Deep Learning

*Data pre-processing*

Athens 2018

Avgeros Ioannis 2022201704003

In the preprocess step, I decided to perform a unity scale normalization of the values that scale from 0-255(8-byte format of a pixel representing the color of the pixel) to scale 0-1. I did that with the following type:

D:\userdata\avgeros\Desktop\1.PNG

In our case that translates to simply dividing by 255(min value =0 and max value=255).

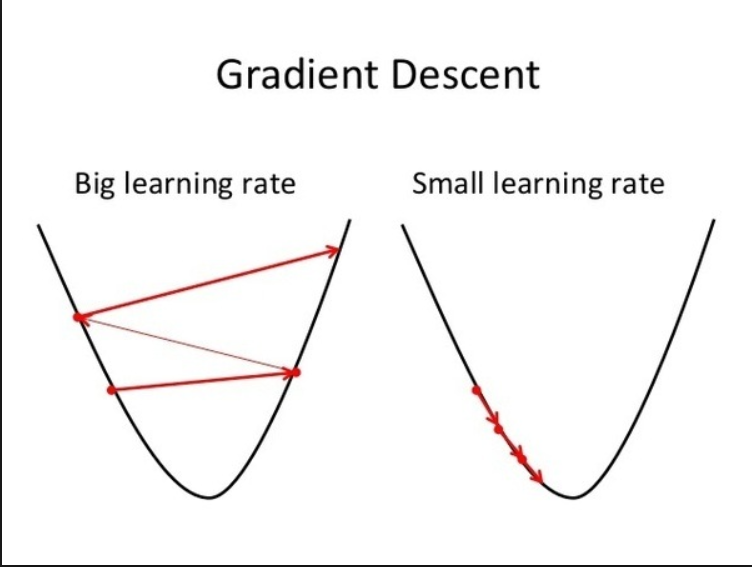
Normalizing the inputs can make training a lot faster and reduce the chances of getting stuck in local optima hence it is highly recommended before inputting the data into the network to train

*Neural Network Design*

For the design of the network, I implemented a grid search over the parameters of the neural network and the optimizer. The absence of GPU was a very important factor to make me choose the Dense Layer architecture. Dense layer is a simple ‘connect everything with everything’. I made some tests using some random parameters before performing grid search, aiming to narrow down the range of the variables to optimize. I noticed and followed the following properties for a dense layer architecture network and cifar10 dataset and enforced the parameter search within that observations.

* The first hidden layer to have the most units compared to the following. Thus, I designed combinations of three dense layer architectures, the first layer had twice the number of units compared to the second dense layer and the third layer has a quarter of 1st layer units.
* As far as the optimizer is concerned, due to the limitation of the CPU ,I present a site where is has an extensive analysis on Adam and the case that is the most appropriate optimizer for cifar10 dataset ([link](https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/)). It compares the accuracy between different classifiers and concludes that the Adam is the most appropriate for the dataset.
* I also included Batch normalization layer. Batch normalization layer reduces the amount by what the hidden unit values shift around (covariance shift). The network should be trained to identify airplanes regardless of redundant characteristics, such as the colour.
* Softmax activation function was used for the output layer to compute the probabilities per class. ReLu activation function was chosen as the next layer of the dense one. Relu helps to eliminated values, speed up training and “*many practitioners have favored the simplicity and reliability of ReLU because the performance improvements of the other activation functions tend to be inconsistent across different models and datasets*”([*Search for activation function*](https://arxiv.org/pdf/1710.05941.pdf)*)* .
* Below I explain the logic behind the values selected to form the most accurate neural network:

1. Epochs [15]: Only one parameter, as most examples I run converged to the maximum accuracy before the 15th epoch. Another good idea would be to execute for 30 epochs, but that would double the execution time without yielding much better results.
2. Batch size [128]: Batch size affects regularization, the lower the number the better the regularization and the faster the training would be.
3. Learning rate [0.001, 0.01,0.1]: The learning rate affect the rate the loss function converges. The parameters cover a range from a small one (0.001) to a much faster one (0.1). Smaller values proved to affect the speed of convergence negatively and bigger values tend to jump randomly from point to point.



Εικόνα 1: Different learning rates and the gradient.

1. Dropout rate [0.2,0.35,0.5]: Dropout rate vastly improves regularization. Mostly used values are 0.2 and 0.5. I included also the median value of 0.2 and 0.5, the value 0.35 to see if it will have and affect to the accuracy of the network. In each step, a percentage of the neurons is deactivated from that, the remaining neurons will have to perform extra computations to step up and make predictions for missing neurons. The network is forced to learn multiple independent representations for each problem.
2. Neurons [64,128,256]: Neurons mostly are a power of two in dense layers. It helps efficiently split up tasks to different cores. Neuron number can take values. Based on this [paper](https://openreview.net/pdf?id=1WvovwjA7UMnPB1oinBL), there can be a good representation of densely connected layer even with one linear layer with 1000 units, accompanied with 2 relu layers. After executing multiple tasks (including 1024 and 512 dense layer unit size) the accuracy did not improve that much, whereas the executional time became much higher.

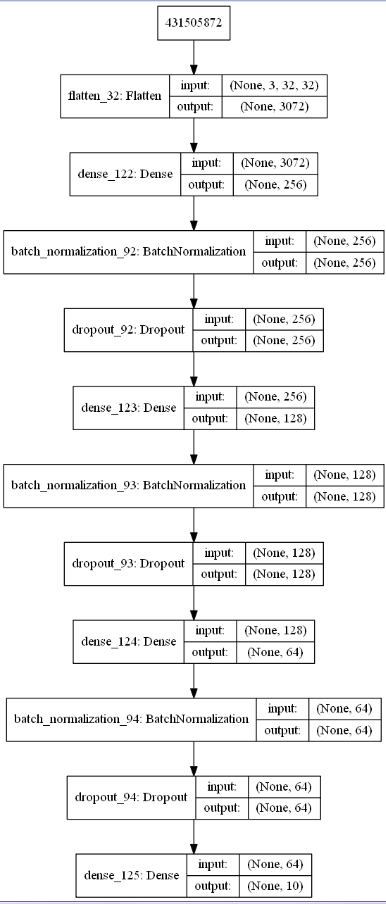
As I performed the parameter search (and most of runs I performed with random parameters), I noticed most cases were hovering around the range 0.3-0.5. The neural network seems to fail to perform good on the 10.000 validation images and that is normal .as the most appropriate neural network design would include convolution.

*Best Results*

The results proved to be the best for the following parameters:

* ‘learn\_rate': 0.001
* 'epochs': 15
* 'dropout\_rate': 0.2
* 'neurons': 256

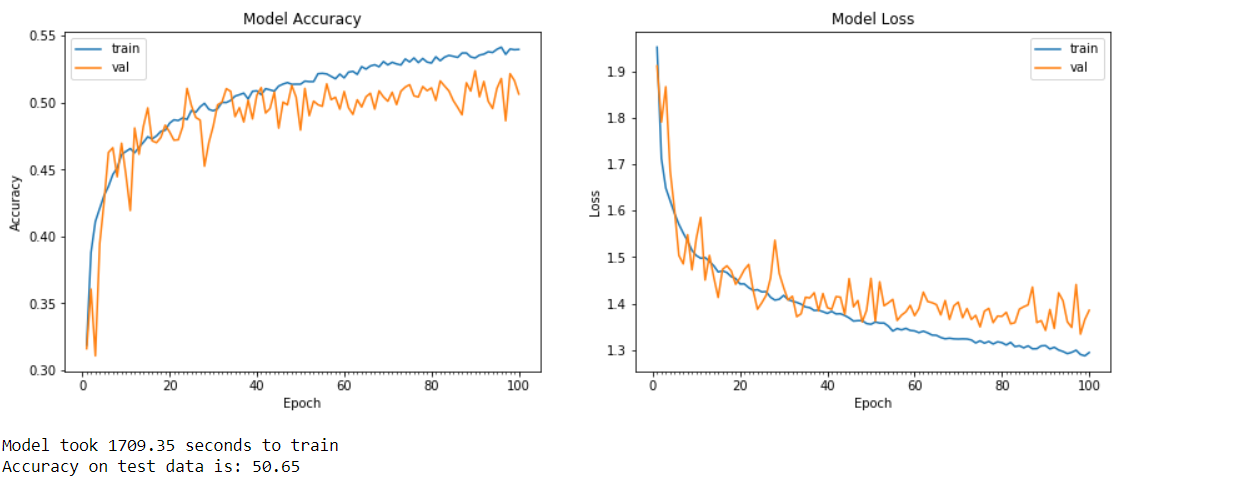
We can see that the network is constructed as it follows:



Εικόνα 2 Graphical view of the network.

Lastly, model trained for 100 epochs proved to increase

the accuracy a little bit, but fail to pass the threshold of 53%.



The explanation of metric for the above graph:

* ***Loss***: A scalar value that we attempt to minimize during our training of the model. The lower the loss, the closer our predictions are to the true labels. Here we use the categorical cross-entropy.
* *Accuracy*: The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then the test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. While accuracy may lead to inaccurate results, specifically for cifar dataset is valuable as the classes are balanced.

To sum up, model achieved an accuracy of 50% on the validation set after training for 100 epochs, using the parameters learned from a grid search over asset of values. I used relu and softmax as activation functions and used a small batch size, batch normalization and dropout to achieve regularization.