# Regularization Strategies in Deep Learning

Sandesh Kumar Srivastava
Department of Computer Science and Engineering
University at Buffalo, The State University of New York
Buffalo, NY 14260-2500
sandeshk@buffalo.edu

## Abstract

*Regularization is a technique which is widely used in machine learning to reduce the generalization error(error on unseen data) but not the training error(error on available data). It is an extremely important concept in training deep learning algorithms as the learned models often exhibit the problem of overfitting and perform exceptionally well on training data but not so well on the test data. Various regularization strategies are already defined which try to address this problem. In this current work, we study some of these regularization strategies namely L1 norm, L2 norm, Data Set Augmentation, Noise robustness, Early Stopping, Dropout, Adversarial Training along with some of their combinations. We discuss the implementation details of these regularization strategies using Keras library in Python. Based on our implementation, we then show the results obtained on the test data of the popular CIFAR-10 dataset. Our experiment shows that regularization helps to achieve an improvement from* **77.85%** *to* **84.81%** *in terms of accuracy and* **77.896%** *to* **84.854%** *in terms of precision on the test data. We also demonstrate the importance of hyperparameter tuning in regularization by studying the L2 norm case.*

## 1. Introduction

As deep learning algorithms become substantially complex, they tend to perform extremely efficiently on the training data and in practice often achieve an accuracy which is close to 100% on training data. Even though it might be tempting to stick with such a model, it is generally observed that such a high performing model/algorithm is not able to replicate the same performance on the test or unseen data. In such situations, we say that the model is overfitting[3] i.e. it has learned features which are only true for the training data and not so for the general data under consideration. To address this problem, we use regularization techniques which are aimed at reducing the generalization error even at the expense of training error. The definition of regularization has been further broadened in [5] to *any supplementary technique that aims at making the model generalize better i.e. produce better results on the test set*.This can include various properties of the loss function, the loss optimization algorithm, or other techniques.

Now that we realize the importance of regularization in deep learning algorithms, the next question to tackle is what are the standard methods of regularization. Not surprisingly, there are many regularization techniques which are very effective and popular in deep learning world. These methods or techniques of regularization tend to put some constraints on the machine learning(ML) model or the objective function to achieve regularization.

We begin by briefly covering the idea behind some of these regularization techniques in section 2. Section 3 covers the implementation details of these techniques in Keras library and we present the results obtained in our experiment in Section 4. Finally, concluding remarks are provided in section 5.

## 2. Regularization Strategies

The main idea behind different regularization techniques is described below:

### 2.1. L1 Regularization

L1 regularization belongs to the class of regularization which put constraint on the objective function J by adding a parameter norm penalty $\Omega(\Theta)$to it. We denote the regularized objective function by $\tilde{J}$:

$$\tilde{J}(\Theta; X, y) = J(\Theta; X, y) + \alpha\Omega(\Theta) \tag{1}$$

For L1 norm regularization the norm penalty $\Omega(\Theta)$ is defined as the sum of the absolute values of parameters:

$$\Omega(\Theta) = ||w||_1 = \sum_i |w_i|_1 \tag{2}$$

So, the regularized objective function $\tilde{J}$ for L1 norm is given by:

$$\tilde{J}(\Theta; X, y) = J(\Theta; X, y) + \alpha \sum_i |w_i|_1 \quad (3)$$

The L1 penalty induces sparsity property and causes a subset of weights to become zero, suggesting that those features can be discarded.

### 2.2. L2 Regularization

L2 regularization is the simplest and most commonly used norm penalty regularization technique[1] which means that the objective function J is constrained as defined in equation (1). It is commonly referred as the *ridge regression* or *Tikhonov regularization*

For L2 regularization the norm penalty $\Omega(\Theta)$ is defined as the sum of the squared values of parameters:

$$\Omega(\Theta) = \frac{1}{2}||w||_2^2 \quad (4)$$

So, the regularized objective function $\tilde{J}$ for L2 norm is given by:

$$\tilde{J}(\Theta; X, y) = J(\Theta; X, y) + \alpha \frac{1}{2}||w||_2^2 \quad (5)$$

### 2.3. Data Set Augmentation

This regularization technique is based on the idea that in order to reduce generalization error we need to train the machine learning model on more data. We obtain more data by transforming the given input to obtain new input. Data set augmentation is very effective for the classification problem of object recognition as images are high dimensional and include a variety of variations[3]. So it is easy to simulate new input images by performing operations such as translation, rotation, scaling etc.. However, while applying data set augmentation we should be careful as to not apply a transformation which can change the label of the input like for instance in the digit recognition problem applying a horizontal flip can cause the model to get confused between classes 6 and 9. Data set augmentation is also not suitable for certain problems like density estimation as the problem depends on the original input only.

### 2.4. Noise Robustness

It might appear strange but adding noise while training is an efficient technique which can help achieve regularization. Noise can be applied at different levels to a ML model. If applied at input, it serves as a data augmentation and when applied to output layers, it helps to handle the mistakes made by ML model. However, noise application is most efficient when it is done at hidden layers and it can perform better than simply shrinking the parameters. Adding noise to weights can also be interpreted as a stochastic implementation of Bayesian inference over the weights[3].

### 2.5. Early Stopping

As the name suggests, early stopping is based on the idea of stopping training process whenever there is not significant improvement on the validation data metrics. When training large models the normally observed pattern is that initially both training and validation data performance improves. However, after a certain point, even though the training data performance improves the validation data performance does not improve. This point indicates that the model has started overfitting and we need to stop further training.

### 2.6. Dropout

Dropout training was introduced by Srivastava et al.[4] and can be viewed as a technique similar to bagging which is computationally efficient and hence can be applied to deep neural networks. Bagging is a method of averaging over several models to improve generalization. However, applying bagging directly is extremely inefficient as training multiple neural networks is expensive in terms of time and memory. Dropout makes designing this bagging approach feasible by randomly dropping some hidden and input units by simply multiplying their output value to zero. Dropout can also be viewed as a form of adaptive regularization[2].

### 2.7. Adversarial Training

Even though not a direct regularization technique, Adversarial Training helps in addressing one of the key challenges ML model face today which is their poor performance on adversarial examples. In order to overcome this problem, ML models are explicitly trained on the intentionally generated adversarial examples. Adversarial examples can be easily generated by applying small perturbations to the training dataset examples. There are various methods for generating these adversarial examples like *Fast Gradient Sign Method(FGSM)*, *Basic Iterative Method(BIM)* and *Projected Gradient Descent(PGD*. In this paper, we utilize FGSM method which creates adversarial examples by adding an imperceptibly small vector, equal to the sign of the elements of the gradient of the cost function, to the training dataset input *x*

$$x \rightarrow x + \epsilon sign(\nabla_x J(\Theta, x, y) \quad (6)$$

## 3. Implementation in Keras

We have implemented all the above mentioned regularization techniques in Python using Keras library APIs. We briefly mention the APIs along with some important attributes and other implementation details below.

### 3.1. L1 Regularization

Regularization penalties in Keras are applied on a per-layer basis so the exact API will depend on the layer like *Conv2D, Dense* etc.. These layers expose 3 keyword parameters $kernel\_regularizer$, $bias\_regularizer$ and $activity\_regularizer$ which indicate whether the penalty is to be applied to layer's kernel, bias or output respectively.

For L1 regularization the API is simply called *l1* and it has one attribute *l1* which indicates the regularization factor $\alpha$ mentioned in equation(1). Sample python code for using l1 norm regularization in Keras is shown here:

```
Conv2D(filters=f_5x5, kernel_size=(5,5),
padding='same', activation='relu',
kernel_regularizer=l1(l1=0.01),
bias_regularizer=l1(1e-4),
activity_regularizer=l1(1e-5)
```

### 3.2. L2 Regularization

The API for L2 regularization is exactly similar to L1 with the only difference being that it is called *l2* instead of *l1* and the regularization factor attribute is also called *l2*.

### 3.3. L1L2 Regularization

Keras also provides a single API called *l1_l2* which can apply both regularization penalties to a given layer. It takes two regularization factor attributes namely *l1* and *l2*.

### 3.4. Data Set Augmentation

Keras provides *ImageDataGenerator* class in its preprocessing library which can be used to generate batches of tensor image with real-time data augmentation. There are numerous attributes defined which can be used to specify the kind of augmentation desired like *zca_whitening, width_shift_range, height_shift_range, horizontal_flip, vertical_flip* etc.

### 3.5. Noise Robustness

Keras provides a layer *GaussianNoise* which can be used to apply additive zero-centered Gaussian noise. We can add this layer to the ML model at the point where we want to introduce noise. Being a regularization layer, it is only active during training time. It takes a float attribute which is the standard deviation of the noise distribution.

### 3.6. Early Stopping

Keras provides a callback API(which can be called during different stages of training) called *EarlyStopping* to stop training when a monitored metric has stopped improving. Some important attributes of this API are *monitor* which is used to specify the quantity to be monitored like validation accuracy, loss etc. and *patience* which indicates the number of epochs with no improvement after which training will be stopped.

### 3.7. Dropout

Keras provides *Dropout* layer which randomly sets input units to 0 with a frequency equal to the attribute value *rate*. Inputs not set to 0 are scaled up such that the sum over all inputs is unchanged. As with other regularization techniques, Dropout only applies during the training period.

### 3.8. Adversarial Training

Keras does not provide any direct way of creating adversarial examples for training ML model. So we create our own implementation of equation (6) by calculating the sign of the gradient and use it to calculate the perturbations.

### 3.9. CIFAR-10 dataset

We have used the CIFAR-10 dataset which consists of images belonging to 10 classes. There are 50000 training images and 10000 test images which are evenly distributed among the 10 classes. Each image is a colourful image of size 32x32. The 10 classes are namely *airplane, automobile, bird, cat, deer, dog, frog, horse, ship* and *truck* and these classes are completely mutually exclusive.

### 3.10. ML Model

For comparison amongst different regularization strategies, we use same ML model for our experiments with the only difference being the regularization strategy used with the ML model. We have used an Inception like model[6] which is created using the basic Keras layers. We set the base performance using this ML model as the *No Regularization* run and compare the improvement of regularization techniques with this value.

The main method for creating the model is *create_InceptionV2Model ()* which also utilizes a helper method called *inceptionBuilder ()*. This utility function passes the input to 1x1, 3x3, 5x5 kernel size *Conv2D* layers and *MaxPool2D* and then concatenates these outputs together. We invoke the *inceptionBuilder()* module three times to add the micro-architecture thrice in our main architecture. We then perform *GlobalAveragePolling2D* followed by a final *Dense* output layer using *'softmax'* activation. More complex architecture can be formed by increasing the calls to the *inceptionBuilder()* module with required number of filter parameters.

### 3.11. Other implementation details

We perform standard data preprocessing steps to the input before feeding it to the ML model. The preprocessing steps are same across all cases and consists of normalizing the input image using mean and standard deviation. For

compiling the model, we use *Adamax* compiler. We also use Keras callback API *ModelCheckPoint* to store the best model weights by monitoring the validation data accuracy.

## 4. Results

We perform the experiments for a batch size of 100 for 200 *epochs* for all cases. *No Regularization* case serves as base for comparison and does not use any regularization technique. Table 1 shows the different attribute values used for each regularization technique to obtain the results.

| Regularization Strategy | Attribute values |
|---|---|
| *No Regularization* | N/A |
| *L1* | kernel_regularizer=1e-2<br>bias_regularizer=1e-4<br>activity_regularizer=1e-5 |
| *L2* | kernel_regularizer=1e-3<br>bias_regularizer=1e-4<br>activity_regularizer=1e-5 |
| *L1L2* | kernel_regularizer=l1_l2<br>(l1=1e-3, l2=1e-3) |
| *Dataset Augmentation* | zca_epsilon=1e-6<br>rotation_range=10<br>width_shift_range=0.1<br>height_shift_range=0.1<br>horizontal_flip=True |
| *Noise Robustness* | GaussianNoise<br>stdev=0.005 |
| *Early Stopping* | monitor='val_accuracy'<br>patience=40 |
| *Dropout* | rate=0.1 |
| *Adversarial Training(FGSM)* | $\alpha = 0.005$ |
| *Combination* | L2 & Dataset<br>Augmentation & Dropout |

Table 1. Attribute values of regularization techniques

Table 2 shows the precision and accuracy observed on the test data of CIFAR-10 dataset by using the different regularization strategies. We notice that *L1, L2, L1L2, Dataset Augmentation, Noise Robustness, Dropout and Combination* achieve better precision and accuracy than the *No Regularization* case. *Early Stopping* and *Adversarial Training(FGSM)* methods achieve performance which is comparable to the *No Regularization* case. However, these 2 methods provide additional advantage. *Early Stopping* achieves the accuracy in only 168 epochs compared to 200 epochs used by all methods so it is fast in training the ML model. *Adversarial Training(FGSM)* method trains the ML model on adversarial examples as well and so it achieves better security and is more robust in handling adversarial examples. We have used a *Combination* method, which combines 3 regularization techniques namely *L2, Dataset Augmentation and Dropout*, to depict how different regularization techniques can be combined together. As can be seen from the results in Table 2, *Combination* method provides best results and achieves precision of **84.854%** and accuracy of **84.81%** on test data.

| Regularization Strategy | Precision | Accuracy |
|---|---|---|
| *No Regularization* | 0.77896 | 0.7785 |
| *L1* | 0.78399 | 0.7841 |
| *L2* | 0.79902 | 0.8005 |
| *L1L2* | 0.79272 | 0.7923 |
| *Dataset Augmentation* | 0.82933 | 0.8301 |
| *Noise Robustness* | 0.78409 | 0.7803 |
| *Early Stopping* | 0.77789 | 0.7789 |
| *Dropout* | 0.79751 | 0.7961 |
| *Adversarial Training(FGSM)* | 0.77787 | 0.777 |
| *Combination* | 0.84854 | 0.8481 |

Table 2. Precision and Accuracy on test data using different regularization strategies

Results mentioned in Table 2 are obtained using the attribute values listed in Table 1. We arrived at these attribute values through our experiments. It should be noted that these attribute, like other hyperparameters, play vital role in the effect of regularization on the ML model. To illustrate this point, we present results obtained for *L2* regularization with different values of *kernel_regularizer* in Table 3.

| *kernel_regularizer* | Precision | Accuracy |
|---|---|---|
| 1e-1 | 0.01 | 0.1 |
| 1e-2 | 0.504229 | 0.5173 |
| 1e-3 | 0.79902 | 0.8005 |
| 1e-4 | 0.77536 | 0.7759 |

Table 3. Effect of attribute value *kernel_regularizer* on performance of *L2* regularization

By inspecting the values obtained in Table 3, we can appreciate the effect of regularization factor $\alpha$ mentioned in equation(1) on the performance of the ML model. When $\alpha$ is large as in 1e-1, the parameter norm penalty $\Omega(\Theta)$ is so large that the model is not able to learn efficiently and as $\alpha$ is reduced, the effect of regularization on the ML model becomes evident and provides optimal performance at $\alpha = 1e-3$. Reducing $\alpha$ further pushes the model performance towards the *No Regularization* case. As we demonstrate the effect of hyperparameter *kernel_regularizer* on *L2* regularization, similarly other hyperparameters of other regularization techniques effect their performance. We should, therefore, experimentally determine optimal values of all the hyperparameters involved.

## 5. Conclusion

In this paper, we state the problem of overfitting which is very commonly observed in deep learning ML models and then discuss various regularization strategies which can help to address it. We discuss the fundamental idea behind these regularization methods and the corresponding Keras library APIs which can help to implement them along with relevant parameters. We show the results obtained by using these regularization techniques on the test data of CIFAR-10 dataset and observe that all regularization strategies provide some benefits over the *No Regularization* case. We also demonstrate the importance of hyperparameter tuning and its effect on the performance of regularization strategy selected.

The work presented here should help us to identify whether the ML model is overfitting or not. Once overfitting is detected, we can decide and select which regularization technique(s) will be most beneficial for the deep learning problem that we are working on . As we observed in the results obtained, it is extremely common to use a combination of regularization techniques to achieve best generalization results.

## References


[1] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. L2 regularization for learning kernels, 2012. arXiv:1205.2653. 2

[2] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Dropout training as adaptive regularization, 2013. https://papers.nips.cc/paper/2013/file/38db3aed920cf82ab059bfccbd02be6a-Paper.pdf. 2

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 1, 2

[4] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. arXiv:1207.0580. 2

[5] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy, 2017. arXiv:1710.10686. 1

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. arXiv:1409.4842. 3