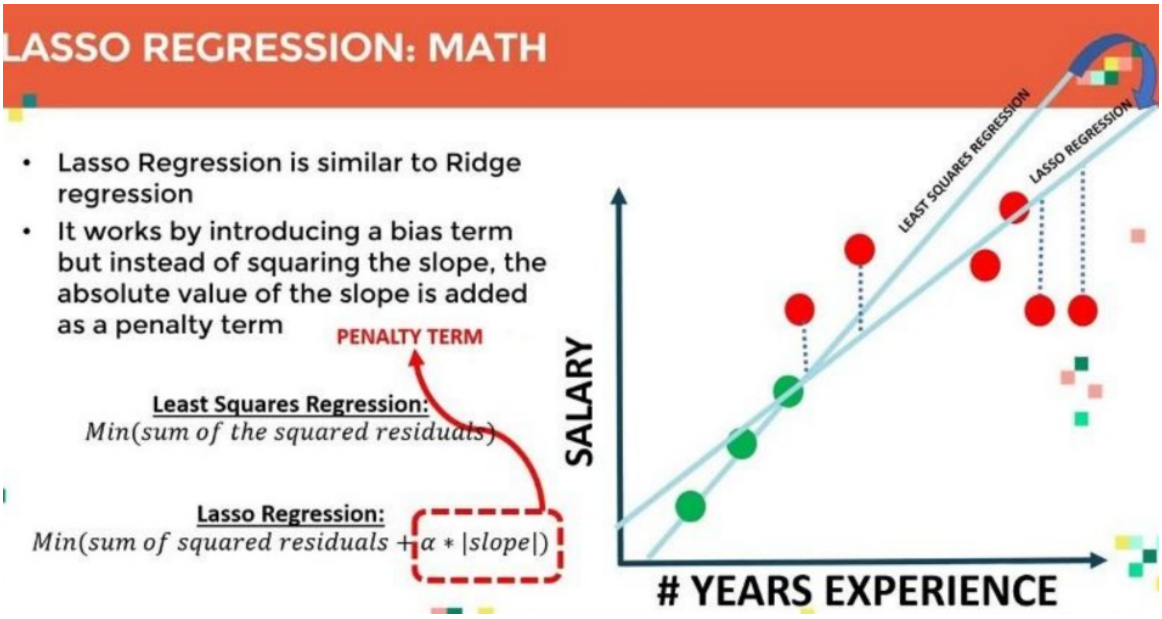
Regularization is a technique used to reduce the errors by fitting the function appropriately on the given training set and avoid overfitting.
The commonly used regularization techniques are:

1. L1 regularization
2. L2 regularization
3. Dropout regularization

A regression model which uses **L1 Regularization** technique is called **LASSO(Least Absolute Shrinkage and Selection Operator)** regression.
A regression model that uses **L2 regularization** technique is called **Ridge regression**.

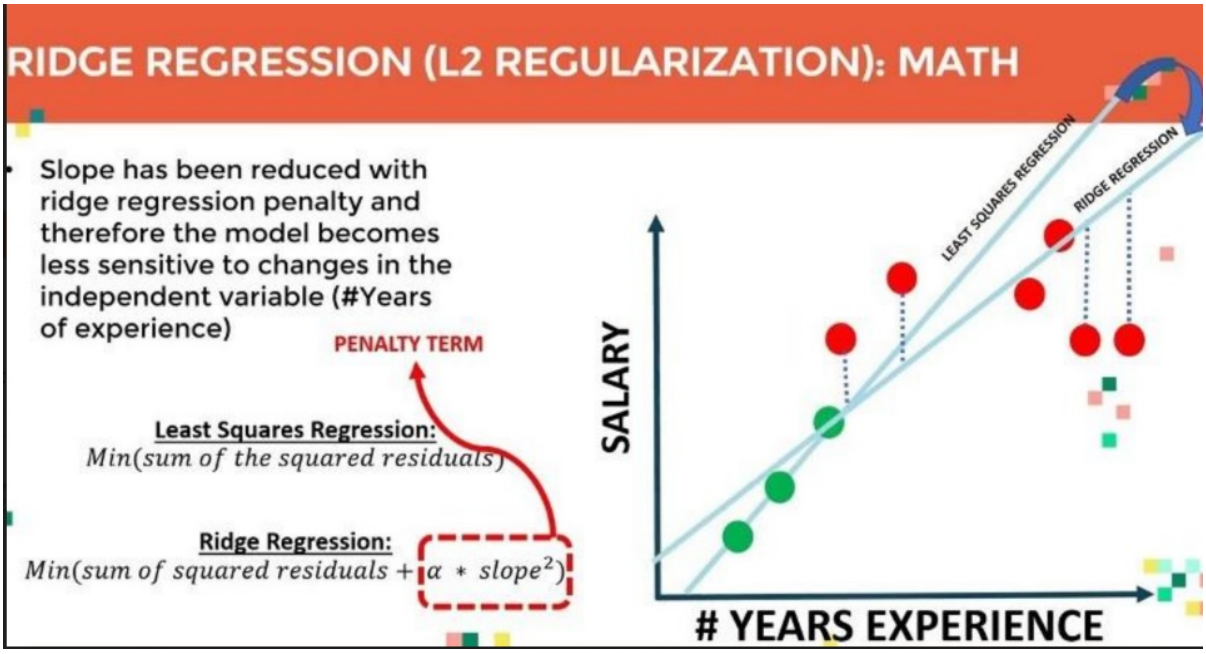
Lasso Regression adds *"absolute value of magnitude"* of coefficient as penalty term to the loss function(L).

$$||\mathbf{w}||_1 = |w_1| + |w_2| + ... + |w_N|$$



Ridge regression adds "*squared magnitude*" of coefficient as penalty term to the loss function(L).
The L2 regularization is the most common type of all regularization techniques and is also commonly known as weight decay or Ride Regression.

$$\|\mathbf{w}\|_2 = \left(|w_1|^2 + |w_2|^2 + \ldots + |w_N|^2\right)^{\frac{1}{2}}$$



NOTE that during Regularization the output function(y_hat) does not change. The change is only in the loss function.
The output function:

$$\hat{y} = w_1 x_1 + w_2 x_2 + ... + w_N x_N + b$$

The loss function before regularization:

$$Loss = Error(y, \hat{y})$$

The loss function after regularization:

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

We define Loss function in Logistic Regression as :

```
L(y_hat,y) = y log y_hat + (1 - y)log(1 - y_hat)
```

Loss function with no regularization :

```
L = y log (wx + b) + (1 - y)log(1 - (wx + b))
```

Lets say the data overfits the above function.

Loss function with L1 regularization :

```
L = y log (wx + b) + (1 - y)log(1 - (wx + b)) + lambda*||w||₁
```

Loss function with L2 regularization :

```
L = y log (wx + b) + (1 - y)log(1 - (wx + b)) + lambda*||w||₂₂
```

lambda is a Hyperparameter Known as regularization constant and it is greater than zero.

```
lambda > 0
```

Elastic Net

Elastic Net combines L1 and L2 With the addition of an alpha Parameter.

- Elastic net combines L1 and L2 with the addition of an alpha parameter deciding the ratio between them:

$$\frac{\sum_{i=1}^{n}(y_i - x_i^J \hat{\beta})^2}{2n} + \lambda \left(\frac{1-\alpha}{2} \sum_{j=1}^{m} \hat{\beta}_j^2 + \alpha \sum_{j=1}^{m} |\hat{\beta}_j| \right)$$

During the L2 regularization the loss function of the neural network as extended by a so-called regularization term, which is called here **Ω**.

$$\Omega(W) = ||W||_2^2 = \sum_i \sum_j w_{ij}^2$$

Eq. 1 Regularization Term

The regularization term **Ω** is defined as the Euclidean Norm (or L2 norm) of the weight matrices, which is the sum over all squared weight values of a weight matrix. The regularization term is weighted by the scalar alpha divided by two and added to the regular loss function that is chosen for the current task. This leads to a new expression for the loss function:

$$\hat{\mathcal{L}}(W) = \frac{\alpha}{2}||W||_2^2 + \mathcal{L}(W) = \frac{\alpha}{2}\sum_i \sum_j w_{ij}^2 + \mathcal{L}(W)$$

Eq 2. Regularization loss during L2 regularization.

Alpha is sometimes called as the **regularization rate** and is an additional hyperparameter we introduce into the neural network. Simply speaking alpha determines how much we regularize our model.
In the next step we can compute the gradient of the new loss function and put the gradient into the update rule for the weights:

$$\nabla_W \hat{\mathcal{L}}(W) = \alpha W + \nabla_W \mathcal{L}(W)$$

$$W_{new} = W_{old} - \epsilon(\alpha W_{old} + \nabla_W \mathcal{L}(W_{old}))$$

Eq. 3 Gradient Descent during L2 Regularization.

Some reformulations of the update rule lead to the expression which very much looks like the update rule for the weights during regular gradient descent:

$$W_{new} = (1 - \epsilon\alpha)W_{old} - \epsilon\nabla_W \mathcal{L}(W_{old})$$

Eq.4 Gradient Descent during L2 Regularization.

The only difference is that by adding the regularization term we introduce an additional subtraction from the current weights (first term in the equation).
In other words independent of the gradient of the loss function we are making our weights a little bit smaller each time an update is performed.

L1 Regularization

In the case of L1 regularization (also knows as Lasso regression), we simply use another regularization term **Ω**. This term is the sum of the absolute values of the weight parameters in a weight matrix:

$$\Omega(W) = ||W||_1 = \sum_i \sum_j |w_{ij}|$$

Eq. 5 Regularization Term for L1 Regularization.

As in the previous case, we multiply the regularization term by alpha and add the entire thing to the loss function.

$$\hat{\mathcal{L}}(W) = \alpha||W||_1 + \mathcal{L}(W)$$

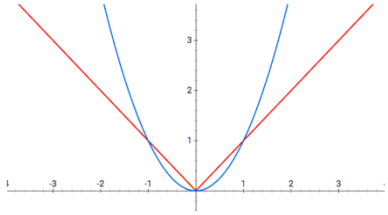
Eq. 6 Loss function during L1 Regularization.

The derivative of the new loss function leads to the following expression, which the sum of the gradient of the old loss function and sign of a weight value times alpha.

$$\nabla_W \hat{\mathcal{L}}(W) = \alpha sign(W) + \nabla_W \mathcal{L}(W)$$

Eq. 7 Gradient of the loss function during L1 Regularization.

Please consider the plots of the and functions, where represents the operation performed during L1 and the operation performed during L2 regularization.



Graph. 2 L1 function (red), L2 function (blue).

In the case of L2 regularization, our weight parameters decrease, but not necessarily become zero, since the curve becomes flat near zero. On the other hand during the L1 regularization, the weight are always forced all the way towards zero.

5. 1 What does Regularization achieve?

- Performing L2 regularization encourages the weight values towards zero (but not exactly zero)
- Performing L1 regularization encourages the weight values to be zero

Intuitively speaking smaller weights reduce the impact of the hidden neurons. In that case, those hidden neurons become neglectable and the overall complexity of the neural network gets reduced.

As mentioned earlier: less complex models typically avoid modeling noise in the data, and therefore, there is no overfitting.

But you have to be careful. When choosing the regularization term **α.** The goal is to strike the right balance between low complexity of the model and accuracy

- If your alpha value is too high, your model will be simple, but you run the risk of *underfitting* your data. Your model won't learn enough about the training data to make useful predictions.
- If your alpha value is too low, your model will be more complex, and you run the risk of *overfitting* your data. Your model will learn too much about the particularities of the training data, and won't be able to generalize to new data.

What's Dropout?

In machine learning, "dropout" refers to the practice of disregarding certain nodes in a layer at random during training. A dropout is a regularization approach that prevents overfitting by ensuring that no units are codependent with one another.

Training with Drop-Out Layers

Dropout is a regularization method approximating concurrent training of many neural networks with various designs. During training, some layer outputs are ignored or dropped at random. This makes the layer appear and is regarded as having a different number of nodes and connectedness to the preceding layer. In practice, each layer update during training is carried out with a different

perspective of the specified layer. Dropout makes the training process noisy, requiring nodes within a layer to take on more or less responsible for the inputs on a probabilistic basis.

According to this conception, dropout may break apart circumstances in which network tiers co-adapt to fix mistakes committed by prior layers, making the model more robust. Dropout is implemented per layer in a neural network. It works with the vast majority of layers, including dense, fully connected, convolutional, and recurrent layers such as the long short-term memory network layer. Dropout can occur on any or all of the network's hidden layers as well as the visible or input layer. It is not used on the output layer.

Why will dropout help with overfitting?

- It can't rely on one input as it might be randomly dropped out.
- Neurons will not learn redundant details of inputs

Other Popular Regularization Techniques

When combating overfitting, dropping out is far from the only choice. Regularization techniques commonly used include:

Early stopping: automatically terminates training when a performance measure (e.g., validation loss, accuracy) ceases to improve.

Weight decay: add a penalty to the loss function to motivate the network to utilize lesser weights.

Noise: Allow some random variations in the data through augmentation to create noise (which makes the network robust to a larger distribution of inputs and hence improves generalization).

Model Combination: the outputs of separately trained neural networks are averaged (which requires a lot of computational power, data, and time).