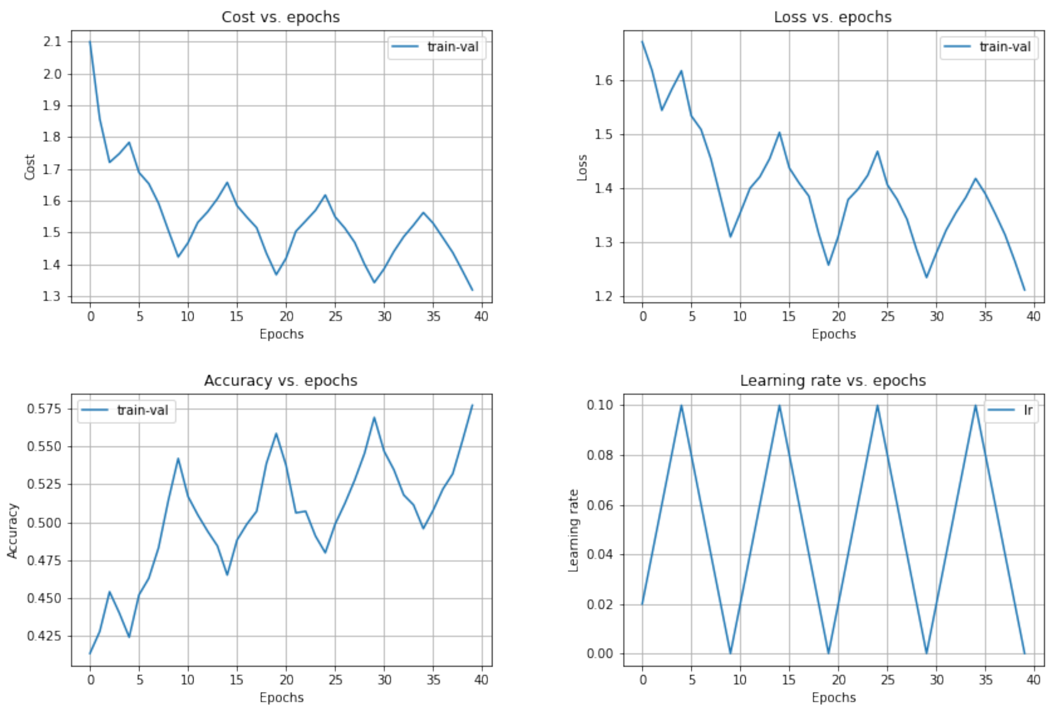

Figure 1: The cost, loss, accuracy, and learning rate curves of training the final network for 4 cycles (i.e.: 40 epochs). Note that accuracy measures are given as real numbers in the range 0-1.

Note that the reason for the learning rate curve being shifted is that the learning rate is always recorded at the end of an epoch (i.e.: update steps have already taken place). Also note that the validation dataset is now a subset of the train-val set, on which the final model is trained. Therefore, only the train-val curves are shown.

The test accuracy of the final trained model was 0.5213, which is a relatively decent performance and which indicates that the a good value for the L2 regularization rate and the network architecture were found during the hyperparameter search. Furthermore, data augmentation as well definitely helped achieve such a good generalization capability.

4 Conclusions

In this assignment, different k-layer, fully-connected networks were trained to classify images in the CIFAR-10 dataset. Training networks with three to nine layers, batch normalization, and dropout using the AdaGrad optimizer and data augmentation, a hyperparameter search was conducted to find a good network architecture and a good value for the L2 regularization. The best model showed decent performance indicating the that the hyperparameter search discovered worthwhile regions.

References

[1] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*, IEEE, 2017, pp. 464–472.

[2] A. B. Jung, *imgaug*. 2020, [Online; accessed 2-June-2021]. [Online]. Available: `https://github.com/aleju/imgaug`.

[3] J. Bergstra, D. Yamins, and D. D. Cox, *Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures*. Proceedings of the 30th International Conference on Machine Learning, ICML 2013, 2013. [Online]. Available: `http://hyperopt.github.io/hyperopt/`.