Regularization in Convolutional Neural Networks

Josep Famadas
jfamadas95@gmail.com

April 5th, 2018

Contents

1 Introduction

Convolutional Neural Networks are very similar to ordinary Neural Networks from the point of view that they are made up of neurons that have learnable weights and biases, each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.

What makes them different is that convolutional neural networks make the explicit assumption that the inputs are images, which allow to create the convolutional layers which make hugely reduces the amount of parameters in the network without losing performance.

As any of the other machine learning techniques, the goal of the convolutional neural networks is the generalization. This concept refers to how well the model performs with data it has not been trained with. If there is no care taken during the training phase, it can lead to overfitting which happens when the models learns too well the details and the noise from training data, but it does not generalize well, so the performance is poor for testing data.

The solution to the problem of overfitting is the regularization, which is going to be studied in this document.

2 Dataset: CIFAR-10

The CIFAR-10 dataset (Canadian Institute For Advanced Research) is one of the most widely used datasets for machine learning research. It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

For the sake of this work, the dataset has been divided into 40000 training images, 10000 validation images and 10000 test images.

3 Basic Convolutional Neural Network

3.1 Architecture

This work starts with a basic convolutional neural network in which the regularization techniques will be applied. As it can be appreciated in Figure 1 this is a deeper version of the network provided in the guided lab. The objective of making it deeper is to make it more prompt to overfit and see clearly the regularization effects.

The activation function selected for this architecture is the ReLU because it outperforms the classical sigmoid by preventing the saturation [3]. This function is applied on every layer (convolutional and fully connected) except for the decision layer in which the soft-max is used.

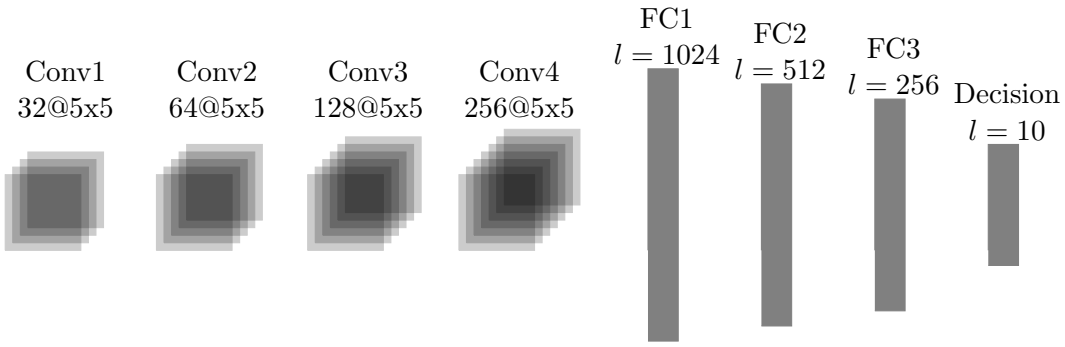


Figure 1: Basic architecture composed of 4 convolutional layers and 4 fully connected layers.

Also, after each one of the four convolutional layers it is applied a 2x2 max-pooling layer to reduce the amount of parameters and also able the convolutional kernels to find features at different scales of the image.

3.2 Initialization

First of all, to initialize the weights of both kind of layers it is used a method specially designed for networks using ReLUs which is deeply explained in [1]. To make it short, the weights are randomly initialized following the Gaussian distribution seen in (1) where n is the number of inputs that the layer has. In the case of a convolutional layer, n is computed as $kernelHeight * kernelWidth * numInputChannels$.

$$W \sim \mathcal{N}(0, \frac{2}{n}) \tag{1}$$

3.3 Training

In order to train the network it has been selected the cross entropy as the loss function, since it is the most suitable for a multiclass classification problem. As optimizer, the selected has been the Adam since it is one of the most recommended in the literature.[2]

After some trials with different batch sizes (8,16,32,64), the selected one has been 32. The number of epochs is set to 50 so it can be ensured that the network will reach overfitting. It is not a problem because it will be applied a variation of Early Stopping which is explained in subsection 4.1.

4 Regularization

In a few words, regularization can be defined as the techniques applied to a learning algorithm with the objective of increasing its generalization capabilities and reducing the overfitting.

There exist many of these techniques, but in this document only 4 of them will be applied. Due to the lack of time, only L2-Regularization and Dropout will be studied. The other two will be directly applied since they do not interfere with the others.

4.1 Best Model Saving

This technique is a variation of the known Early Stopping[4], which consists on stop the training when the validation error stops decreasing and begins to increase again.

In Best Model Saving the training is performed completely but saving the model in every epoch only if it is better than the current saved model. Using Algorithm 1 it is ensured that, despite occurring overfitting in later epochs of the training, the model will not be overfitted.

$bestValidationAccuracy = 0$;
$epoch = 0$;
while $epoch < maxEpochs$ **do**
    new epoch training;
    compute current validation accuracy;
    if $currentValidationAccuracy > bestValidationAccuracy$ **then**
        $bestValidationAccuracy = currentValidationAccuracy$;
        save model;
    end
    $epoch++$;
end

Algorithm 1: Best Model Saving

4.2 Data Augmentation

The first idea that comes to someone's mind to help to generalize might be "get more training data". The problem comes when it is not possible or not feasible to get more, so data augmentation is a solution that increases the amount of data available by applying some transformations on the current one. This transformations are not universal and it is needed a knowledge on the data we are working with to know if they can be applied.

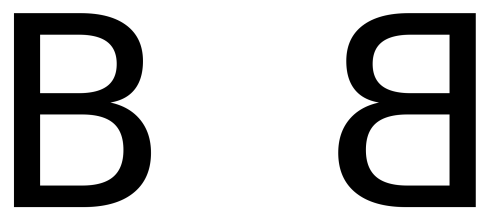
For example, if we want to use horizontal flip it can be done in a automobile classification problem like Figure 2 but not in an alphabet letter classification as in Figure 3.

In this project it has been applied the horizontal flip, since all the classes allow it.

(a) Car

(b) Car

Figure 2: Horizontal flip can be applied



(a) Letter B

(b) This is not letter B

Figure 3: Horizontal flip can not be applied

4.3 L2 Regularization

L2 Regularization is a type of regularization known as Weight Decay. It is based on the assumption that a network with small weights will work better than a network with higher ones. What it does is penalizing the square value of the weights by adding them to the cost function as seen in Equation (2), being N the number of weights and β the "strength" of the regularization.

$$J = J + \beta \sum_{n=1}^{N} W_n^2 \tag{2}$$

In the hyper-parameters it will be tried to find the optimal value for β.

4.4 Dropout

This technique consists on adding a dropout layer after one or more of the network layers. What it does is to, while training, keeping each of the neurons active with some probability p, or setting it to zero otherwise. This forces the other neurons of that layer to do the job

of the ones that have been disabled and in the end increases the individual performance of each one of the neurons.

Dropout can be applied in every layer, even the input. Even though, given the fact that convolutional layers already have a small number of parameters, in this work the dropout has been applied to all the layers but with different p values for the fully connected than for the convolutional.

In the hyper-parameters it will be tried to find the optimal value for p_{fc} and p_c.

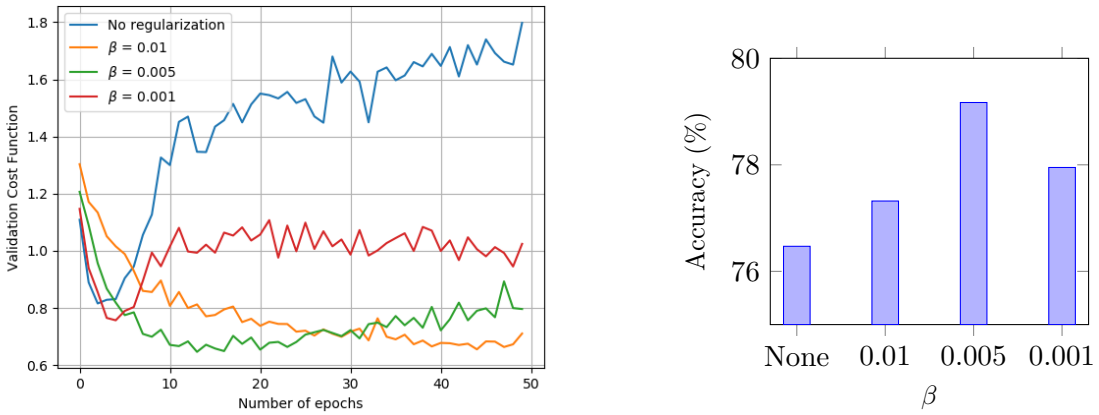
5 Hyper-parameters

In this section the objective is to find the optimal values for the hyper-parameters of the L2 Regularization and the Dropout. They will be found in three different situations When only L2 is applied, when only Dropout is applied and when both are applied.

5.1 Only L2 Regularization

In this situation the only parameter to "tune" is the β. A look into the Equation (2) gives the intuition that if the value is too low it will be as if the regularization technique was not applied, and if it is too high the optimizer will focus on reducing the value of the weights instead of on adapting them to the data.

Due to the fact that there is not much time to check too many values for β some have been selected and evaluated.



(a) Validation Cost Function

(b) Best epoch accuracy

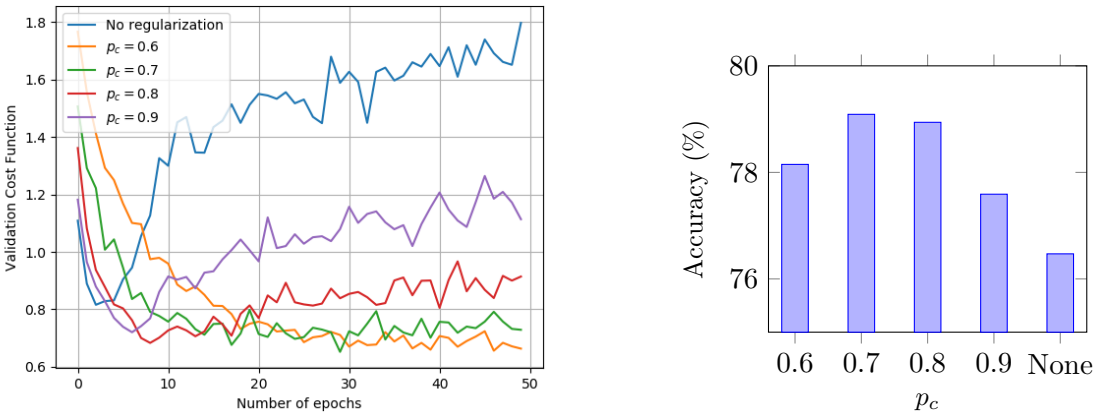
Figure 4: Evaluation of different values of β

5.2 Only Dropout

As it has been explained the dropout is applied in the convolutional layers and the fully connected with different probabilities. So in this situation there are 2 parameters to tune: p_{fc} (for fully connected) and p_c (for convolutional). From the Dropout definition it can be derived that the higher the probabilities, the more similar to unregularized behaviour.

Due to the fact that there is not much time to check too many values for both probabilities, some have been selected and evaluated.

First of all, the intention is to find the optimal value for p_c since the intuition says that in a convolutional neural network the convolutional layers are more important. This is made by setting $p_{fc} = 1.0$ and checking some values for p_c.



(a) Validation Cost Function

(b) Best epoch accuracy

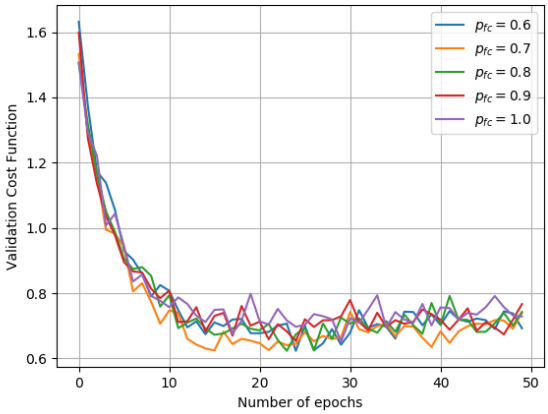
Figure 5: Evaluation of different values of p_c with $p_{fc} = 1.0$

As it can be seen in Figure 5 the optimal value for p_c is 0.7. With this fixed, the next step is to check the optimal value for p_{fc}.

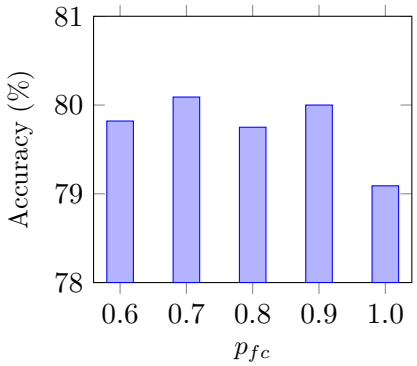
Figure 6 shows that despite being really similar between them, the different values of p_{fc} outperform the case in which it is 1.0, what would mean applying dropout only in the convolutional layers. The value selected has been 0.7 since it provides the highest accuracy.

5.3 L2 and Dropout

In this situation there are 3 parameters to "tune": β, p_{fc} and p_c. The first idea was simply to combine the three values already selected, but the result is an accuracy of around 77%, which is worse than using only the 2 regularization methods separately. This must be because there is an over-regularization which, instead of helping, is counterproductive.



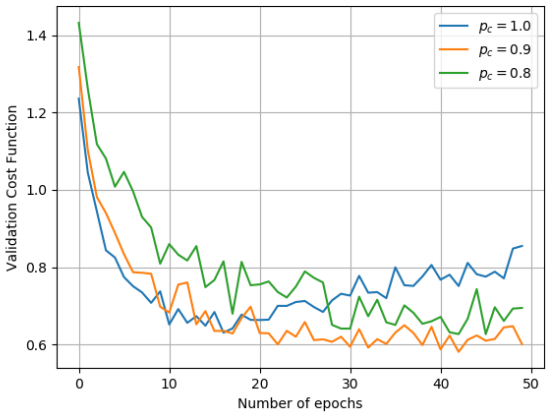
(a) Validation Cost Function



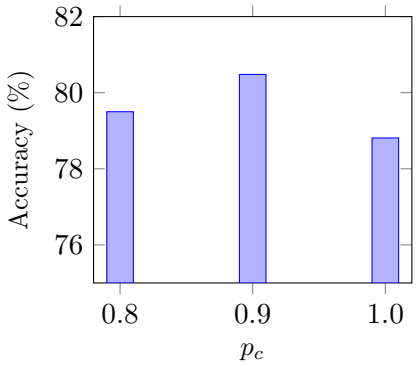
(b) Best epoch accuracy

Figure 6: Evaluation of different values of p_{fc} with $p_c = 0.7$

The solution that has been applied is to set the β to 0.005 (which is the optimal value for only L2) and then find first the optimal p_{fc} and finally the optimal p_c.



(a) Validation Cost Function

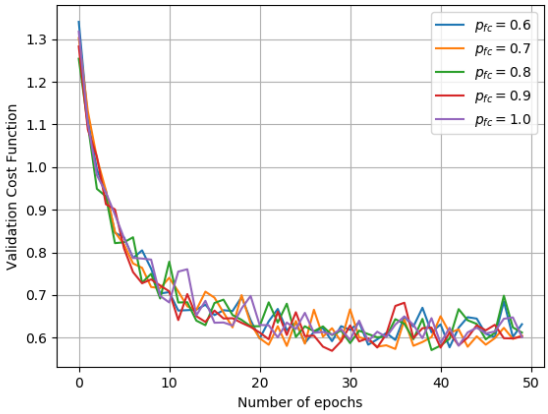


(b) Best epoch accuracy

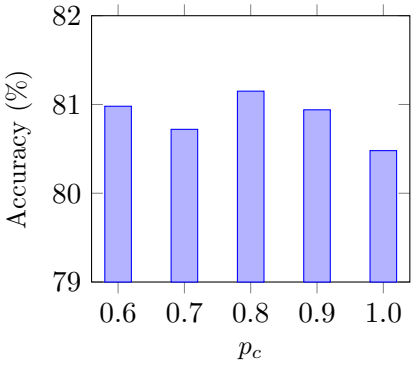
Figure 7: Evaluation of different values of p_c with $p_{fc} = 1.0$ and $\beta = 0.005$

As it can be seen in Figure 7 the best value for p_c is 0.9. Now the next step is to find the optimal value for p_{fc} with the other two parameters set to their respective optimums.

As it is shown in Figure 8 there is not much difference between the different values. The one selected is 0.8 since it provides the maximum validation accuracy.



(a) Validation Cost Function



(b) Best epoch accuracy

Figure 8: Evaluation of different values of p_{fc} with $p_c = 0.9$ and $\beta = 0.005$

6 Final Model

Once all the parameters have been tuned it is time to set the final model and check the test results. With the values of $p_{fc} = 0.8$, $p_c = 0.9$ and $\beta = 0.005$ the model gets the following:

	Train	Validation	Test
Accuracy	96.37%	81.15%	80.03%
Cost Function	0.183[1]	0.571	0.590

7 Conclusions

As a recap of the whole work it can be seen that regularization methods highly improve the performance of deep learning networks.
Another interesting thing to point out is that the majority of regularization methods are not independent between them, and the technique of applying them separately and then combine with the same hyperparameters does not work. They must be applied all together and do the parameters fine-tuning at the same time.

[1]The original value was divided by 4 since there are 4 times more training samples than validation and than test. In this way, the comparison makes more sense.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers:surpassing human-level performance on imagenet classification. *ICCV*, 2015.

[2] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[4] Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. 26:289–315, 08 2007.