Regularization in Deep Learning

What is Regularization?

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.

Sometimes the [machine learning](https://www.javatpoint.com/machine-learning) model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "In regularization technique, we reduce the magnitude of the features by keeping the same number of features."

How does Regularization Work?

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

y= β0+β1x1+β2x2+β3x3+⋯+βnxn +b

In the above equation, Y represents the value to be predicted

X1, X2, …Xn are the features for Y.

β0,β1,…..βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

Regularization in Machine Learning

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**

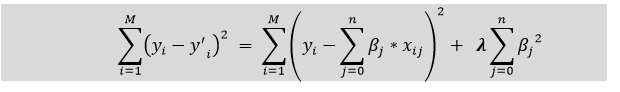
Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

* **Ridge Regression**
* **Lasso Regression**

Ridge Regression

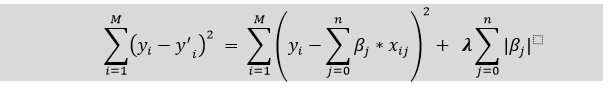
* Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
* Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
* In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
* The equation for the cost function in ridge regression will be:



* In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
* As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.
* A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
* It helps to solve the problems if we have more parameters than samples.

Lasso Regression:

* Lasso regression is another regularization technique to reduce the complexity of the model. It stands for **Least Absolute and Selection Operator.**
* It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
* Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
* It is also called as **L1 regularization.** The equation for the cost function of Lasso regression will be:



* Some of the features in this technique are completely neglected for model evaluation.
* Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

Key Difference between Ridge Regression and Lasso Regression

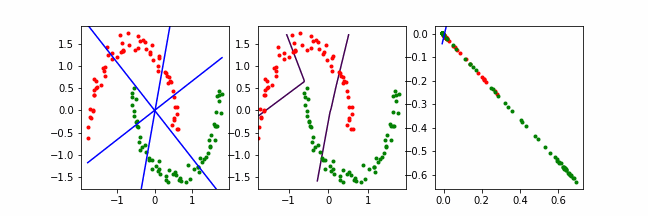
* **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients.
* **Lasso regression** helps to reduce the overfitting in the model as well as feature selection.

Understanding what regularization is and why it is required for machine learning and diving deep to clarify the importance of L1 and L2 regularization in Deep learning.

**Table of contents:**

1. **Introduction**
2. **What is Regularization?**
3. **Regularization in Deep learning**
4. **Parameter Norm penalties**
5. **L1 Regularization**
6. **L2 Regularization**
7. **Differences, usage and importance**
8. **Summary table**
9. **Introduction:**

A universal problem in machine learning has been making an algorithm that performs equally well on training data and any new samples or test dataset. Techniques used in machine learning that have specifically been designed to cater to reducing test error, mostly at the expense of increased training error, are globally known as regularization.



**Isn't fitting a function to our data our end goal?**

**What is Regularization?**

Regularization may be defined as any modification or change in the learning algorithm that helps reduce its error over a test dataset, commonly known as generalization error but not on the supplied or training dataset.

In learning algorithms, there are many variants of regularization techniques, each of which tries to cater to different challenges. These can be listed down straightforwardly based on the kind of challenge the technique is trying to deal with:

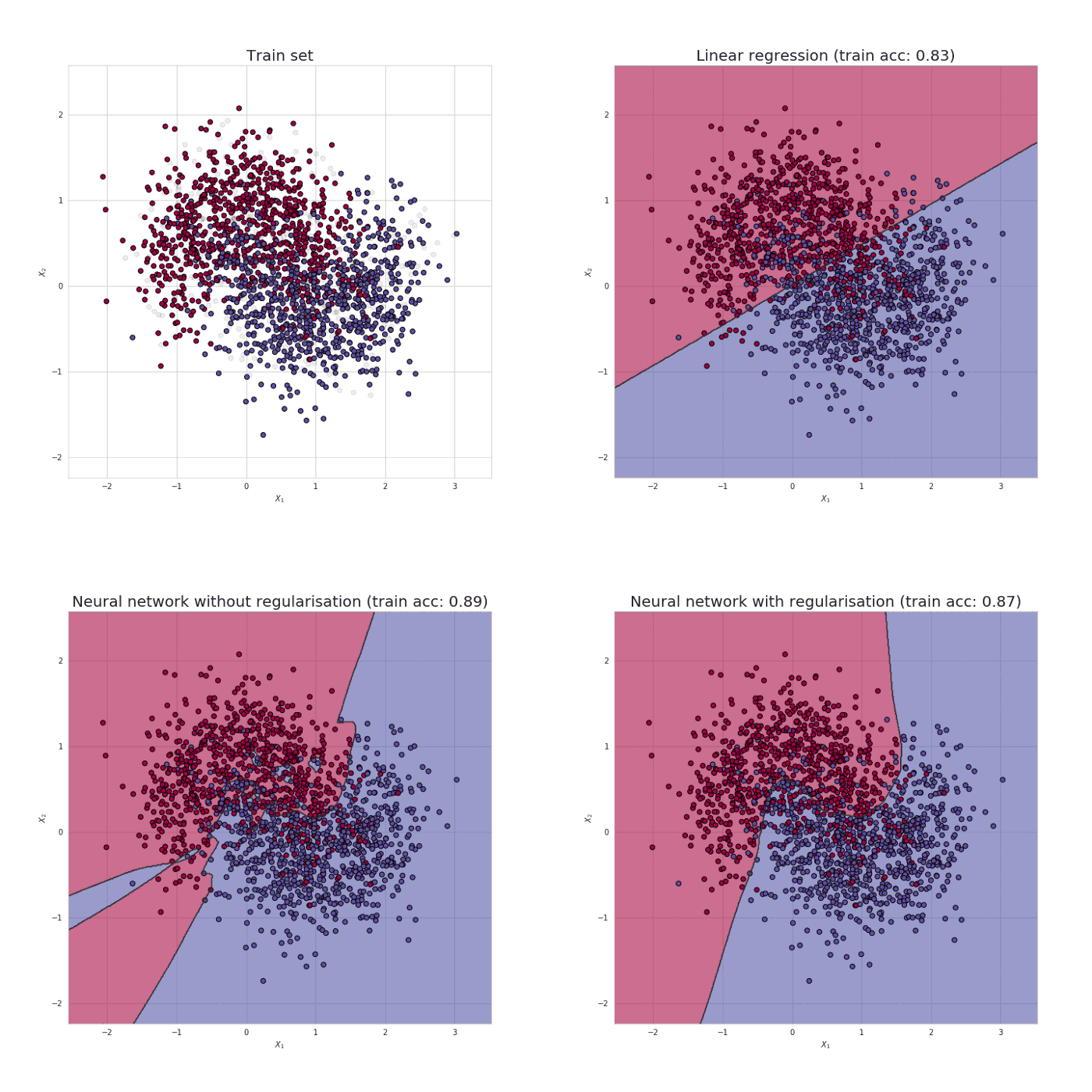
Some try to put extra constraints on the learning of an ML model, like adding restrictions on the range/type of parameter values.

Some add more terms in the objective or cost function, like a soft constraint on the parameter values. More often than not, a careful selection of the right constraints and penalties in the cost function contributes to a massive boost in the model's performance, specifically on the test dataset.

These extra terms can also be encoded based on some prior information that closely relates to the dataset or the problem statement.

One of the most commonly used regularization techniques is creating ensemble models, which take into account the collective decision of multiple models, each trained with different samples of data.

The main aim of regularization is to reduce the over-complexity of the machine learning models and help the model learn a simpler function to promote generalization.

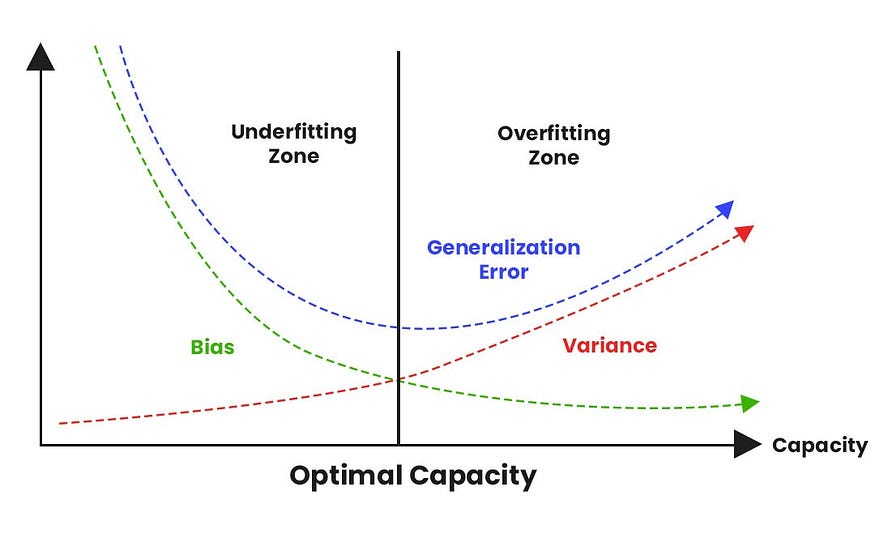


**Impact of Regularization in Machine learnin**g

**Regularization in Deep Learning:**

In the context of deep learning models, most regularization strategies revolve around regularizing estimators. So now the question arises what does regularizing an estimator means?

Bias vs variance trade-off graph here sheds a bit more light on the nuances of this topic and demarcation:



**Bias vs Variance tradeoff graph**

Regularization of an estimator works by trading increased bias for reduced variance.**An effective regularize will be the one that makes the best trade between bias and variance, and the end-product of the trade-off should be a significant reduction in variance at minimum expense to bias.** In simpler terms, this would mean low variance without immensely increasing the bias value.

We consider two scenarios:

* The true data-generating process/function: F1, which created the dataset
* Creating a generating process/function: F2 that mimics F1 but also explores other possible generating scenarios/functions

The work of regularization techniques is to help take our model from F2 to F1 without overly complicating F2. Deep learning algorithms are mostly used in more complicated domains like images, audio, text sequences or simulating complex decision-making tasks.**The True data-generation process: F1 is almost impossible to be correctly mapped, hence with regularization, we aim to bring our model with F2 function as close as possible to the original F1 function.**The following analogy helps understand it more clearly:

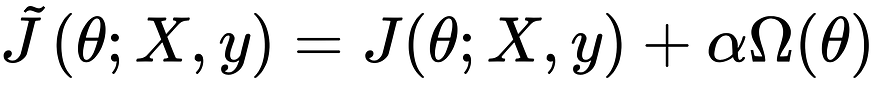
Fitting F2, our ML model, onto F1, our true data generation process is almost like fitting a square-shaped toy in a round hole by closed approximations.

In practical deep learning training and scenarios, we mostly find that the best fitting model (in the sense of least generalization error) is often a large model that has been regularized appropriately.

We will now dive deep to study one type of regularization technique that helps to create a large but deeply regularized model using parameter-based penalties.

**Parameter Norm Penalties:**

Under this kind of regularization technique, the capacity of the models like neural networks, linear or logistic regression is limited by adding a parameter norm penalty Ω(θ) to the objective function J. The equation can be represented as the following:



Parameter Norm penalties

where α lies within [0, ∞) is a hyperparameter that weights the relative contribution of a norm penalty term, Ω, pertinent to the standard objective function J.

**Setting α to 0 results in no regularization while larger values correspond to a greater regularization.**

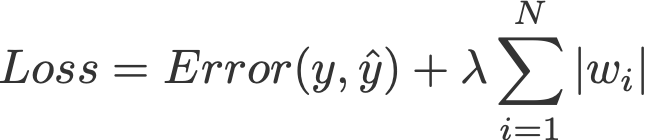
This type of regularization penalizes only the weights of the affine transformation at each layer of the network which leaves the biases unregularized. This is done with the notion in mind that it typically requires lesser data to fit the biases than the weights.

For deep learning, it sometimes feels desirable to use a separate parameter to induce the same affect.

**L1 Parameter Regularization:**

L1 regularization is a method of doing regularization. It tends to be more specific than gradient descent, but it is still a gradient descent optimization problem.

Formula and high level meaning over here:

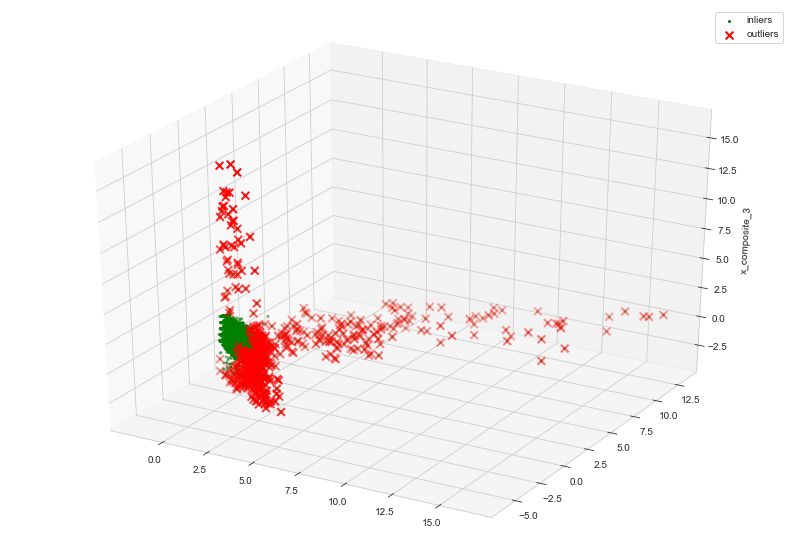


Formula for L1 regularization terms

Lasso Regression (**Least Absolute Shrinkage and Selection Operator**) adds **“Absolute value of magnitude”** of coefficient, as penalty term to the loss function.

Lasso shrinks the less important feature’s coefficient to zero; thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features.

The L1 regularizer basically looks for the parameter vectors that minimize the norm of the parameter vector (the length of the vector). This is essentially the problem of how to optimize the parameters of a single neuron, a single layer neural network in general, and a single layer feed-forward neural network in particular.



Building a ML model while accounting for outliers to be incorporated in the cost penalization is not a trivial task. The image shows visualization of one dummy dataset where L1 helps to identify the outliers in the distant vicinity

A good way of conceptualizing about it is that it is a method of maximizing the area of the parameter hyperspace that the true parameter vector is within. To do this, it finds the “sharpest” edge, one that is as close to the parameter vector as possible. Key points that should be noted for L1 regularization:

**L1 regularization is that it is easy to implement and can be trained as a one-shot thing,**meaning that once it is trained you are done with it and can just use the parameter vector and weights.

**L1 regularization is robust in dealing with outliers.**It creates sparsity in the solution (most of the coefficients of the solution are zero), which means the less important features or noise terms will be zero. It makes L1 regularization robust to outliers.

To understand the above mentioned point, let us go through the following example and try to understand what it means when an algorithm is said to be sensitive to outliers

For instance, we are trying to classify images of various birds of different species and have a neural network with a few hundred parameters.

We find a sample of birds of one species, which we have no reason to believe are of any different species from all the others.

We add this image to the training set and try to train the neural network. This is like throwing an outlier into the mix of all the others. By looking at the edge of the hyperspace where the hyperplane is closest to, we pick up on this outlier, but by the time we’ve got to the hyperplane it’s quite far from the plane and is hence an outlier.

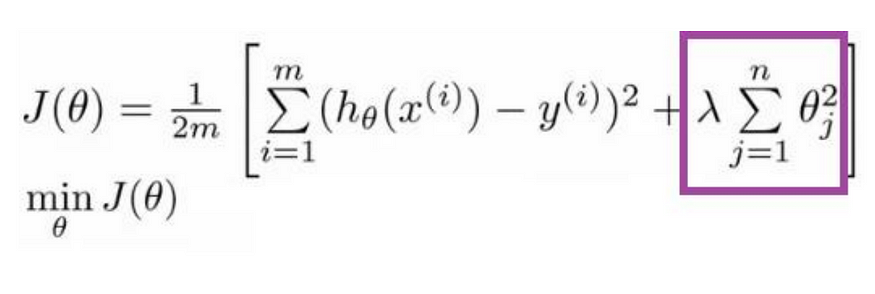
The solution in such cases is to perform iterative dropout. L1 regularization is a one-shot solution, but in the end we are going to have to make some kind of hard decision on where to cut off the edges of the hyperspace.

Iterative dropout is a method of deciding exactly where to cut off. It is a little more expensive in terms of training time, but in the end it might give us an easier decision about how far the hyperspace edges are.

Along with shrinking coefficients, the lasso performs **feature selection**, as well. (Remember the ‘selection‘ in the lasso full-form?) Because some of the coefficients become exactly zero, which is equivalent to the particular feature being excluded from the model.

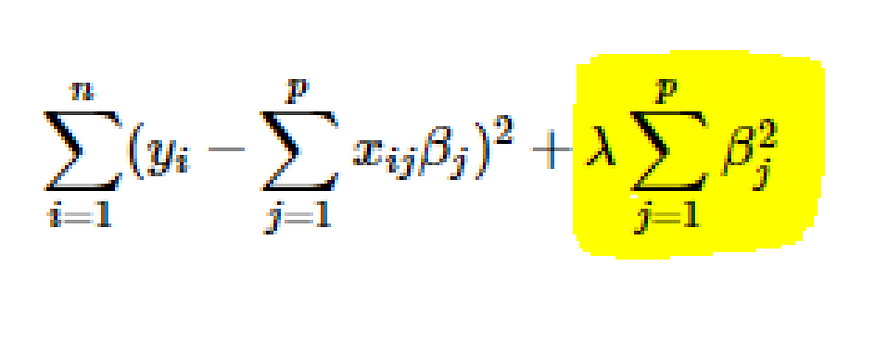
**L2 Parameter Regularization:**

The Regression model that uses L2 regularization is called Ridge Regression.



**Formula for Ridge Regression**

Regularization adds the penalty as model complexity increases. The regularization parameter (lambda) penalizes all the parameters except intercept so that the model generalizes the data and won’t overfit. Ridge regression adds **“squared magnitude of the coefficient”** as penalty term to the loss function. Here the box part in the above image represents the L2 regularization element/term.



Lambda is a hyperparameter.

If lambda is zero, then it is equivalent to OLS.

**Ordinary Least Square or OLS, is a stats model which also helps us in identifying more significant features that can have a heavy influence on the output.**

But if lambda is very large, then it will add too much weight, and it will lead to under-fitting. Important points to be considered about L2 can be listed below:

**Ridge regularization forces the weights to be small but does not make them zero and does not give the sparse solution**.

**Ridge is not robust to outliers** as square terms blow up the error differences of the outliers, and the regularization term tries to fix it by penalizing the weights.

Ridge regression performs better when all the input features influence the output, and all with **weights are of roughly equal size**.

**L2 regularization can learn complex data patterns**

**Differences, Usage and Importance:**

It is important to understand the demarcation between both these methods. In comparison to L2 regularization, L1 regularization results in a solution that is more sparse.

Sparsity in this context refers to the fact that some parameters have an optimal value of zero. The sparsity of L1 regularization is a qualitatively different behavior than arises with L2 regularization

The sparsity feature used in L1 regularization has been used extensively as a feature selection mechanism in machine learning. Feature selection is a mechanism which inherently simplifies a machine learning problem by efficiently choosing which subset of the available features should be used of the model.

Lasso integrates an L1 penalty with a linear model and a least-squares cost function. The L1 penalty causes a subset of the weights to becomes zero, which is safe to suggest that the corresponding features associated with the respective weights, may easily be discarded.

**Regularization as Bayesian inference?**

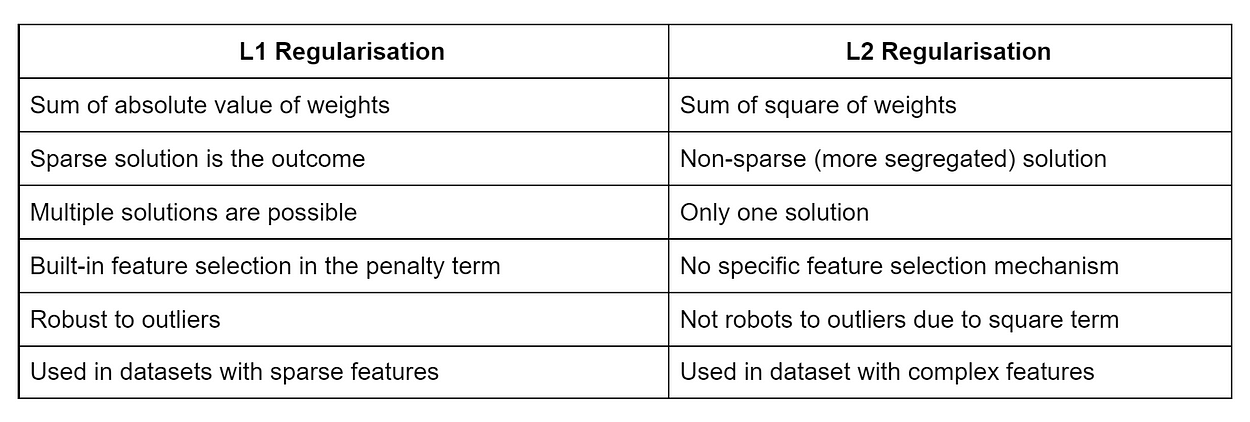
Many regularization techniques can be interpreted as MAP Bayesian inferences.

L2 in particular is almost equivalent to MAP Bayesian inference with a Gaussian prior on the weights.

In L1 regularization, the penalty term used to penalize the cost function can be compared to the log-prior term that is maximized by MAP Bayesian inference when the prior is an isotropic Laplace Distribution over the real number dataset

**Summary table**

The entire post can also be summarized into small bullet points which might be useful during an interview preparation or to skim through the content and just find the right part. Hope this helps:



**Introduction to Semi-Supervised Learning**

Semi-Supervised learning is a type of Machine Learning algorithm that represents the intermediate ground between Supervised and Unsupervised learning algorithms. It uses the combination of labelled and unlabelled datasets during the training period.

Before understanding the Semi-Supervised learning, you should know the main categories of Machine Learning algorithms. Machine Learning consists of three main categories: **Supervised Learning, Unsupervised Learning, and Reinforcement Learning.** Further, the basic difference between Supervised and unsupervised learning is *that*supervised learning datasets consist of an output label training data associated with each tuple,*and*unsupervised datasets do not consist the same. **Semi-supervised learning is an important category that lies between the Supervised and Unsupervised machine learning.** Although Semi-supervised learning is the middle ground between supervised and unsupervised learning and operates on the data that consists of a few labels, it mostly consists of unlabelled data. As labels are costly, but for the corporate purpose, it may have few labels.

The basic disadvantage of supervised learning is that it requires hand-labelling by ML specialists or data scientists, and it also requires a high cost to process. Further unsupervised learning also has a limited spectrum for its applications. **To overcome these drawbacks of supervised learning and unsupervised learning algorithms, the concept of Semi-supervised learning is introduced**. In this algorithm, training data is a combination of both labelled and unlabelled data. However, labelled data exists with a very small amount while it consists of a huge amount of unlabelled data. Initially, similar data is clustered along with an unsupervised learning algorithm, and further, it helps to label the unlabelled data into labelled data. It is why label data is a comparatively, more expensive acquisition than unlabelled data.

We can imagine these algorithms with an example. Supervised learning is where a student is under the supervision of an instructor at home and college. Further, if that student is self-analyzing the same concept without any help from the instructor, it comes under unsupervised learning. Under semi-supervised learning, the student has to revise itself after analyzing the same concept under the guidance of an instructor at college.

**Assumptions followed by Semi-Supervised Learning**

To work with the unlabelled dataset, there must be a relationship between the objects. To understand this, semi-supervised learning uses any of the following assumptions:

**Continuity Assumption:**

As per the continuity assumption, the objects near each other tend to share the same group or label. This assumption is also used in supervised learning, and the datasets are separated by the decision boundaries. But in semi-supervised, the decision boundaries are added with the smoothness assumption in low-density boundaries.

**Cluster assumptions-** In this assumption, data are divided into different discrete clusters. Further, the points in the same cluster share the output label.

**Manifold assumptions-** This assumption helps to use distances and densities, and this data lie on a manifold of fewer dimensions than input space.

The dimensional data are created by a process that has less degree of freedom and may be hard to model directly. **(This assumption becomes practical if high).**

**Working of Semi-Supervised Learning**

Semi-supervised learning uses pseudo labelling to train the model with less labelled training data than supervised learning. The process can combine various neural network models and training ways. The whole working of semi-supervised learning is explained in the below points:

Firstly, it trains the model with less amount of training data similar to the supervised learning models. The training continues until the model gives accurate results.

The algorithms use the unlabelled dataset with pseudo labels in the next step, and now the result may not be accurate.

Now, the labels from labelled training data and pseudo labels data are linked together.

The input data in labelled training data and unlabeled training data are also linked.

In the end, again train the model with the new combined input as did in the first step. It will reduce errors and improve the accuracy of the model.

**Difference between Semi-supervised and Reinforcement Learning.**

Reinforcement learning is different from semi-supervised learning, as it works with rewards and feedback. **Reinforcement learning aims to maximize the rewards by their hit and trial actions, whereas in semi-supervised learning, we train the model with a less labelled dataset.**

Real-world applications of Semi-supervised Learning-

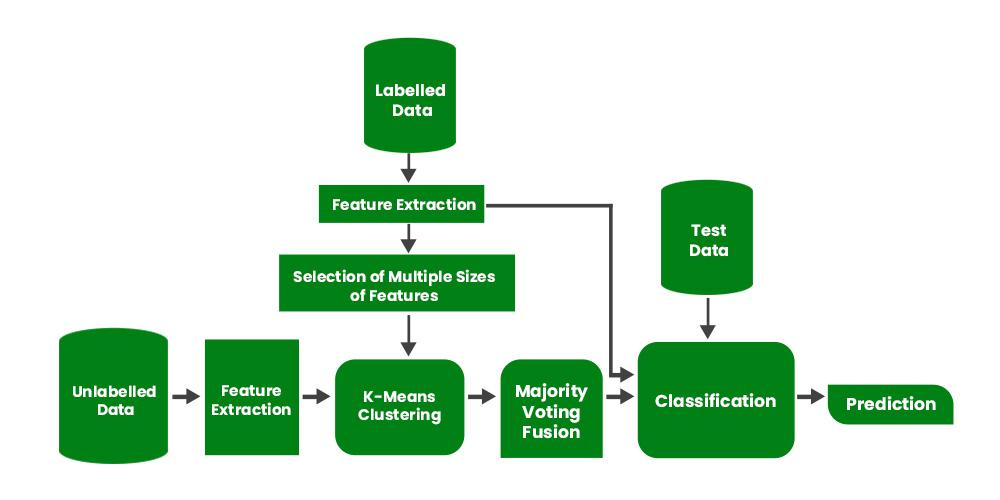
Semi-supervised learning models are becoming more popular in the industries. Some of the main applications are as follows.

**Speech Analysis-** It is the most classic example of semi-supervised learning applications. Since, labelling the audio data is the most impassable task that requires many human resources, this problem can be naturally overcome with the help of applying SSL in a Semi-supervised learning model.

**Web content classification-** However, this is very critical and impossible to label each page on the internet because it needs mode human intervention. Still, this problem can be reduced through Semi-Supervised learning algorithms.  
Further, Google also uses semi-supervised learning algorithms to rank a webpage for a given query.

**Protein sequence classification-** DNA strands are larger, they require active human intervention. So, the rise of the Semi-supervised model has been proximate in this field.

**Text document classifier-** As we know, it would be very unfeasible to find a large amount of labelled text data, so semi-supervised learning is an ideal model to overcome this.



**Introduction to Multi-Task Learning (MTL) for Deep Learning**

Multi-Task Learning (MTL) is a type of machine learning technique where a model is trained to perform multiple tasks simultaneously. In deep learning, MTL refers to training a neural network to perform multiple tasks by sharing some of the network’s layers and parameters across tasks.

In MTL, the goal is to improve the generalization performance of the model by leveraging the information shared across tasks. By sharing some of the network’s parameters, the model can learn a more efficient and compact representation of the data, which can be beneficial when the tasks are related or have some commonalities.

There are different ways to implement MTL in deep learning, but the most common approach is to use a shared feature extractor and multiple task-specific heads. The shared feature extractor is a part of the network that is shared across tasks and is used to extract features from the input data. The task-specific heads are used to make predictions for each task and are typically connected to the shared feature extractor.

Another approach is to use a shared decision-making layer, where the decision-making layer is shared across tasks, and the task-specific layers are connected to the shared decision-making layer.

MTL can be useful in many applications such as natural language processing, computer vision, and healthcare, where multiple tasks are related or have some commonalities. It is also useful when the data is limited, MTL can help to improve the generalization performance of the model by leveraging the information shared across tasks.

However, MTL also has its own limitations, such as when the tasks are very different.

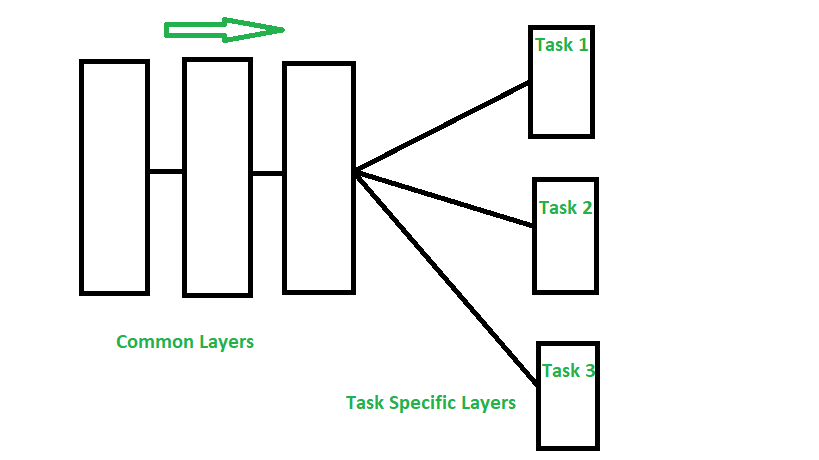
Multi-Task Learning is a sub-field of Deep Learning. It is recommended that you familiarize yourself with the concepts of neural networks to understand what multi-task learning means.

**What is Multi-Task Learning?**

Multi-Task learning is a sub-field of Machine Learning that aims to solve multiple different tasks at the same time, by taking advantage of the similarities between different tasks. This can improve the learning efficiency and also act as a regularizer which we will discuss in a while. Formally, if there are **n** tasks (conventional deep learning approaches aim to solve just 1 task using 1 particular model), where these **n** tasks or a subset of them are related to each other but not exactly identical, Multi-Task Learning **(MTL)** will help in improving the learning of a particular model by using the knowledge contained in all the n tasks. **Intuition behind Multi-Task Learning (MTL):**By using Deep learning models, we usually aim to learn a good representation of the features or attributes of the input data to predict a specific value. Formally, we aim to optimize for a particular function by training a model and fine-tuning the hyperparameters till the performance can’t be increased further. By using MTL, it might be possible to increase performance even further by forcing the model to learn a more generalized representation as it learns (updates its weights) not just for one specific task but a bunch of tasks. Biologically, humans learn in the same way. We learn better if we learn multiple related tasks instead of focusing on one specific task for a long time.

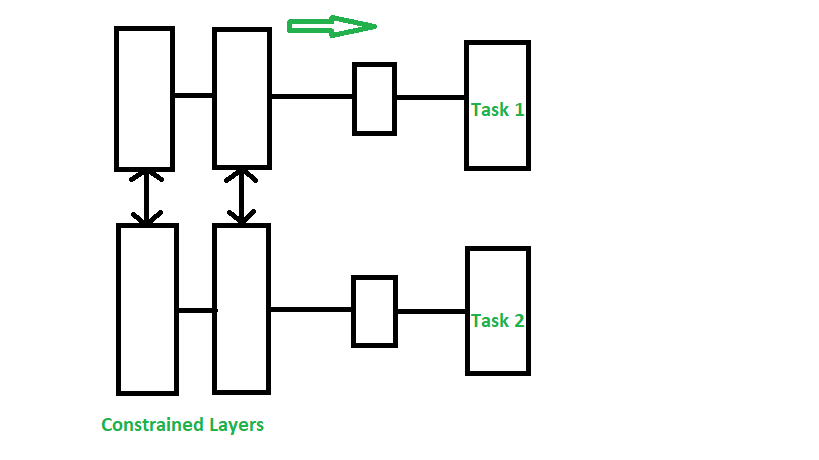
**MTL as a regularizer:**

In the lingo of Machine Learning, MTL can also be looked at as a way of inducing bias. It is a form of inductive transfer, using multiple tasks induces a bias that prefers hypotheses that can explain all the **n** tasks. MTL acts as a regularizer by introducing inductive bias as stated above. It significantly reduces the risk of overfitting and also reduces the model’s ability to accommodate random noise during training. Now, let’s discuss the major and prevalent techniques to use MTL. **Hard Parameter Sharing –**A common hidden layer is used for all tasks but several task specific layers are kept intact towards the end of the model. This technique is very useful as by learning a representation for various tasks by a common hidden layer, we reduce the risk of overfitting.



*Hard Parameter Sharing*

**Soft Parameter Sharing –**Each model has their own sets of weights and biases and the distance between these parameters in different models is regularized so that the parameters become similar and can represent all the tasks.



*Soft Parameter Sharing*

**Assumptions and Considerations –**Using MTL to share knowledge among tasks are very useful only when the tasks are very similar, but when this assumption is violated, the performance will significantly decline.

**Applications:** MTL techniques have found various uses, some of the major applications are-

* Object detection and Facial recognition
* Self-Driving Cars: Pedestrians, stop signs and other obstacles can be detected together
* Multi-domain collaborative filtering for web applications
* Stock Prediction
* Language Modelling and other NLP applications

**Important points:**

Here are some important points to consider when implementing Multi-Task Learning (MTL) for deep learning:

1. Task relatedness: MTL is most effective when the tasks are related or have some commonalities, such as natural language processing, computer vision, and healthcare.
2. Data limitation: MTL can be useful when the data is limited, as it allows the model to leverage the information shared across tasks to improve the generalization performance.
3. Shared feature extractor: A common approach in MTL is to use a shared feature extractor, which is a part of the network that is shared across tasks and is used to extract features from the input data.
4. Task-specific heads: Task-specific heads are used to make predictions for each task and are typically connected to the shared feature extractor.
5. Shared decision-making layer: another approach is to use a shared decision-making layer, where the decision-making layer is shared across tasks, and the task-specific layers are connected to the shared decision-making layer.
6. Careful architecture design: The architecture of MTL should be carefully designed to accommodate the different tasks and to make sure that the shared features are useful for all tasks.
7. Overfitting: MTL models can be prone to overfitting if the model is not regularized properly.
8. Avoiding negative transfer: when the tasks are very different or independent, MTL can lead to suboptimal performance compared to training a single-task model. Therefore, it is important to make sure that the shared features are useful for all tasks to avoid negative transfer.

**What is Data Augmentation?**

Data augmentation is the process of artificially generating new data from existing data, primarily to train new machine learning (ML) models. ML models require large and varied datasets for initial training, but sourcing sufficiently diverse real-world datasets can be challenging because of data silos, regulations, and other limitations. Data augmentation artificially increases the dataset by making small changes to the original data. Generative artificial intelligence (AI) solutions are now being used for high-quality and fast data augmentation in various industries.

**Why is data augmentation important?**

Deep learning models rely on large volumes of diverse data to develop accurate predictions in various contexts. Data augmentation supplements the creation of data variations that can help a model improve the accuracy of its predictions. Augmented data is vital in training.

Here are some of the benefits of data augmentation.

**Enhanced model performance**

Data augmentation techniques help enrich datasets by creating many variations of existing data. This provides a larger dataset for training and enables a model to encounter more diverse features. The augmented data helps the model better generalize to unseen data and improve its overall performance in real-world environments.

**Reduced data dependency**

The collection and preparation of large data volumes for training can be costly and time-consuming. Data augmentation techniques increase the effectiveness of smaller datasets, vastly reducing the dependency on large datasets in training environments. You can use smaller datasets to supplement the set with synthetic data points.

**Mitigate overfitting in training data**

Data augmentation helps prevent overfitting when you’re training ML models. Overfitting is the undesirable ML behavior where a model can accurately provide predictions for training data but it struggles with new data. If a model trains only with a narrow dataset, it can become overfit and can give predictions related to only that specific data type. In contrast, data augmentation provides a much larger and more comprehensive dataset for model training. It makes training sets appear unique to deep neural networks, preventing them from learning to work with only specific characteristics.

**Improved data privacy**

If you need to train a deep learning model on sensitive data, you can use augmentation techniques on the existing data to create synthetic data. This augmented data retains the input data's statistical properties and weights while protecting and limiting access to the original.

**What are the use cases of data augmentation?**

Data augmentation offers several applications in various industries, improving the performance of ML models across many sectors.

**Healthcare**

Data augmentation is a useful technology in medical imaging because it helps improve diagnostic models that detect, recognize, and diagnose diseases based on images. The creation of an augmented image provides more training data for models, especially for rare diseases that lack source data variations. The production and use of synthetic patient data advances medical research while respecting all data privacy considerations.

**Finance**

Augmentation helps produce synthetic instances of fraud, enabling models to train to detect fraud more accurately in real-world scenarios. Larger pools of training data help in risk assessment scenarios, enhancing the potential of deep learning models to accurately assess risk and predict future trends.

**Manufacturing**

The manufacturing industry uses ML models to identify visual defects in products. By supplementing real-world data with augmented images, models can improve their image recognition capabilities and locate potential defects. This strategy also reduces the likelihood of shipping a damaged or defective project to factories and production lines.

**Retail**

Retail environments use models to identify products and assign them to categories based on visual factors. Data augmentation can produce synthetic data variations of product images, creating a training set that has more variance in terms of lighting conditions, image backgrounds, and product angles.

**How does data augmentation work?**

Data augmentation transforms, edits, or modifies existing data to create variations. The following is a brief overview of the process.

**Dataset exploration**

The first stage of data augmentation is to analyze an existing dataset and understand its characteristics. Features like the size of input images, the distribution of the data, or the text structure all give further context for augmentation.

You can select different data augmentation techniques based on the underlying data type and the desired results. For example, augmenting a dataset with many images includes adding noise to them, scaling, or cropping them. Alternatively, augmenting a text dataset for natural language processing (NLP replaces synonyms or paraphrases excerpts.

**Augmentation of existing data**

After you’ve selected the data augmentation technique that work best for your desired goal, you begin applying different transformations. Data points or image samples in the dataset transform by using your selected augmentation method, providing a range of new augmented samples.

During the augmentation process, you maintain the same labeling rules for data consistency, ensuring that the synthetic data includes the same labels corresponding to the source data.

Typically, you look through the synthetic images to determine whether the transformation succeeded. This additional human-led step helps maintain higher data quality.

**Integrate data forms**

Next, you combine the new, augmented data with the original data to produce a larger training dataset for the ML model. When you’re training the model, you use this composite dataset of both kinds of data.

It’s important to note that new data points that are created by synthetic data augmentation carry the same bias as the original input data. To prevent biases from transferring into your new data, address any bias in the source data before starting the data augmentation process.

**What are some data augmentation techniques?**

Data augmentation techniques vary across different data types and distinct business contexts.

**Computer vision**

Data augmentation is a central technique in computer vision tasks. It helps create diverse data representations and tackle class imbalances in a training dataset.

The first usage of augmentation in computer vision is through position augmentation. This strategy crops, flips, or rotates an input image to create augmented images. Cropping either resizes the image or crops a small part of the original image to create a new one. Rotation, flip, and resizing transformation all alter the original randomly with a given probability of providing new images.

Another usage of augmentation in computer vision is in color augmentation. This strategy adjusts the elementary factors of a training image, such as its brightness, contrast degree, or saturation. These common image transformations change the hue, dark and light balance, and separation between an image's darkest and lightest areas to create augmented images.

Read about computer vision

**Audio data augmentation**

Audio files, such as speech recordings, are also a common field where you can use data augmentation. Audio transformations typically include injecting random or Gaussian noise into some audio, fast-forwarding parts, changing the speed of parts by a fixed rate, or altering the pitch.

**Text data augmentation**

Text augmentation is a vital data augmentation technique for NLP and other text-related sectors of ML. Transformations of text data include shuffling sentences, changing the positions of words, replacing words with close synonyms, inserting random words, and deleting random words.

**Neural style transfer**

Neural style transfer is an advanced form of data augmentation that deconstructs images into smaller parts. It uses a series of convolutional layers that separate the style and context of an image, producing many images from a single one.

**Adversarial training**

Changes on the pixel level create a challenge for an ML model. Some samples include a layer of imperceptible noise over an image to test the model’s ability to perceive the image underneath. This strategy is a preventative form of data augmentation focusing on potential unauthorized access in the real world.

**What is the role of generative AI in data augmentation?**

Generative AI is essential in data augmentation because it facilitates the production of synthetic data. It helps increase data diversity, streamline the creation of realistic data, and preserve data privacy.

**Generative adversarial networks**

Generative adversarial networks (GAN) are a framework of two central neural networks that work in opposition. The generator produces samples of synthetic data, then the discriminator distinguishes between the real data and the synthetic samples.

Over time, GANs continually improve the generator's output by focusing on deceiving the discriminator. Data that can fool the discriminator counts as high-quality synthetic data, providing data augmentation with highly reliable samples that closely mimic the original data distribution.

**Variational autoencoders**

Variational autoencoders (VAE) are a type of neural network that help to increase the sample size of core data and reduce the need for time-consuming data collection. VAEs have two connected networks: a decoder and an encoder. The encoder takes sample images and translates them into an intermediate representation. The decoder takes the representation and recreates similar images based on its understanding of the initial samples. VAEs are useful because they can create data highly similar to sample data, helping add variety while maintaining the original data distribution.

**How can AWS support your data augmentation requirements?**

Generative AI on Amazon Web Services (AWS) is a set of technologies that organizations of all sizes can use to build and scale generative AI applications with customized data for custom use cases. You can innovate faster with new capabilities, a choice of industry-leading foundation models (FMs), and the most cost-effective infrastructure. The following are two examples of generative AI services on AWS.

Amazon Bedrock is a fully managed service that offers a choice of high-performing FMs from leading AI companies. You can securely integrate and deploy generative AI capabilities for data augmentation without managing infrastructure.

Amazon Rekognition is a fully managed AI service that offers pre-trained and customizable computer vision capabilities to extract information and insights from your images and videos. The development of a custom model to analyze images is a significant undertaking that requires time, expertise, and resources. It often requires thousands or tens of thousands of hand-labeled images to provide the model with enough data to make decisions accurately.

With Amazon Rekognition Custom Labels, various data augmentations are performed for model training, including random cropping of the image, color jittering, and random Gaussian noises. Instead of thousands of images, you need to upload only a small set of training images (typically a few hundred or less) specific to your use case to the easy-to-use console.

**Noise Robustness**

Noise applied to inputs is a data augmentation, for some models addition of noise with extremely small variance at the input is equivalent to imposing a penalty on the norm of the weights.

Noise applied to hidden units; Noise injection can be much more powerful than simply shrinking the parameters. Noise applied to hidden units is so important that Dropout is the main development of this approach.

Adding Noise to Weights, this technique primarily used with Recurrent Neural Networks (RNNs). This can be interpreted as a stochastic implementation of Bayesian inference over the weights. Bayesian learning considers model weights to be uncertain and representable via a probability distribution p(w) that reflects that uncertainty. Adding noise to weights is a practical, stochastic way to reflect this uncertainty.

Noise applied to weights is equivalent to traditional regularization, encouraging stability. This can be seen in a regression setting, Train ŷ (x) to map x to a scalar using least squares between model prediction ŷ (x) and true values y.

J= Ep(x,y)[(ŷ (x)−y)2]

The training set consists of m labeled examples {(x(1), y(1)), . . . , (x(m), y(m))}. We perturb each input with εW∼N(ε;0,ηI) For small η, this is equivalent to a regularization term η Ep(x,y)[∥∇wŷ (x)∥2] It encourages parameters to regions where small perturbations of weights have small influence on output

Injecting Noise at the Output Targets, most datasets have some amount of mistakes in the y labels. It can be harmful to maximize log p(y | x) when y is a mistake. Most datasets have some amount of mistakes in the y labels. It can be harmful to maximize log p (y | x) when y is a mistake. This can be incorporated into the cost function, Ex: Local Smoothing regularizes a model based on a softmax with k output values by replacing the hard 0 and 1 classification targets with targets of ε/(k-1) and 1-ε respectively

**Regularization by Early Stopping**

Regularization is a kind of regression where the learning algorithms are modified, to reduce overfitting. This may incur a higher bias but will lead to lower variance when compared to non-regularized models i.e. increases generalization of the training algorithm.

**Why Regularisation is needed?**

In a general learning algorithm, the dataset is divided into a training set and a test set. After each epoch of the algorithm, the parameters are updated accordingly after understanding the dataset. Finally, this trained model is applied to the test set.

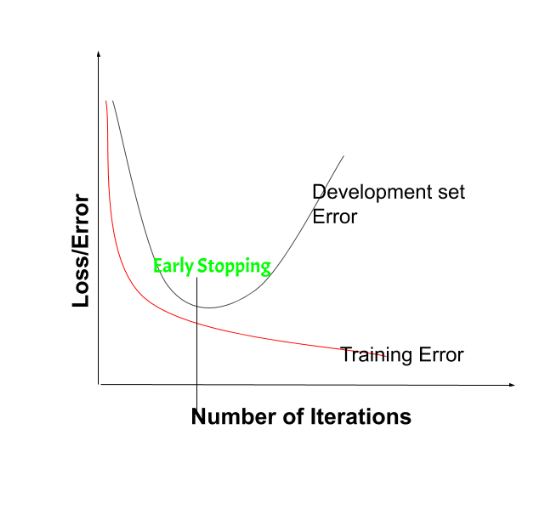
Generally, the training set error will be less compared to the test set error. This is because of overfitting whereby the algorithm memorizes the training data and produces the right results on the training set. So, the model becomes highly exclusive to the training set and fails to produce accurate results for other datasets including the test set.

Regularization techniques are used in such situations to reduce overfitting and increase the model’s performance on any general dataset.

**What is Early Stopping?**

In Regularization by Early Stopping, we stop training the model when the performance on the validation set is getting worse- increasing loss decreasing accuracy, or poorer scores of the scoring metric. By plotting the error on the training dataset and the validation dataset together, both the errors decrease with a number of iterations until the point where the model starts to overfit. After this point, the training error still decreases but the validation error increases.

So, even if training is continued after this point, early stopping essentially returns the set of parameters that were used at this point and so is equivalent to stopping training at that point. So, the final parameters returned will enable the model to have low variance and better generalization. The model at the time the training is stopped will have a better generalization performance than the model with the least training error.



on the validation set is getting worse- increasing loss or decreasing accuracy or poorer scores

Early stopping can be thought of as **implicit regularization**, contrary to regularization via weight decay. This method is also efficient since it requires less amount of training data, which is not always available. Due to this fact, early stopping requires lesser time for training compared to other regularization methods. Repeating the early stopping process many times may result in the model overfitting the validation dataset, just as similar as overfitting occurs in the case of training data.

The number of iterations (i.e. epoch) taken to train the model can be considered a **hyperparameter**. Then the model has to find an optimum value for this hyperparameter (by hyperparameter tuning) for the best performance of the learning model.

***Tip:****The downside of early stopping are as follows:*

*By stopping early, we can’t able to optimize Cost function(J) much for the training set. So, we use a different concept Known as****Orthogonalization****is used.*

**Benefits of Early Stopping:**

* Helps in reducing overfitting
* It improves generalisation
* It requires less amount of training data
* Takes less time compared to other regularisation models
* It is simple to implement

**Limitations of Early Stopping:**

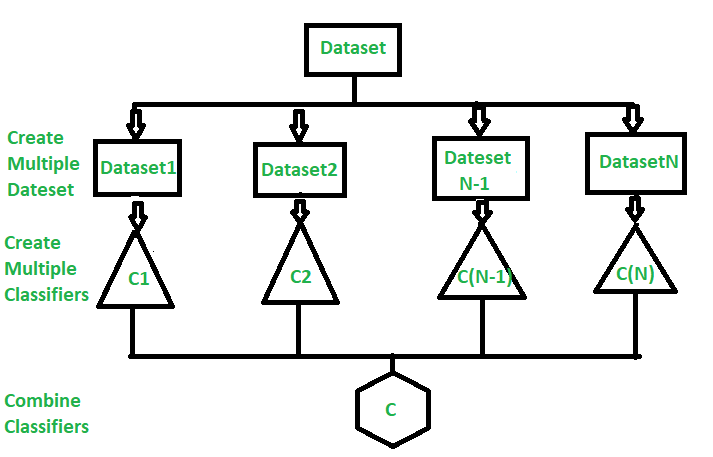
* If the model stops too early, there might be risk of underfitting
* It may not be beneficial for all types of models
* If validation set is not chosen properly, it may not lead to the most optimal stopping

To summarize, early stopping can be best used to prevent overfitting of the model, and saving resources. It would give best results if taken care of few things like – parameter tuning, preventing the model from overfitting, and ensuring that the model learns enough from the data.

**ENSEMBLE METHODS**

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote.

**Advantage :** Improvement in predictive accuracy.  
**Disadvantage :** It is difficult to understand an ensemble of classifiers



**Why do ensembles work?**

Dietterich(2002) showed that ensembles overcome three problems –

**Statistical Problem –**  
The Statistical Problem arises when the hypothesis space is too large for the amount of available data. Hence, there are many hypotheses with the same accuracy on the data and the learning algorithm chooses only one of them! There is a risk that the accuracy of the chosen hypothesis is low on unseen data!

**Computational Problem –**  
The Computational Problem arises when the learning algorithm cannot guarantees finding the best hypothesis.

**Representational Problem –**  
The Representational Problem arises when the hypothesis space does not contain any good approximation of the target class(es).

**Main Challenge for Developing Ensemble Models?**

The main challenge is not to obtain highly accurate base models, but rather to obtain base models which make different kinds of errors. For example, if ensembles are used for classification, high accuracies can be accomplished if different base models misclassify different training examples, even if the base classifier accuracy is low.

* **Methods for Independently Constructing Ensembles –**
* Majority Vote
* Bagging and Random Forest
* Randomness Injection
* Feature-Selection Ensembles
* Error-Correcting Output Coding
* **Methods for Coordinated Construction of Ensembles –**
* Boosting
* Stacking
* **Reliable Classification:** Meta-Classifier Approach
* **Co-Training and Self-Training**

**Types of Ensemble Classifier –**

**Bagging:**  
Bagging (Bootstrap Aggregation) is used to reduce the variance of a decision tree. Suppose a set D of d tuples, at each iteration *i*, a training set Di of d tuples is sampled with replacement from D (i.e., bootstrap). Then a classifier model Mi is learned for each training set D < i. Each classifier Mi returns its class prediction. The bagged classifier M\* counts the votes and assigns the class with the most votes to X (unknown sample).

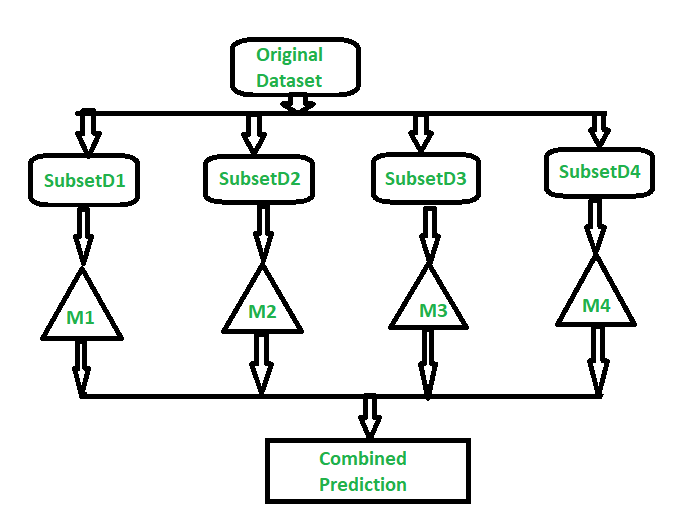
**Implementation steps of Bagging –**

Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.

A base model is created on each of these subsets.

Each model is learned in parallel from each training set and independent of each other.

The final predictions are determined by combining the predictions from all the models.



**Random Forest:**  
Random Forest is an extension over bagging. Each classifier in the ensemble is a decision tree classifier and is generated using a random selection of attributes at each node to determine the split. During classification, each tree votes and the most popular class is returned.

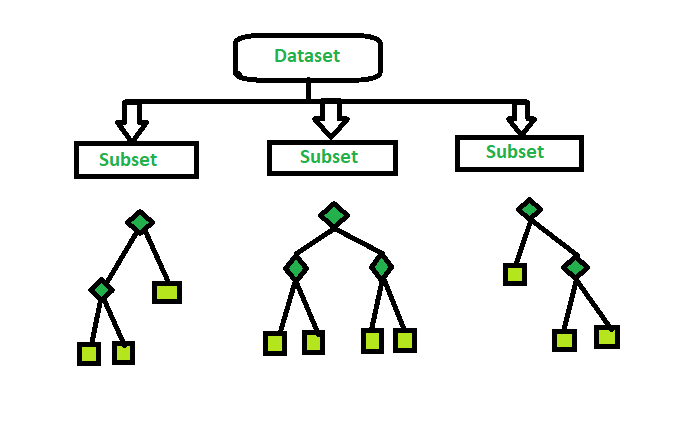
**Implementation steps of Random Forest –**

Multiple subsets are created from the original data set, selecting observations with replacement.

A subset of features is selected randomly and whichever feature gives the best split is used to split the node iteratively.

The tree is grown to the largest.

Repeat the above steps and prediction is given based on the aggregation of predictions from n number of trees.



* **Bagging vs Boosting**

**Bagging** and **Boosting**are two types of **Ensemble Learning**. These two decrease the variance of a single estimate as they combine several estimates from different models. So the result may be a model with higher stability. Let’s understand these two terms in a glimpse.

**Bagging**: It is a homogeneous weak learners’ model that learns from each other independently in parallel and combines them for determining the model average.

**Boosting**: It is also a homogeneous weak learners’ model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

Let’s look at both of them in detail and understand the Difference between Bagging and Boosting.

**Bagging**

**B**ootstrap **A**ggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods. Bagging is a special case of the model averaging approach.

**Description of the Technique**

Suppose a set D of d tuples, at each iteration i, a training set Di of d tuples is selected via row sampling with a replacement method (i.e., there can be repetitive elements from different d tuples) from D (i.e., bootstrap). Then a classifier model Mi is learned for each training set D < i. Each classifier Mi returns its class prediction. The bagged classifier M\* counts the votes and assigns the class with the most votes to X (unknown sample).

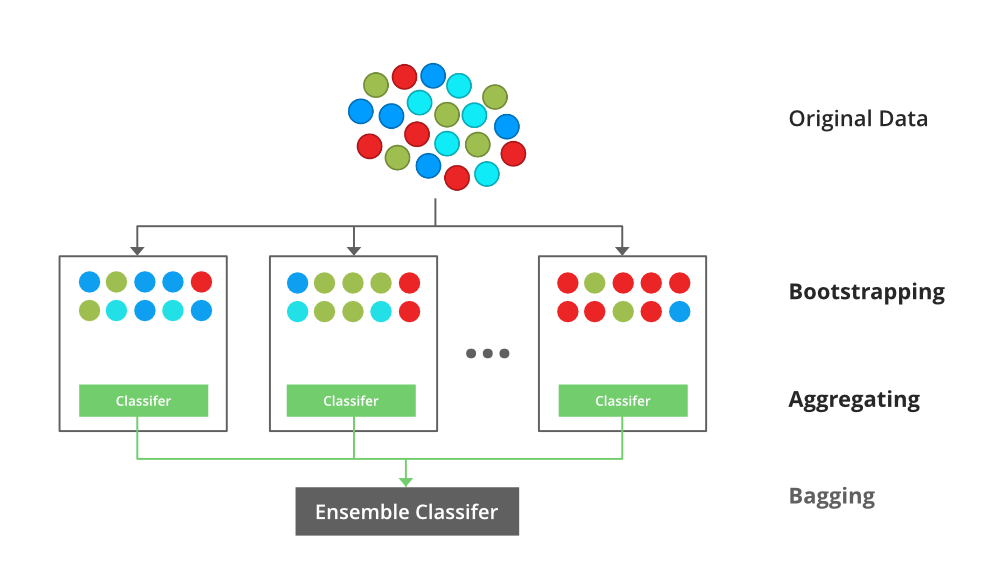
**Implementation Steps of Bagging**

**Step 1:** Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.

**Step 2:** A base model is created on each of these subsets.

**Step 3:**Each model is learned in parallel with each training set and independent of each other.

**Step 4:**The final predictions are determined by combining the predictions from all the models.



An illustration for the concept of bootstrap aggregating (Bagging)

**Example of Bagging**

The random forest model uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest.

**Boosting**

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models is added.

**Boosting Algorithms**

There are several boosting algorithms. The original ones, proposed by **Robert Schapire** and **Yoav Freund** were not adaptive and could not take full advantage of the weak learners. Schapire and Freund then developed adaboost, an adaptive boosting algorithm that won the prestigious Gödel Prize. AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. AdaBoost is short for Adaptive Boosting and is a very popular boosting technique that combines multiple “weak classifiers” into a single “strong classifier”.

**Algorithm:**

Initialise the dataset and assign equal weight to each of the data point.

Provide this as input to the model and identify the wrongly classified data points.

Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.

if (got required results)  
  Goto step 5  
else  
  Goto step 2

End



An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset.

**Similarities between Bagging and Boosting**

Bagging and Boosting, both being the commonly used methods, have a universal similarity of being classified as ensemble methods. Here we will explain the similarities between them.

1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
4. Both are good at reducing variance and provide higher stability.

**Between Bagging and Boosting**

| **S.NO** | **Bagging** | **Boosting** |
| --- | --- | --- |
| 1. | The simplest way of combining predictions that belongs to the same type. | A way of combining predictions that  belong to the different types. |
| 2. | Aim to decrease variance, not bias. | Aim to decrease bias, not variance. |
| 3. | Each model receives equal weight. | Models are weighted according to their performance. |
| 4. | Each model is built independently. | New models are influenced  by the performance of previously built models. |
| 5. | Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset. | Every new subset contains the elements that were misclassified by previous models. |
| 6. | Bagging tries to solve the over-fitting problem. | Boosting tries to reduce bias. |
| 7. | If the classifier is unstable (high variance), then apply bagging. | If the classifier is stable and simple (high bias) the apply boosting. |
| 8. | In this base classifier are trained parallel. | In this base classifier are trained sequentially. |
| 9 | Example: The Random Forest model uses Bagging. | Example: The AdaBoost uses Boosting techniques |

**Dropouts**

**INTRODUCTION**

So before diving deep into its world, let’s address the first question. What is the problem that we are trying to solve?

The deep neural networks have different architectures, sometimes shallow, sometimes very deep trying to generalise on the given dataset. But, in this pursuit of trying too hard to learn different features from the dataset, they sometimes learn the **statistical noise** in the dataset. This definitely improves the model performance on the training dataset but fails massively on new data points (test dataset). This is the problem of **overfitting.**To tackle this problem we have various regularisation techniques that penalise the weights of the network but this wasn’t enough.

The best way to reduce overfitting or the best way to regularise a fixed-size model is to get the average predictions from all possible settings of the parameters and aggregate the final output. But this becomes too computationally expensive and isn’t feasible for a real-time inference/prediction.

The other way is inspired by the ensemble techniques (such as AdaBoost, XGBoost, and Random Forest) where we use multiple neural networks of different architectures. But this requires multiple models to be trained and stored, which over time becomes a huge challenge as the networks grow deeper.

So, we have a great solution known as Dropout Layers.

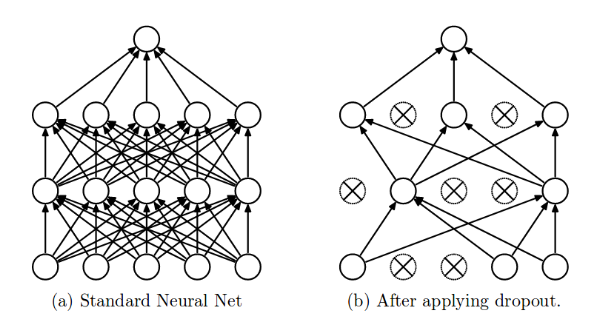


Figure 1: Dropout applied to a Standard Neural Network (Image by Nitish)

**What is a Dropout?**

The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network (as seen in Figure 1). All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p.

Let’s try to understand with a given input x: {1, 2, 3, 4, 5} to the fully connected layer. We have a dropout layer with probability p = 0.2 (or keep probability = 0.8). During the forward propagation (training) from the input x, 20% of the nodes would be dropped, i.e. the x could become {1, 0, 3, 4, 5} or {1, 2, 0, 4, 5} and so on. Similarly, it applied to the hidden layers.

For instance, if the hidden layers have 1000 neurons (nodes) and a dropout is applied with drop probability = 0.5, then 500 neurons would be randomly dropped in every iteration (batch).

Generally, for the input layers, the keep probability, i.e. 1- drop probability, is closer to 1, 0.8 being the best as suggested by the authors. For the hidden layers, the greater the drop probability more sparse the model, where 0.5 is the most optimised keep probability, that states dropping 50% of the nodes.

**So how does dropout solve the problem of overfitting?**

**How does it solve the Overfitting problem?**

In the overfitting problem, the model learns the statistical noise. To be precise, the main motive of training is to decrease the loss function, given all the units (neurons). So in overfitting, a unit may change in a way that fixes up the mistakes of the other units. This leads to complex co-adaptations, which in turn leads to the overfitting problem because this complex co-adaptation fails to generalise on the unseen dataset.

Now, if we use dropout, it prevents these units to fix up the mistake of other units, thus preventing co-adaptation, as in every iteration the presence of a unit is highly unreliable. So by randomly dropping a few units (nodes), it forces the layers to take more or less responsibility for the input by taking a probabilistic approach.

This ensures that the model is getting generalised and hence reducing the overfitting problem.

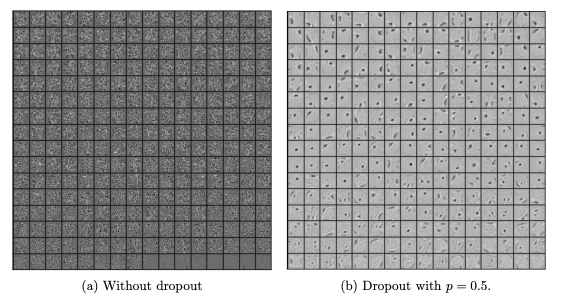


Figure 2: (a) Hidden layer features without dropout; (b) Hidden layer features with dropout (Image by Nitish)

From figure 2, we can easily make out that the hidden layer with dropout is learning more of the generalised features than the co-adaptations in the layer without dropout. It is quite apparent, that dropout breaks such inter-unit relations and focuses more on generalisation.

Dropout Implementation

Enough of the talking! Let’s head to the mathematical explanation of the dropout.

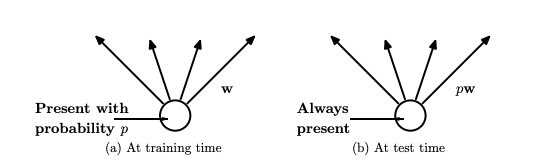


Figure 3: (a) A unit (neuron) during training is present with a probability p and is connected to the next layer with weights ‘**w**’ ; (b) A unit during inference/prediction is always present and is connected to the next layer with weights, ‘**pw**’ (Image by Nitish)

In the original implementation of the dropout layer, during training, a unit (node/neuron) in a layer is selected with a keep probability (1-drop probability). This creates a thinner architecture in the given training batch, and every time this architecture is different.

In the standard neural network, during the forward propagation we have the following equations:

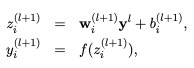


Figure 4: Forward propagation of a standard neural network (Image by Nitish)

where:  
z: denote the vector of output from layer (l + 1) before activation  
y: denote the vector of outputs from layer l  
w: weight of the layer l  
b: bias of the layer l

Further, with the activation function, z is transformed into the output for layer (l+1).

Now, if we have a dropout, the forward propagation equations change in the following way:

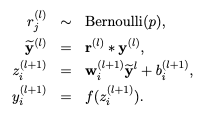


Figure 5: Forward propagation of a layer with dropout (Image by Nitish)

So, before we calculate **z,**the input to the layer is sampled and multiplied element-wise with the independent Bernoulli variables. **r**denotes the Bernoulli random variables each of which has a probability p of being 1. Basically, **r**acts as a mask to the input variable, which ensures only a few units are kept according to the keep probability of a dropout. This ensures that we have thinned outputs “**y(bar)”**, which is given as an input to the layer during feed-forward propagation.

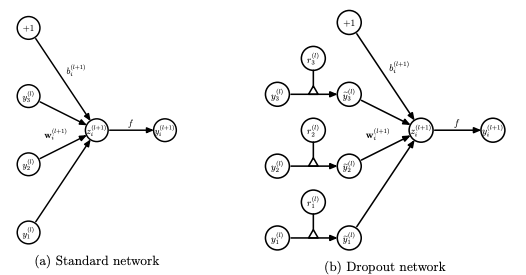


Figure 6: Comparison of the dropout network with the standard network for a given layer during forward propagation (Image by Nitish)

**Dropout during Inference**

Now, we know the dropout works mathematically but what happens during the inference/prediction? Do we use the network with dropout or do we remove the dropout during inference?

This is one of the most important concepts of dropout which very few data scientists are aware of.

According to the original implementation (Figure 3b) during the inference, we do not use a dropout layer. This means that all the units are considered during the prediction step. But, because of taking all the units/neurons from a layer, the final weights will be larger than expected and to deal with this problem, weights are first scaled by the chosen dropout rate. With this, the network would be able to make accurate predictions.

To be more precise, if a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p during the prediction stage.

**How Dropout was conceived?**

According to Geoffrey Hinton, one of the authors of “Dropout: A Simple Way to Prevent Neural Networks from Overfitting” there were a set of events that inspired the fundamental dropout.

The analogy with Google Brain is that it should be big because it learns a large ensemble of models. In neural networks, it is not a very efficient use of hardware since the same features would need to be invented separately by different models. This is when the idea of using the same subset of neurons was discovered.

Bank Teller: In those days, the tellers keep changing regularly and it must be because it would require cooperation between the employees to successfully defraud the bank. This implanted the idea of randomly selecting different neurons such that with every iteration there is a different set of neurons used. This would ensure that neurons are unable to learn the co-adaptations and prevent overfitting, similar to preventing the conspiracies in the bank.

Sexual Reproduction: It involves talking half of the genes of one parent and half of the other, adding a very small amount of random mutation, to produce an offspring. This creates a mixed ability of the genes and makes them more robust. This can be linked to a dropout which is used to break co-adaptations (adds randomness just like a gene mutation).

**Isn’t the entire journey fascinating?**

The main motive to introduce the idea of how the dropout was conceived is to motivate the readers and explore the world around them and relate it to the working principles of several other neural networks. It would definitely give birth to many such innovations.

**Tensorflow Implementation**

If we follow the original implementation, we need to multiply the weights with the dropout probability during the prediction stage. Just to remove any processing during this stage, we have an implementation known as “**inverse dropout”.**

The intention of multiplying weights with dropout probability is to ensure that the final weights are of the same scale, thus the predictions are correct. In inverse dropout, this step is performed during the training itself. At the training time, all the weights that remain after the dropout operation is multiplied by the inverse of keep probability, i.e. w \* (1/p).

To gain mathematical proof of why both operations are similar on the layer weights, I recommend going through a blog by Lei Mao.

**CONCLUSION**

Finally!! We have covered the in-depth analysis of the dropout layers that we use with almost all the neural networks.

Dropouts can be used with most types of neural networks. It is a great tool to reduce overfitting in a model. It is far better than the available regularisation methods and can also be combined with max-norm normalisation which provides a significant boost over just using dropout.

In the upcoming blogs, we would learn more about such basic layers which are used in almost all networks. Batch normalisation, layer normalisation, and attention layers to name a few.