# 

[**7.3: Regularization Methods**](#_qq326ov5pcdt) **1**

[7.3.1: Simple vs complex models](#_wyujvhdpz4j6) 1

[7.3.2: Analysing the behaviour of simple and complex models](#_20cvps37pf74) 4

[7.3.3: Bias and Variance](#_abchongavxmz) 5

[7.3.4: Test error due to high bias and high variance](#_h8bgmn88zgq4) 7

[7.3.5: Overfitting in deep neural networks](#_4t9y9d64szc3) 9

[7.3.6: A detour into hyperparameter tuning](#_uacflssrzip1) 10

[7.3.7: L2 regularization](#_lghpft1jdumx) 11

[7.3.8: Dataset Augmentation and Early Stopping](#_mae7s5xd8q4n) 13

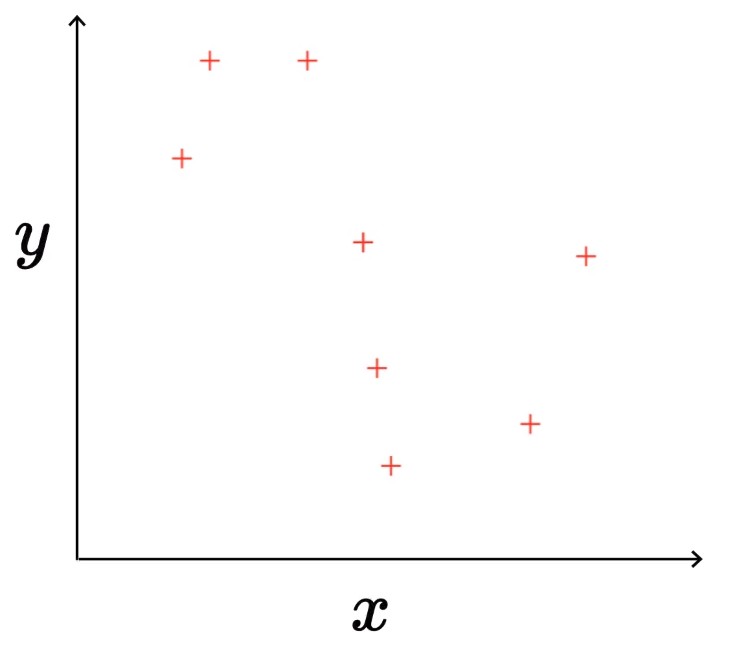
[7.3.9: Summary](#_let8gtpuj18) 14

# 

# 7.3: Regularization Methods

## 7.3.1: Simple vs complex models

The timeline of where we are

1. In this section, we will look at how better Regularization methods have accelerated the growth of DL over the last decade 
2. Why do we need **Regularization**?
   1. To answer this question, we must look at a concept known as **Bias Variance trade-off**. The Bias that we’re speaking about here is different from the bias parameter b that we have seen so far in Neural Networksμ
   2. Consider the following toy data visualisation
   3. In the above figure, the true relation is **y = f(x), where f(x) = sin(x)**, however, in practice, that is not known to us. So we try to approximate models.
   4. **Simple**(degree 1):
      1. We assume that the relationship between y and x is a straight line of the form mx + c
      2. This looks like a very naive assumption.
      3. It is represented by the Red line in the figure
      4. The best fitting Red line is plotted while trying to minimize the error/loss between the predicted points and the actual points
      5. This is a pretty bad model, where even the minimised loss is still far too high
   5. **Complex**(degree 25):
      1. This is a degree 25 polynomial, with 26 parameters (including w0)
      2. It is represented by the Blue curve in the figure
      3. The Blue curve is plotted the same way, by minimising the error/loss between predicted and actual values
      4. Here, there is zero error/loss, it is a perfect fit.
3. Now, how does this relate to Bias and Variance and how does it in turn lead to regularization.

## 7.3.2: Analysing the behaviour of simple and complex models

What happens if you train using different sets of training data

1. Consider a dataset of say 1000 points. When we train our models (Simple and Complex), we shuffle the dataset and then take different subsets of data (around 100 points each).
2. Let us observe how the two models behave when dealing with varying training subsets from the same dataset.
3. **Simple**(degree 1):
   1. Let us look at how the model behaves for 3 different subsets of 100 points each
   2. What we can infer from this is that the model is not very sensitive to the training data, i.e. the model doesn’t respond too much to the points given, thus all the predicted lines are very similar to each other.
4. **Complex**(degree 25):
   1. Let us look at how the model behaves for 3 different subsets
   2. Here, we can see that each of the functions are quite different from each other
   3. What we can infer from this is that the model is highly sensitive to the training data provided, i.e. The models adapt highly to the points given, thus producing different plots each time.

## 7.3.3: Bias and Variance

Let’s define some terms based on our observations.

1. Here is the same experiment as conducted above, except for 25 subsets instead of 3.

|  |  |
| --- | --- |
| **Simple Model** | **Complex Model** |
|  |  |
| Let us define the term **Bias**:  Bias  Here E stands for expected value of the predictions (The average of predictions)  Bias is the difference between the expectation of the predicted values and the true value | |
|  | |
| In the **simple model**, the Expected value is very different from the true value, leading to a **high bias**. | In the **complex model**, the Expected value is very similar to the true value, leading to a **low bias**. |
| Let’s define another term, **Variance**:  Variance  As before, E stands for expected value, which is nothing but the average value of the points  First, we calculate the square error between the predicted points and the prediction’s average  Then we take the average/expected value of the square error term | |
|  |  |
| In the **simple model**, the average line is very similar to the other lines. The lines all predict very similar values.  Thus, the square error between the lines and the average line will be small, thereby its expected value will also be small.  This corresponds to a **low variance.** | In the **complex model**, the average curve is quite different from the other curves. The curves predict noticeably different values.  Thus, the square error between the curves and the average lincurvee will be large, thereby its expected value will also be large.  This corresponds to a **high variance.** |

1. The following observations can be made
   1. **Simple Model**: high bias, low variance
   2. **Complex Model**: low bias, high variance
   3. **Ideal Model**: low bias, low variance.

## 7.3.4: Test error due to high bias and high variance

What is the effect of high bias and high variance on the test error

1. So far, we have been analysing the performance of the models on training data, and determining if they were high/low bias/variance
   1. The Simple Model failed miserably on the training data, with a very high error/loss value
   2. The Complex Model however performed extremely well. Though it did deviate the sine-function (true curve), it was still able to fit all the training points, scoring a very low error/loss value
2. Let’s look at how it performs on the test dataset
3. Consider the simple model
   1. Let’s look at a visualisation of the test set predictions
   2. Here, the high bias model does poorly on the test set. This is understandable as the model performed poorly on the test set, so it was never very likely to perform well on the test set
4. Consider the complex model
   1. Let’s look at a visualisation of the test set predictions
   2. Here, the high variance model also shows a high test error, unlike its test set performance. This is because the model over-familiarised itself with the training set, to the point that it was unable to successfully predict new points from the test set.
5. Let us look at how training and test error vary with model complexity
6. From the above figure, we can make the following observations
   1. For simpler/high-bias models, the training and test error are both very high. This is because the model has not adjusted in accordance with the inputs given. It can be said that the model is under-fitting.
   2. For complex/high-variance models, the training error is low but the test error is high. This is because the model has adjusted too much to the training inputs given, thereby not being able to predict any new points well. It can be said that the model is overfitting.
   3. The sweet-spot of model-complexity is the perfect trade-off between bias and variance. It is characterised by low training and test error.

## 7.3.5: Overfitting in deep neural networks

Why do we care about bias variance trade-off in the context of Deep Learning

1. Consider the same image from the previous section
2. Deep Neural Networks are highly complex models (many parameters and many non-linearities)
3. Easy to overfit (drive training error to 0)
4. The aim is to maintain the model complexity near the sweet-spot and not have it get too complex.
5. How do we deal with this in practice in Deep Neural Networks? Let’s look at some of the recommended practices
6. Divide data into train, test and validation/development splits
   1. Good rations would be (60:20:20) or (70:20:10) in the order train:validation:test
   2. Never handle the test data except during the final evaluation. All other evaluation must be done with the training set first then the validation set.
   3. Training data is used to minimise the loss/error
   4. Validation data is used to check if the model has become too complex or not.
   5. We must aim to get a good score during evaluation of the validation set
7. Start with some network configuration (say, 2 hidden layers, 50-100 neurons each)
8. Make sure that you are using the:
   1. Right activation function (tanh(RNN), ReLU(CNN), leaky ReLU(CNN))
   2. Right initialisation method (Xavier, He)
   3. Right optimization method (say Adam)
9. Monitoring training and validation error similar to the figure in point number 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Training Error** | **Validation Error** | **Cause** | **Solution** |
| High | High | High Bias | * Increase model complexity * Train for more epochs |
| Low | High | High Variance | * Add more training data (**dataset augmentation**) * Use **regularization** * Use **early stopping** (train less) |
| Low | Low | Perfect trade-off | * You are done! |

## 7.3.6: A detour into hyperparameter tuning

Is the concept of train/validation error also related to hyperparameter tuning?

1. The following image shows us all the variables under our control when configuring a DNN
2. To determine the ideal combination of variables when configuring a DNN, it is recommended to analyse the curves shown in the figure above
3. We need to minimise the difference between train and validation error based on monitoring the curves plotted above.
4. Parameters are variables you learn from the data, i.e. weights, biases etc
5. Hyper-Parameters are variables that you figure out during experiments on the model, by analysing the error and other evaluators.

## 7.3.7: L2 regularization

What is the intuition behind L-2 regularization?

1. Consider the error curves for training and test set
2. In the case of Square error loss:
   1. Where
   2. Our aim has been to minimise the loss function
3. Now, imagine if we include a new term in the minimization condition
   1. Here, in addition to minimising the training loss, we are also minimising some other quantity that is dependent on our parameters
   2. In the case of L2 Regularisation, (sq.root of the sum of the squares of the weight)
   3. Here, we should aim to minimize both , it wouldn’t make sense for either of them to be high values.
4. What if we set all weights to 0? In this case, the model would not have learned much, therefore would be high.
5. What if we try to minimise to 0? In this case, it is possible that some of the weights would take on large values, thereby driving the value of high.
6. To counter the previous point’s shortcoming, we need to minimize but shouldn’t allow the weights to grow too large
7. Thus, as shown in the figure, in L2 Regularisation, we do not allow the training loss to be brought to be zero, instead we maintain it at slightly above zero, so that doesn’t become too high
8. This works in the Gradient Descent Algorithm as well
9. The algorithm
   1. **Initialise:** w111, w112, … w313, b1, b2, b3 randomly
   2. **Iterate over data**
      1. Compute ŷ
      2. Compute L(w,b) Cross-entropy loss function
      3. w111 = w111 - η𝚫w111
      4. w112 = w112 - η𝚫w112

…

* + 1. w313 = w111 - η𝚫w313
  1. **Till satisfied**

1. The derivative of the loss function w.r.t any weight is
2. In the case of L2 Regularisation, that value would be
3. Here, the derivative of the regularisation term will cancel out all other weights except the concerned weight and we will compute its derivative. I.e.
4. So the new derivative term will be
5. This process is automatically done in PyTorch.

## 7.3.8: Dataset Augmentation and Early Stopping

1. What is the intuition behind dataset augmentation?
   1. Let’s look at the train-validation error curves as drawn in the previous explanations
   2. If our original dataset size is small, then it becomes easy to drive the training error to zero (too many parameters for very little data). This is because the parameters learn from the data too well to the point of overfitting. Here we will see a low train error and a high validation error
   3. Augmenting with more data will make it harder to drive the training error to zero
   4. Data Augmentation could be used to obtain multiple data points from a single input, by performing operations such as blurring, cropping, translating (move horizontally or vertically) etc. The benefit of this is that no extra effort needs to be made in labelling the data, as all the augmented images have have the same label as the original image
   5. By augmenting more data, we might also end up seeing data which is similar to validation/test data (hence, effectively reduce the validation/test data)
2. What is early stopping?
   1. Look at the image of the error curves to see how early stopping works
   2. First we keep training our model for a large number of epochs, and keep monitoring the loss.
   3. With a patience parameter p, say p = 5 epochs, monitor the validation error after a large number of epochs k.
   4. If the training error continues to decrease but the validation error stays constant in the patience period of 5 epochs, then we can avoid any more steps and revert back to k-p epochs.
   5. This can be compared to losing patience while waiting for the loss to decrease.
   6. Thus, we return the weights corresponding to the no. of epochs with lowest error.

## 7.3.9: Summary

Let’s look at where we are now

1. We have covered a lot of interesting topics in regularization
2. We haven’t covered the regularization methods such as dropout & batch-normalisation, but they will be covered as we move forward
3. The next few sections will be more hands-on, and we will get to start working with PyTorch and CNNs
4. The next contest will cover all of the following concepts