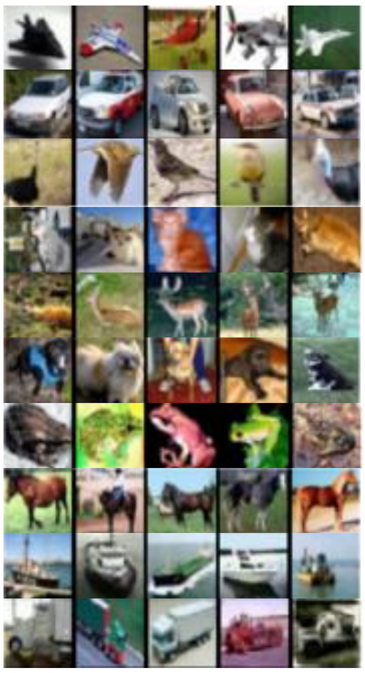
MICHAŁ RYSZARD BALICKI

Neural networks — Project report

PROBLEM DESCRIPTION

The goal of the final project assigned for “Neural networks” is to construct a network, which will be able to learn to classify the photographs from the dataset of the Canadian Institute of Advanced Research with 10 types. The dataset consists of 60 000 colour photographs, each of size 32 × 32 and each labeled assigned to it. The ten labels are:

— airplane —
— automobile —
— bird —
— cat —
— deer —
— dog —
— frog —
— horse —
— ship —
— truck —



PROPOSED SOLUTION

My network capable of recognising photographs of the CIFAR-10 dataset with the error of 22.09% is largely based on the network shared by dr Jan Chorowski as a lecture material “10-convolutions.ipynb” on the website of the subject “Neural networks” 2015.

The file contains a convolutional network built using the open source computation library “Theano” which effectively categorises pictures from the Mixed National Institute of Standards and Technology database.`

NETWORK ARCHITECTURE

My network consists of the following parts:

— convolutional layer of 50 filters of size 5 × 5
— rectified linear unit activation
— max-pool downsampling with filters 2 × 2
— convolutional layer of 75 filters of size 5 × 5
— rectified linear unit activation
— max-pool downsampling with filters 2 × 2
— fully connected layer of 500 neurons
— rectified linear unit activation
— fully connected layer of 10 neurons
— softmax normalisaton

FINE TUNING

After adjusting the convolutional neural network provided by dr Chorowski to the technical requirements of the CIFAR-10 dataset the error rate was equal to 33.83%. To find the limits of this architecture I begun changing parametres of the network. My experimental approach led to some interesting observations.

momentum	lrate_const	test error rate	best epoch	max epoch	valid_err_rate	avg train_err_rate
0.90	0.002	34.41%	27	42	34.25%	20.3725%
	0.004	32.91%	15	24	33.83%	9.6500%
	0.008	32.05%	14	22	30.99%	0.3525%
	0.012	32.45%	10	16	32.40%	1.4225%
	0.016	31.46%	9	15	31.19%	2.0025%
	0.020	31.21%	31	48	30.28%	0.0050%
	0.024	31.16%	26	40	31.48%	0.0475%
	0.028	31.24%	33	51	30.55%	0.0100%
	0.032	32.76%	11	18	32.59%	5.2225%
0.70	0.024	31.47%	9	14	31.23%	4.8075%
0.75		31.41%	16	25	31.01%	0.0825%
0.80		30.95%	9	14	30.98%	1.7150%
0.85		31.09%	26	40	31.34%	0.0075%
0.90		31.16%	26	40	31.48%	0.0475%

An empty cell means that the same value was used as in a row above.

I started with testing different possible values of the base constant used in the expression for `lrate` and concluded that by modifying it I am able to improve the results significantly. With every other parametre remaining unchanged I got to the value of the test error rate of 31.16% by sextupling the `lrate_const`.

For this precise value of `lrate_const` I performed numerous test for different values of the `momentum` parametre and conculded that it doesn't influence my results greatly. However, by setting it to `0.80` I was able to get down with the error rate a little bit to 30.98%, thus I settled to introduce this change.

K	momentum	lrate_const	num_filters_1	num_filters_2	gauss	test error rate
2000	0.004	0.90	10	25	0.050	32.91%
	0.024	0.80				30.98%
	0.024	0.80		35		30.08%
	0.024	0.80	15	50		28.35%
	0.024	0.80	15	50	0.025	27.53%
	0.024	0.80	50	50	0.025	25.91%
4000	0.024	0.80	50	50	0.025	26.03%
3000	0.024	0.80	50	50	0.025	25.69%
3000	0.024	0.80	50	75	0.025	25.34%

The table above presents second part of fine tuning the parametres of the network after settling with values of `momentum` and `lrate_const`. The greatest improvements came from increasing the number of filters used in convolutional layers — collectively the improvement was of 3 percentage points. Other improvements came, although to a lesser extent, from decreasing the standard deviation of the Gaußian distribution used as a bias and from increasing the `K` constant. Altogether, I was able to bring the test error rate down to 25.34%, just a little bit over the expected threshold.

LEARNING BREAKTHROUGH

Even though I was convinced that I would be able to make my network achieve the expected quality just by adjusting the numeral parametres, I felt obliged to implement one of the possible improvements which were discussed on the lecture on this topic. Among them I was most interested in testing out the improvement yielded by choosing batches with probability ½ to flip vertically every image from that batch.

I recall classes of “Knowledge of Culture” in my liceum, where often we were presented with works of art in a form of reversal slides. And since our teacher didn’t care for the order of those slides, she didn’t care whether the pack was inserted correct way or other way around. Nevertheless, with a great accuracy I was able to spot in which direction the pack was inserted, just by seeing the direction of photographs. And I wouldn’t say that I didn’t believe in the improvement that could be achieved by implementing this random flipping, but, since the dataset consists of real photographs, all the more I was interested in checking that myself. Therefore, I inserted the following piece of code:

```
if np.random.randint(2) == 1:
    X_batch = X_batch[:, :, :, : : -1]
```

As it turned out, vertical flipping yielded results which were more than satisfactory — the test error rate decreased to 21.62%, thus making my network achieve the expected threshold of 25% and even exceeding it by a large margin. Since the performance of the network is randomised, the final result which I present in the attached notebook is of 22.08%. List of predicted labels and true ones is available in the file `listo_KIPE10.txt`.