Lecture-6

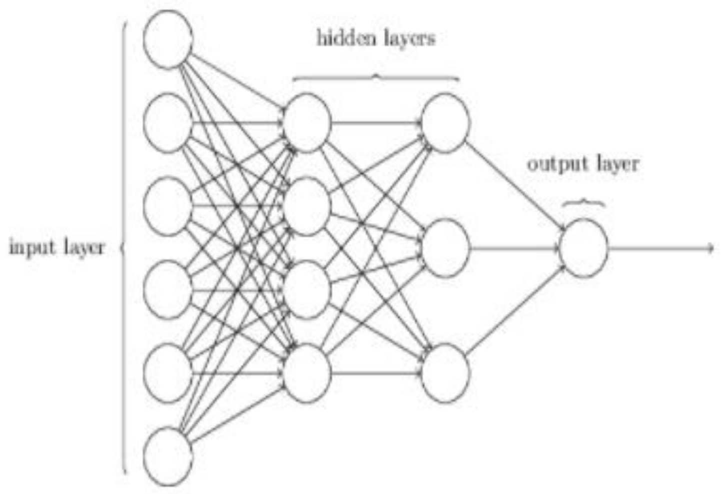
Regularizer and Loss Function

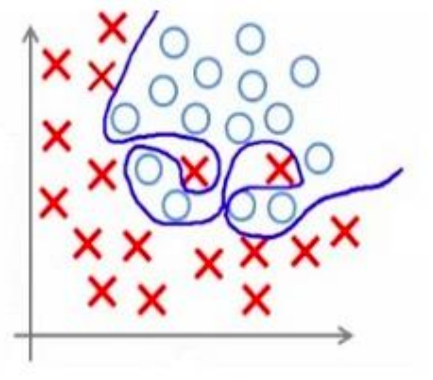
Submitted by:- Harish kumar(15MI421) , Aman Choudhary(15MI408)

Regularizer :- **Regularization** is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in turn improves the model's performance on the unseen data as well.

How does Regularization help reduce Overfitting?

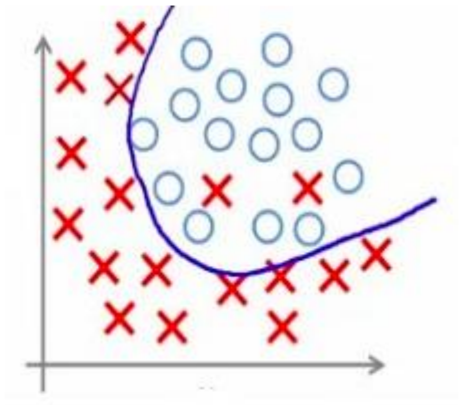
Let's consider a neural network which is overfitting on the training data.





Over-fitting

You will have a fair idea that regularization penalizes the coefficients. In deep learning, it actually penalizes the weight matrices of the nodes. Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero. This will result in a much simpler linear network and slight underfitting of the training data. Such a large value of the regularization coefficient is not that useful. We need to optimize the value of regularization coefficient in order to obtain a well-fitted model as shown in the image below.



Different Regularization Techniques in Deep Learning

1. L1 regularization (also called Lasso)
2. L2 regularization (also called Ridge)
3. L1/L2 regularization (also called Elastic net)

L1 regularization (also called Lasso)

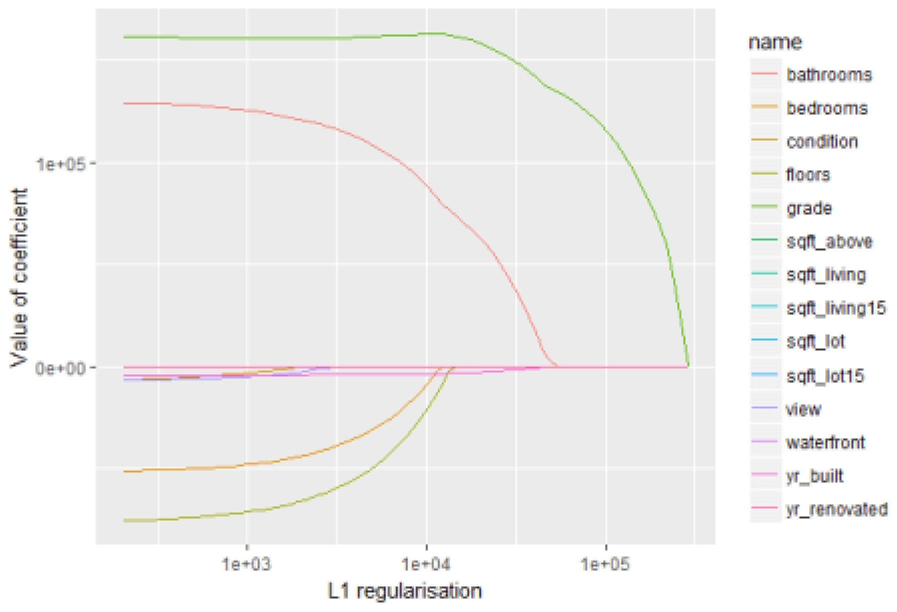
The L1 regularization adds a penalty equal to the sum of the absolute value of the coefficients.

$$\text{Loss} = \text{Loss}(\text{Data}|\text{Model}) + \lambda\Sigma w_n$$

If λ=0, no penalty- overfit
If λ=1 or high, very high penalty- underfit

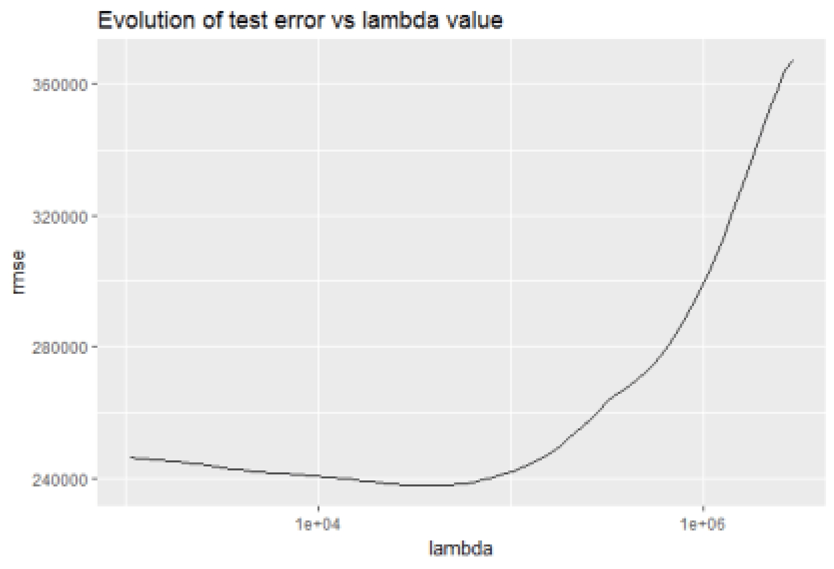
The L1 regularization will **shrink some parameters to zero**. Hence some variables will not play any role in the model, L1 regression can be seen as a way to select features in a model. Let's see this with an example!
We want to predict the prices of houses from House Sales in King County, USA dataset on Kaggle. We will train the data on 0.5% of the dataset, we are taking such a small dataset to ensure that they will be overfitting. The test will be done on the other 99.5% of the data.

We will use the glmnet packages to set a penalization and to test several lambda values. As lambda grows bigger, more coefficient will be cut. Below is the evolution of the value of the different coefficients while lambda is growing.



.

As expected, coefficients are cut one by one until no variables remain. Let's see how the test error is evolving:



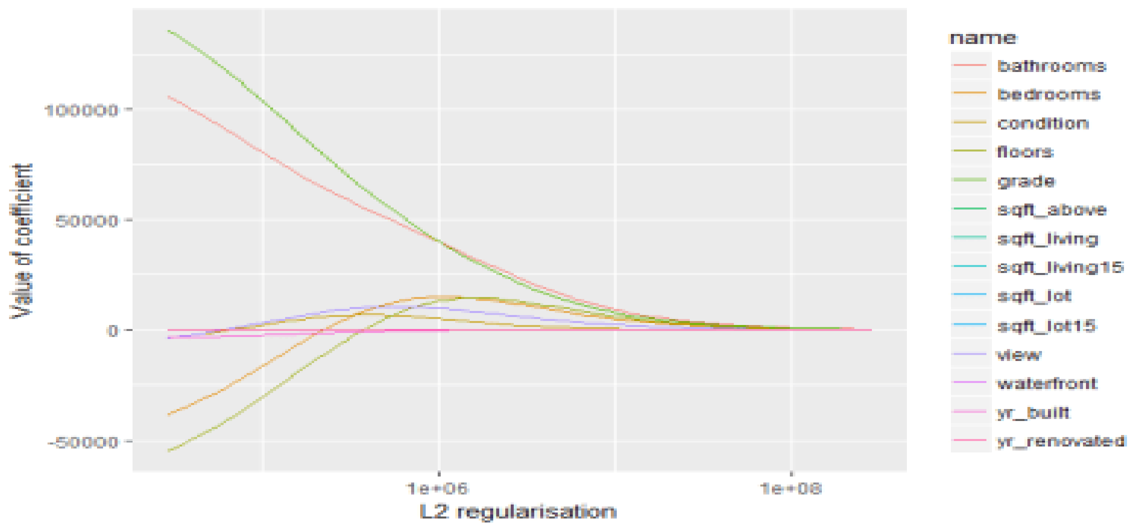
At the beginning, cutting coefficient reduces the overfitting and the generalization abilities of the model. Hence, the test error is decreasing. However, as we are cutting more and more coefficient, the test error start increasing. The model is not able to learn complex pattern with so few variables.

L2 Regularization (Ridge penalisation)

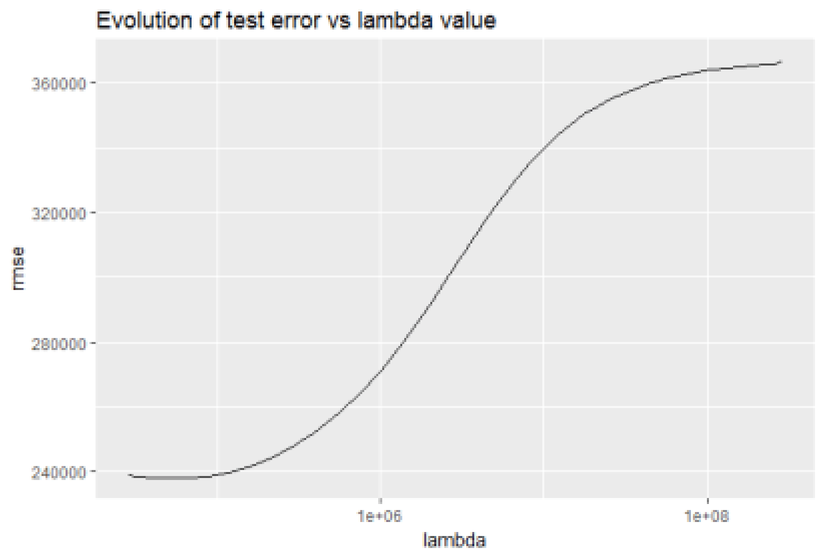
The L2 regularization adds a penalty equal to the sum of the squared value of the coefficients.

$$\text{Loss} = \text{Loss}(\text{Data}|\text{Model}) + \lambda \Sigma w^2_n$$

The L2 regularization will force the parameters to be relatively small, the bigger the penalization, the smaller (and the more robust) the coefficients are.



When we compare this plot to the L1 regularization plot, we notice that the coefficients decrease progressively and are not cut to zero. They slowly decrease to zero. That is the behavior we expected. Let’s see how the test error evolves:



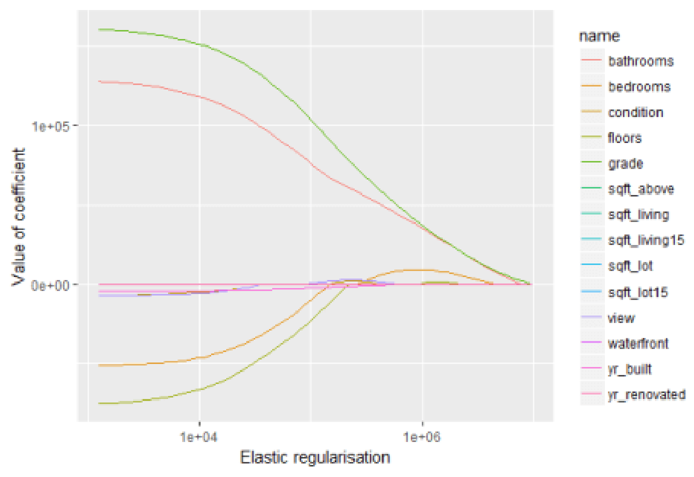
On this case, the L2 regularization is more efficient than the L1 regularization with an error below 240 000.

Elastic-net

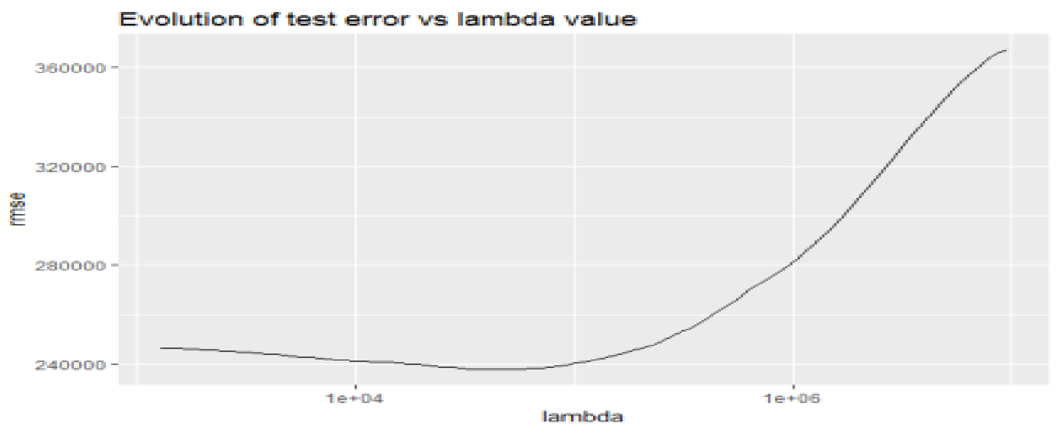
Elastic-net is a mix of **both L1 and L2 regularizations**. A penalty is applied to the sum of the absolute values and to the sum of the squared values:

$$\text{Loss} = \text{Loss}(\text{Data}|\text{Model}) + \lambda\Sigma w_n^2 + \Sigma w_n$$

Lambda is a shared penalization parameter while alpha sets the ratio between L1 and L2 regularization in the Elastic Net Regularization. Hence, we expect a hybrid behavior between L1 and L2 regularization.



And that's happening: Though coefficients are cut, the cut is less abrupt than the cut with lasso penalization alone. The Elastic Net error does not seem to improve the L2 our previous model.



Loss Function

Loss function is an important part in artificial neural networks, which is used to measure the inconsistency between predicted value (^y) and actual label (y). It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function.

Loss functions can be classified into two major categories :-

Regression losses

Classification losses.

In classification, we are trying to predict output from set of finite categorical values i.e given large data set of images of hand written digits, categorizing them into one of 0–9 digits. Regression, on the other hand, deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of room.

Mean Square Error/Quadratic Loss/L2 Loss

Mathematical formulation :-

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

As the name suggests, *Mean square error* is measured as the average of squared difference between predictions and actual observations. It's only concerned with the average magnitude of error irrespective of their direction. However, due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions. Plus MSE has nice mathematical properties which makes it easier to calculate gradients.

Mean Absolute Error/L1 Loss

Mathematical formulation :-

$$MAE = \frac{\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid}{n}$$

Mean absolute error, on the other hand, is measured as the average of sum of absolute differences between predictions and actual observations. Like MSE, this as well measures the magnitude of error without considering their direction. Unlike MSE, MAE needs more complicated tools such as linear programming to compute the gradients. Plus MAE is more robust to outliers since it does not make use of square.

Mean Bias Error

This is much less common in machine learning domain as compared to it's counterpart. This is same as MSE with the only difference that we don't take absolute values. Clearly there's a need for caution as positive and negative errors could cancel each other out. Although less accurate in practice, it could determine if the model has positive bias or negative bias.

Mathematical formulation :-

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

Hinge Loss/Multi class SVM Loss

In simple terms, the score of correct category should be greater than sum of scores of all incorrect categories by some safety margin (usually one). And hence hinge loss is used for maximum-margin classification, most notably for support vector machines. Although not differentiable, it's a convex function which makes it easy to work with usual convex optimizers used in machine learning domain.

Mathematical formulation :-

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

Cross Entropy Loss/Negative Log Likelihood

This is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Mathematical formulation:-

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

Notice that when actual label is 1 (y(i) = 1), second half of function disappears whereas in case actual label is 0 (y(i) = 0) first half is dropped off. In short, we are just multiplying the log of the actual predicted probability for the ground truth class. An important aspect of this is that cross entropy loss penalizes heavily the predictions that are confident but wrong.

Binary Cross-Entropy Loss

Also called **Sigmoid Cross-Entropy loss**. It is a Sigmoid activation plus a Cross-Entropy loss. Unlike Softmax loss it is independent for each vector component (class), meaning that the loss computed for every CNN output vector component is not affected by other component values. That’s why it is used for multi-label classification, were the insight of an element belonging to a certain class should not influence the decision for another class.

Log loss = −(ylog(p)+(1−y)log(1−p))

Squared Hinge Loss

The squared hinge loss is a loss function used for “maximum margin” binary classification problems. Mathematically it is defined as:

$$L(y,\hat{y}) = \sum_{i=0}^{N}(max(0, 1 - y_i \cdot \hat{y}_i)^2)$$

where ŷ the predicted value and y is either 1 or -1.

The hinge loss guarantees that, during training, the classifier will find the classification boundary which is the furthest apart from each of the different classes of data points as possible. In other words, it finds the classification boundary that guarantees the maximum margin between the data points of the different classes.