Deep Learning, MSc in Data Science

Assignment 1: A Simple Classification Network

VOULGARI ELENI

A.M. 17005

Introduction

The task of this assignment is to build an Artificial Neural Network, that can sufficiently classify the CIFAR-10 images by choosing the most appropriate optimizations.

The CIFAR-10 dataset consists of 60.000 color images of size 32x32x3. Each image belongs to one of the 10 equal classes (6.000 images per class) and each class corresponds to a physical object (horse, ship, truck, bird, etc). There are 50.000 training images and 10.000 test images.

To create as much as possible an accurate neural network for predicting the right class of the above images, we experiment with different depths and widths of the network as well as with different loss functions and optimizers. We also experiment with regularization, dropout and data augmentation techniques.

Data Preprocessing

After downloading the data and saving it in an array we should preprocess it. The preprocessing includes the normalization of the integers that appear as the intensity of the color they represent. Their range is from 0-255 and we need to scale it down to floats in the unit interval [0,1]. Neural networks might train faster if their input is normalized values between [0,1] or [-1,1] due to their optimizers. It is essential to normalize if the input data has different scales. In our case there are no different scales, but we perform the normalization.

We are also splitting the training set into the training and the validation set for keeping the performance in track and for the model to be able to adjust.

Basic Choices for the models

To begin with, we defined the batch size to be of size 256. Some claim that the larger the batch size the better the accuracy of the network, while others suggest that small batches make the networks perform better. For one of the models we used a bigger batch size to see what's the case.

Furthermore, the maximum number of epochs was defined to be 150 and the early stopping callback was decided to be used. Early stopping is basically stopping the training once your loss starts to increase (or in other words validation accuracy starts to decrease). It monitors the loss and once it starts to increase it runs for two more epochs to give it a chance to decrease again (patience=2).

For all the models described below the choice of the activation functions was the following: Rectified Linear Unit for the hidden layers and Softmax for the output layer. The reason behind this choice is that with the ReLU function in the hidden layers we are avoiding the vanishing gradient problem. Many activation functions (e.g sigmoid) scale down their input into a small output range. As a result, large regions of the input space are mapped to an extremely small range. This means that even a large change in the input will produce a small change in the output so the gradient is small and untraceable for learning. In our case the input is between [0,1] but still the ReLU function is a good choice. With the Softmax function in the output layer we get the normalized exponential probability of class observations represented as neuron activations.

We generally use the Adam optimizer for the models, except for the third model that we tried out the SGD optimizer with Nesterov momentum. The reason for the use of Adam optimizer is that it offers several advantages over the simple gradient descent optimizer, one of which is momentum, which can give faster convergence. That's why we use the the SGD optimizer with Nesterov momentum for one of the models to compare with Adam. Another benefit from Adam optimizer is that it adaptively selects a separate learning rate for each parameter. So, parameters that would normally be updated less frequently, with Adam they receive larger updates. In cases where the appropriate learning rates vary across parameters, Adam speeds learning.

First Base Model

For our first model, we created a NN with 8 hidden layers and 512 neurons per layer. We used the early stopping function and the accuracy produced was 49.6%. The NN gives a better result than the random guess

(10%), but we observe that the validation loss is much higher than the training loss **(the legend of all the plots is inadvertently wrong – instead of the test set, it should refer to the validation set)**. The model needs to be regularized to have better generalizability.

Second Base Model - Generalization

To improve the performance we can reduce the complexity of the model by using fewer layers and neurons per layer. We chose random numbers for the above parameters, thus we decided to have 4 hidden layers each one with decreasing number of neurons (512, 256, 64, 32).

As we can observe with the random selection there has been a small increase in the accuracy and in the instances that were predicted correctly. We can choose other parameters randomly and do this until we gain better results but this would be time consuming and it could lead to no better performance.

Third Model - Optimizer and Loss

For the next model we used the mean squared error for the loss and the SGD with Nesterov momentum for the optimizer. We also defined the learning rate of the SGD to 0.01. In Nesterov momentum version it first looks at a point where current momentum is pointing to and then computes gradients from that point.

There is a small improvement over the first model which had the Adam optimizer but still lower than the simple second model we built.

Fourth Model – Regularization (Weight decay)

For our fourth model we used a regularization technique called weight decay and with this method we penalized large weights in the cost function. So, the method is to add a term in the cost function that penalizes the L^2-norm of the weight matrix at each layer. This kind of regularization made no difference to the accuracy of the model.

Fifth Model - Regularization (Dropout)

For our fifth model we used another regularization technique, the dropout method, which is deleting a random sample of the activations (makes them zero) in training. This is a way to force each neuron to rely more on its own and not on the other neurons.

As we observed the dropout method caused the accuracy of the NN to drop significantly but we also notice underfitting in this model. The dropout network might require more epochs to train. Or it might have to be calibrated, to begin with low dropout in the first layers and then increase it gradually, because dropout “loses” information and if something is lost in the first layers, then it is lost for all the layers, hence the whole network.

Sixth Model - Data Augmentation

For our sixth model we used the data augmentation technique. NNs are known to need a lot of data to be able to train well. So we performed data augmentation and train the model again with "more" instances. This is done by "augmenting" each instance by random transformations, without changing its class label and it prevents overfitting when the data are of limited size.

Keras contains the ImageDataGenerator class, that can apply several transformations to the images, like rotation, flip vertically or horizontically, zoom, etc. An important thing to consider is that the transformations we apply should not “change” the class label of the image. Let's say, if we have an OCR (Optical Character Recognition) application and flip vertically and then horizontally the character “d”, it will become “p” and then its label will not match the images itself.

As we can observe the data augmentation technique slightly improved the accuracy of the first model, but we also notice underfitting.

Seventh Model - More neurons per layer

To avoid the underfitting problem we can try increasing the number of the neurons each hidden layer has. Unfortunately, the problem was not fixed and the choice to run it with more neurons was pretty time consuming given the technical aspects of the system it was built.

Eighth Model - Increase Batch size with best model

So, for the our last try we decided to take the model that produced the best accuracy and only change the batch size to 1024 to see if it gets better. The observation here was that the performance dropped so it needs to be further be tuned.

Ninth Model - Last try

For our last try, we used the 4 hidden layers with decreasing number of neurons in each layer. We also used the dropout method but not for he first layer and with increasing rate for each of the next two layers. Batch size was reduced to 128 and epochs were reduced to 40 without the callback for early stopping of the network.

As we can observe this model is the best so far and this might be because of the batch size or the fact that we did not use early stopping and the network ran for the whole 40 epochs. Increasing the epochs might produce better results. The problem of underfitting is not avoided and this need to be further investigated.

Conclusions

The Artificial Neural Networks are powerful models that can be used in a wide range of applications, but they can be very costly in calculation. They need computing machines that can perform high. A significant amount of runtime is needed to process large datasets. Furthermore, it is very hard to interpret and measure the impact of individual neurons and the gradient descent cannot guarantee the global minimum of the loss function.

The fact is, that it is not easy to choose the right parameters for them, as neural networks are like humans. They can exist in all variations and there is not one best fit for all problems.

They work well on particular problems like image recognition, but the use of a Convolutional instead of a Dense Neural Network will make the difference in the precision of the predictions.