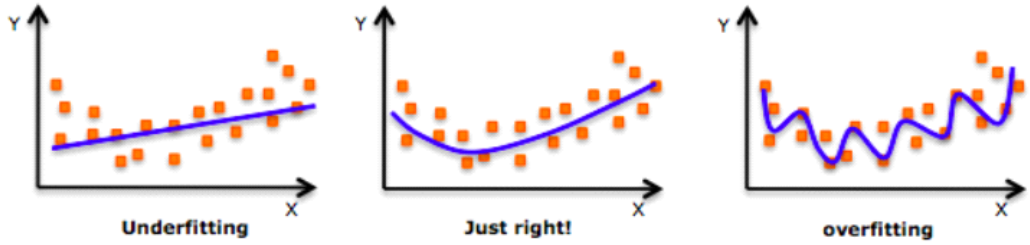
Regularization

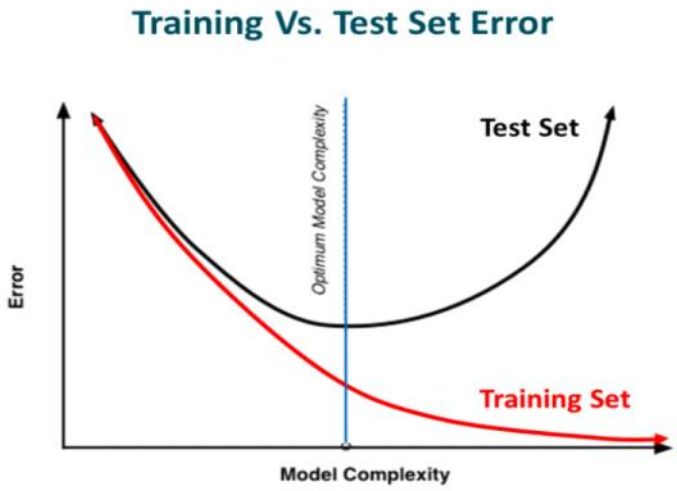
29 July 2024 08:35 PM

What is Regularization?

- Before we deep dive into the topic, take a look at this image:



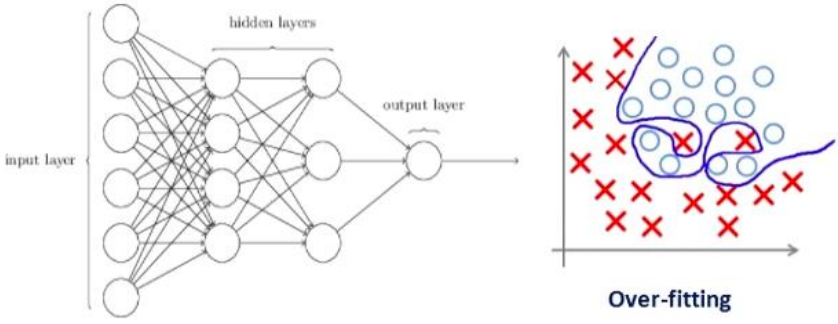
- Have you seen this image before? As we move towards the right in this image, our model tries to learn too well the details an d the noise from the training data, which ultimately results in poor performance on the unseen data.
- In other words, while going towards the right, the complexity of the model increases such that the training error reduces but the testing error doesn't.
- This is shown in the image below.



- If you've built a neural network before, you know how complex they are. This makes them more prone to overfitting.
- **Regularization** is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in tur n improves the model's performance on the unseen data as well.

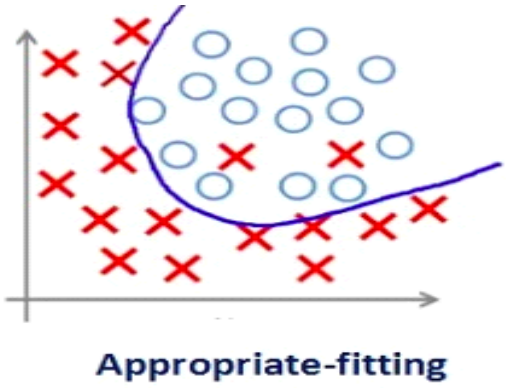
How does Regularization help reduce Overfitting?

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



- If you have studied the concept of regularization in machine learning, 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 re sult 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 coef ficient in order to obtain a well-fitted model as shown in the image below.



In [1]:

Different Regularization Techniques in Deep Learning

- Now that we have an understanding of how regularization helps in reducing overfitting, we ’ll learn a few different techniques in order to apply regularization in deep learning.
- **L2 & L1 Regularization:**
- L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.
- Cost function = Loss (say, binary cross entropy) + Regularization term
- Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural netw ork with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.
- However, this regularization term differs in L1 and L2.
- In L2, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||^2$$

- Here, lambda is the regularization parameter. It is the hyperparameter whose value is optimized for better results. L2 regularization is a lso known as weight decay as it forces the weights to decay towards zero (but not exactly zero).
- In L1, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||$$

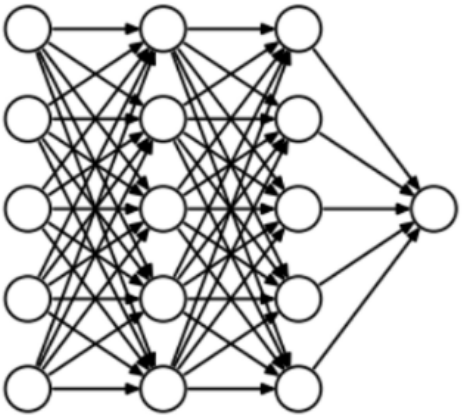
- In this, we penalize the absolute value of the weights. Unlike L2, the weights may be reduced to zero here. Hence, it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it.
- In keras, we can directly apply regularization to any layer using the regularizers. Below I have applied regularizer on dense layer having 500 neurons and relu activation function.

```
model.add(Dense(500,kernel_regularizer=regularizers.l2(0.01),activation="relu"))
model.add(Dense(2,activation="softmax"))#2 represent output layer neurons
```

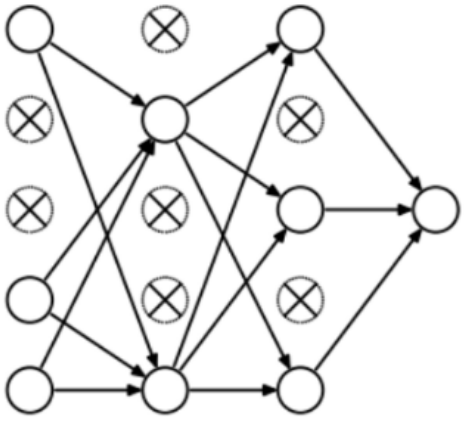
- **Note:** Here the value 0.01 is the value of regularization parameter, i.e., lambda, which we need to optimize further. We can optimiz e it using the grid-search method.
- Similarly, we can also apply L1 regularization.

Dropout:

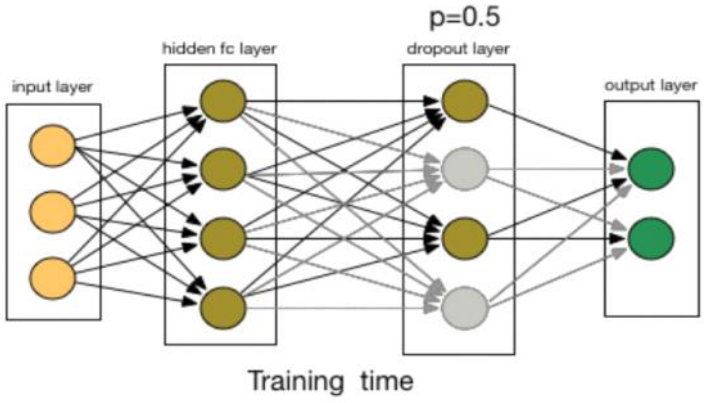
- This is the one of the most interesting types of regularization techniques. It also produces very good results and is consequ ently the most frequently used regularization technique in the field of deep learning.
- To understand dropout, let’s say our neural network structure is akin to the one shown below:



- So what does dropout do? At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown below.



- So each iteration has a different set of nodes and this results in a different set of outputs. It can also be thought of as an ensemble technique in machine learning.
- Ensemble models usually perform better than a single model as they capture more randomness. Similarly, dropout also performs better than a normal neural network model.
- This probability of choosing how many nodes should be dropped is the hyperparameter of the dropout function. As seen in the i mage above, dropout can be applied to both the hidden layers as well as the input layers.



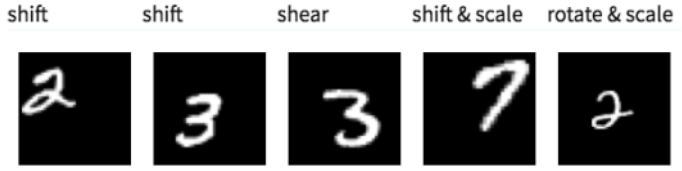
- Due to these reasons, dropout is usually preferred when we have a large neural network structure in order to introduce more randomness.
- In keras, we can implement dropout using the keras layer. Below is the Dropout Implementation. I have introduced dropout of 0.2 as the probability of dropping in my neural network architecture after last hidden layer having 64 kernels and after first dense layer having 500 n eurons.

```
#creating sequential model
model=Sequential()
model.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape=(50,50,3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32,kernel_size=2,padding="same",activation="relu"))
```

```
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64,kernel_size=2,padding="same",activation="relu"))
model.add(MaxPooling2D(pool_size=2))
# 1st dropout
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(500,activation="relu"))
# 2nd dropout
model.add(Dropout(0.2))
model.add(Dense(2,activation="softmax"))#2 represent output layer neurons
```

Data Augmentation:

- The simplest way to reduce overfitting is to increase the size of the training data. In machine learning, we were not able to increase the size of training data as the labeled data was too costly.
- But, now let's consider we are dealing with images. In this case, there are a few ways of increasing the size of the training data – rotating the image, flipping, scaling, shifting, etc. In the below image, some transformation has been done on the handwritten digits dataset.



- This technique is known as data augmentation. This usually provides a big leap in improving the accuracy of the model. It can be considered as a mandatory trick in order to improve our predictions.

Early stopping:

- Early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model. This is known as early stopping.

- In the above image, we will stop training at the dotted line since after that our model will start overfitting on the trainin g data.
- In keras, we can apply early stopping using the callbacks function. Below is the implementation code for it. I have applied early stopping so that it will stop immediately if validation error will not decreased after 3 epochs.

```
from keras.callbacks import EarlyStopping
earlystop= EarlyStopping(monitor='val_acc', patience=3)
epochs = 20 #
batch_size = 256
```

- Here, monitor denotes the quantity that needs to be monitored and 'val_err' denotes the validation error.
- Patience denotes the number of epochs with no further improvement after which the training will be stopped. For better understanding, let's take a look at the above image again. After the dotted line, each epoch will result in a higher value of validation error. Therefore, 5 e pochs after the dotted line (since our patience is equal to 3), our model will stop because no further improvement is seen.
- **Note:** It may be possible that after 3 epochs (this is the value defined for patience in general), the model starts improving again and the validation error starts decreasing as well. Therefore, we need to take extra care while tuning this hyperparameter.