CIFAR-10 classification project

Grzegorz Jurdziński
Department of Computer Science
University of Wrocław

February 19, 2016

Abstract

Image classification is one of the most important problems in machine learning. Popular and efficient approach are convolutional neural networks. There are also problems with them we need to deal with. In this paper I describe my work on such network, my experiments and improvements.

1 Introduction

Image classification problem is a task of matching images with certain labels. It's widely researched area due to it's practical applications. One of the most popular approaches to this problem are convolutional neural networks. In my project I've been working on classifying images from CIFAR-10 data set, that is a popular benchmark set for this problem. It contains 60000 32×32 images in 10 classes with 6000 images per class. There are 50000 training images and 10000 test images. Figure 1 shows examples of images from the set (image was borrowed from [2]).

One of the problems with convolutional neural network is overfitting. After long time of learning network starts to classify examples from the training set almost perfectly while predictions for samples from validation and test set doesn't get better or even get worse. There are few methods to prevent this that I've applied.

In this paper I'm going to show results of my work, describe method's I've applied and compare them.

2 My solution

I've used convolutional neural network with dropout. I've also applied simple image transformations before feeding them to the net. I've applied convolutions and max pooling to the image. Their role is to extract certain features of the image what should let following layers to classify it correctly. After convoluting and max pooling the image is flatten to one dimensional array that is applied to several affine layers. After that we apply it to affine layer with SoftMax activation function and ten units which produces probabilities for each class. The one with the biggest probability is being chosen as a result.

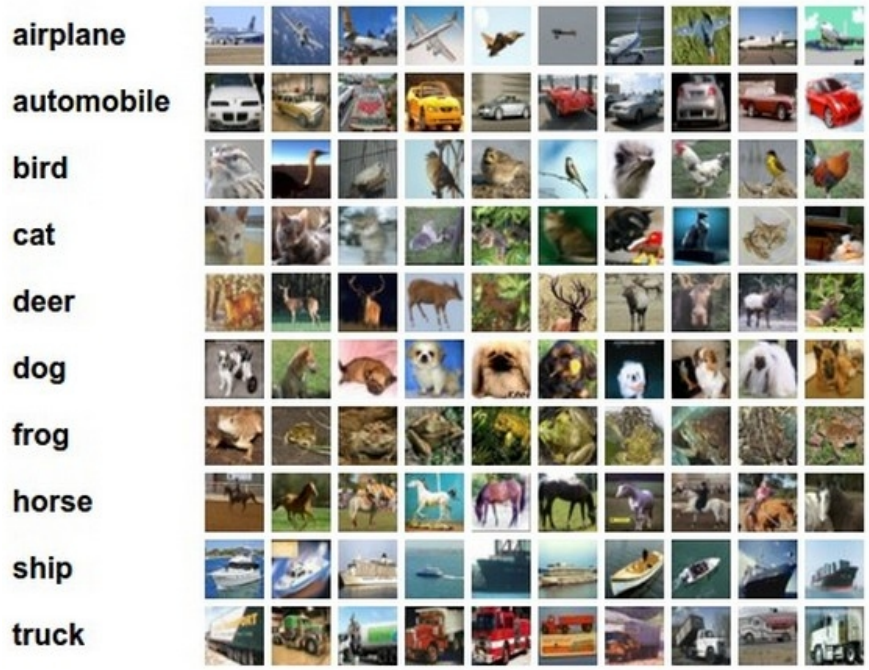

Figure 1: Examples of images from CIFAR-10

2.1 Technical details

I've been working in Python (exactly IPython notebook) with Theano and Lasagne. Lasagne is quite convenient library build on top of Theano that provides various utilities for building neural networks, including ready to use layers. I've been running my computations on computers in II UWr, especially computers 1-6 in room 110 since they have GPU supporting CUDA.

2.2 Learning

For learning I've used Stochastic Gradient Descent algorithm with early stopping. It's very popular algorithm, giving good results for big data sets. I've divided training set into two parts – training set (40000 images) and validation set (10000 images). For each training step mini-batches of 25 images were randomly sampled. After each epoch network was tested on validation set consisting of 100 images big mini-batches. Final performance was checked on test set.

During learning I was minimizing loss function for witch I've chosen categorical cross-entropy as it is widely recommended (e.g. in [1]).

Learning rate was set before computing each batch according to following schedule: `l_rate = base_lrate * K / max(K, i)`, where `base_lrate = 1e-2`, `K = 10000` and `i` is batch number. Such schedule allows network learn fast in the beginning and slow down when we're near to the result and want to make smaller steps.

To accelerate learning I use momentum method. For momentum constant

I've chosen at first 0.9 basing on my experience from assignments for exercises. It worked well so I didn't change it.

I've also used weight decay regularization as one one of the ways to prevent overfitting. To the loss function I've added L2 norm of weights of convolutional layers and affine layers. It is supposed to prevent weights from being very big as it would probably fit to the test data but not generalize. A bit more on weight decay is described in *Dropout* section.

Lasagne implements Nesterov Momentum learning method that I've used. It works similarly to regular gradient descent with momentum although it's convergence is accelerated. Detail can be found in [5].

2.3 Architecture

In project I've used neural network with following architecture:

- Convolution layer (50 filters, 5×5 filter size, ReLU activation function)
- Max pooling layer (2×2 filter size)
- Dropout layer (0.2 dropout probability)
- Convolution layer (100 filters, 5×5 filter size, ReLU activation function)
- Max pooling layer (2×2 filter size)
- Dropout layer (0.2 dropout probability)
- Flatten layer (flattening image to one dimension)
- Affine layer (500 units, ReLU activation function)
- Dropout layer (0.5 dropout probability)
- Affine layer (500 units, ReLU activation function)
- Dropout layer (0.5 dropout probability)
- Affine layer (100 units, ReLU activation function)
- Dropout layer (0.5 dropout probability)
- Affine layer (10 units, SoftMax activation function)

At the beginning architecture was chosen basing on lecture example ([1]), so it had two convolutional layers and one affine layer (not counting the one with softmax). After some experiments I've found that architecture with two more affine layers works better, especially when combined with dropout. Apparently it has then a possibility to extract some more image features. It's also possible that when dropout was applied to the only one affine layer it had too big impact. When we have more layers dropout doesn't spoil the net and sill regularizes it.

Weights for affine layers were chosen according to advice from [4] and for convolutional layers according to [1].

Figure 2 shows outputs of convolutional an max pooling layers for sample image. We can see how they extract certain features of the image and even focus on car forgetting the background.

Using this architecture with dropout turned off and without image transformation the network has reached 75.41% correctness on test set.

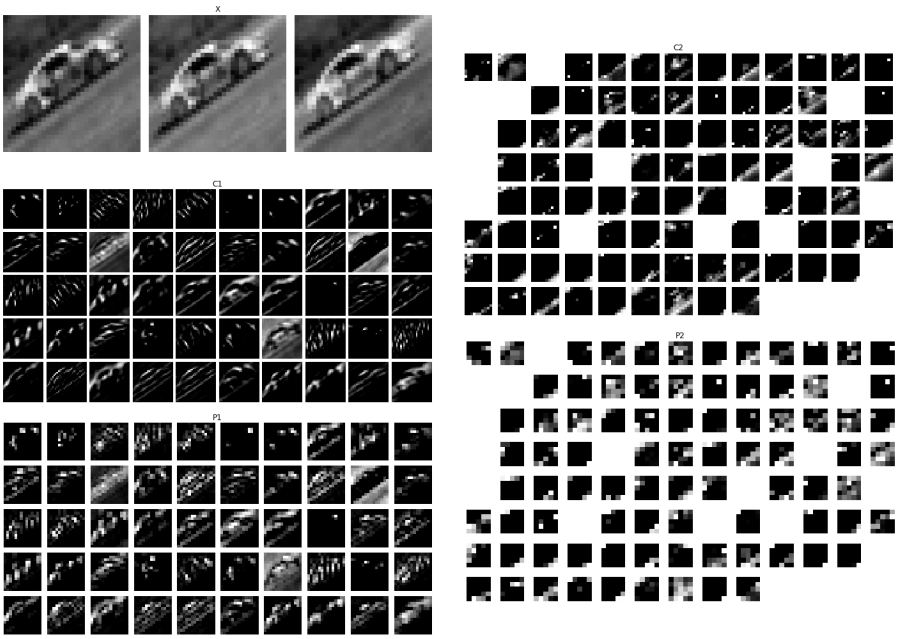

Figure 2: Outputs of convolutional and max pooling layers

3 Improvements

Except basic improvements (like weight decay) in my network I'm using simple image transformations and dropout layer which I'll describe in this section. Both of them prevent overfitting and raise network performance.

3.1 Simple image transformation

One of popular and very useful improvements for image classification networks are simple image transformations. The idea is to change the image slightly so that we have new sample but it is still a good example of its class. Such transformations might me random shift (by up to 4 pixels vertically and horizontally in my case) or horizontal flip. This way we artificially create new samples so the network can be trained better and it's harder for it to overfit (we have bigger diversity in out training set).

I've implemented two simple image transformations:

- random shift – image is shifted vertically and horizontally by random number of pixels in random direction
- horizontal flip – image is flipped horizontally

Using this improvement (without dropout) raised network performance on test set to 81.39%.

Figure 3: Sample image

Figure 4: Shifted image

Figure 5: Flipped image

3.2 Dropout

Dropout is another technique used to prevent overfitting, very popular not only for convolutional neural networks and image classification. The idea is to "turn off" each connection between two neurons with certain probability p (and divide the outputs by p so they're "strengthened" due to the fact that there is less of them). Justification of using dropout is that it prevents co-adapting too much between layers.

Dropout is applied through dropout layers. We want to put dropout after decisive moments so I put it after max pooling layers and affine layers. Since convolutional / max pooling layers can "cooperate" extracting certain features of the image I set lower probability dropout after them (p = 0.2) and bigger after affine layers (p = 0.5). It also gave best results.

Since dropout prevents overfitting we may use weaker regularization so I have experimented with lower weight decay constant. I have run my test on network using both dropout and image transformation. Previously weight decay constant was 5e-5

One of the cons of dropout is increase of required learning time. Since some connection are "turned off", we don't change them during backprop. Epochs required to reach best result raised from about 200 to over 500 or even 800 (when I left network for over a day and a night it was improving even in epoch over 2000 although since around epoch 900 improvements were only slight)..

Weight decay constant	Test error rate
0	16.97%
1e-5	17.62%
1e-4	15.96%
1e-3	13.98%
5e-3	18.21%

As the table shows, the best result was reached for weight decay constant set to 1e-3 (86.02%), although network worked pretty well even without weight decay, reaching better performance than without dropout.

4 Final result

Best result I've reached is 13.98% test error rate, reached for network with dropout and image transformation.

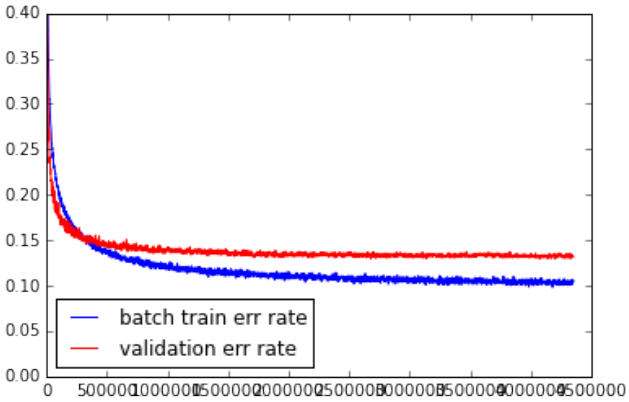

Figure 6: Error rate during learning for my best network.

As we can seen on Figure 6 network still overfits a bit although not much.

Table 1 shows best results for different combinations of my improvements (CNN stands for Convolutional Neural Network).

Network	Test error rate
CNN (without weight decay)	over 25%
CNN	24.59%
CNN + dropout	19.38%
CNN + img trans.	18.61%
CNN + dropout + img trans.	13.98%

Table 1: Best results for different improvements.

References

[1] J. Chorowski. Lecture notes, slides and notebooks for Neural networks course at II UWr (winter 2015/16). II UWr, 2015/16

[2] https://www.cs.toronto.edu/ kriz/cifar.html (access: 17.02.2016, 16:24)

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overtting. *Journal of Machine Learning Research*, 2014

[4] Y. LeCun, L. Bottou, G. B. Orr, K. R. Müller. Efficient BackProp. *Neural Networks: tricks of the trade*, Springer, 1998.

[5] I. Sutskever, J. Martens, G. Dahl, G. Hinton. On the importance of initialization and momentum in deep learning. *JMLR: W&CP*, volume 28, 2013