Deep Learning models have so much flexibility and capacity that **overfitting can be a serious problem**, if the training dataset is not big enough.

**High Bias or Underfitting** means that the model is not able to capture the trend or pattern in data. It is usually caused when the hypothesis function is too simple or has very less features.

Fix:-

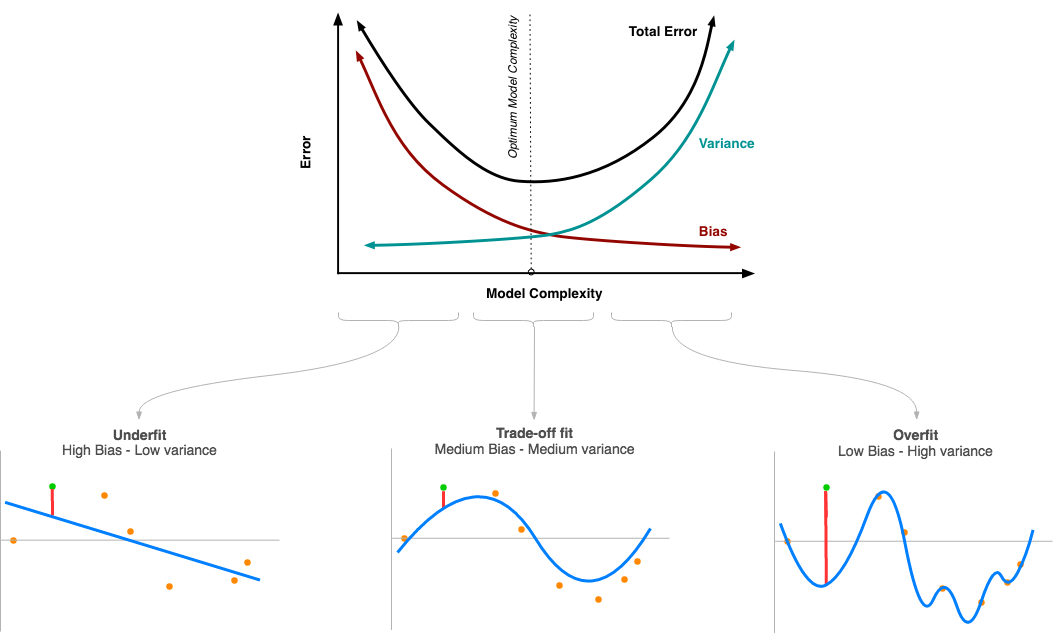
1. add more meaningful factors to the data

2. increasing the degree of the polynomial can increase the complexity thereby reducing Bias

**High variance or Overfitting** means that the model fits the available data but does not generalise well to predict on new data

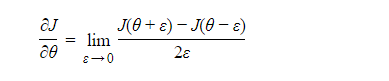
Fix:-

1. reducing the number of features in the model.
2. Increasing the size of the training set
3. Decreasing the degree of the polynomial
4. Regularization is a popular method



<https://medium.datadriveninvestor.com/bias-variance-trade-off-fb5fa4c8ab56>

**Gradient Checking:**- Implement gradient checking to verify the accuracy of your backprop implementation.



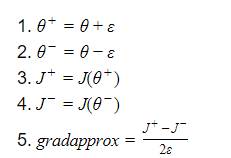
Theta denotes the parameters of the model

### Gradient\_check:-

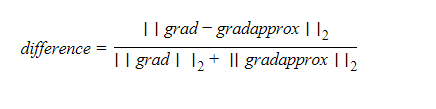
To show that the backward\_propagation() function is correctly computing the gradient ∂*J*∂*θ*, let's implement gradient checking.

**Instructions**:

* First compute "gradapprox" using the formula above (1) and a small value of *ε*. Here are the Steps to follow:



* Then compute the gradient using backward propagation, and store the result in a variable "grad"
* Finally, compute the relative difference between "gradapprox" and the "grad" using the following formula:



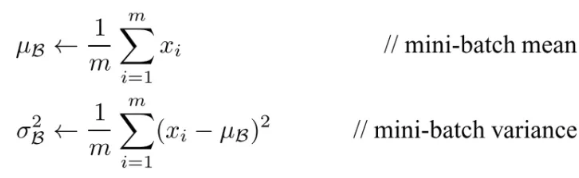
You will need 3 Steps to compute this formula:

* + 1'. compute the numerator using np.linalg.norm(...)
  + 2'. compute the denominator. You will need to call np.linalg.norm(...) twice.
  + 3'. divide them.
* If **this difference is small (say less than 10−7), you can be quite confident** that you have computed your gradient correctly. Otherwise, there may be a mistake in the gradient computation.

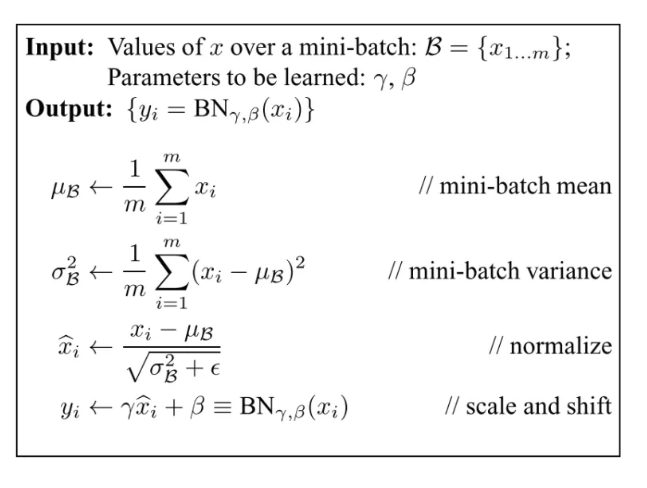
# **Normalization :** decrease your model’s training time by a huge factor.

* Batch normalization
* Weight Normalization
* Layer Normalization
* Instance(or Contrast) Normalization
* Group Normalization
* Batch-Instance Normalization
* Switchable Normalization

**1. Batch normalization** is a method that normalizes activations in a network.It computes the mean and variance of that feature in the mini-batch. It then subtracts the mean and divides the feature by its mini-batch standard deviation.



But wait, what if increasing the magnitude of the weights made the network perform better?



## **Problems associated with Batch Normalization :**

**Variable Batch Size →**If batch size is of 1, then variance would be 0 which doesn’t allow batch norm to work.

Furthermore, if we have small mini-batch size then it becomes too noisy and training might affect.**1.**

**Recurrent Neural Network** → In an RNN, the recurrent activations of each time-step will have a different story to tell(i.e. statistics). This means that we have to fit a separate batch norm layer for each time-step

https://towardsdatascience.com/different-normalization-layers-in-deep-learning-1a7214ff71d6