**Note: This is the summary note from Udacity Introduction to Deep Learning with PyTorch**

## Training Optimization

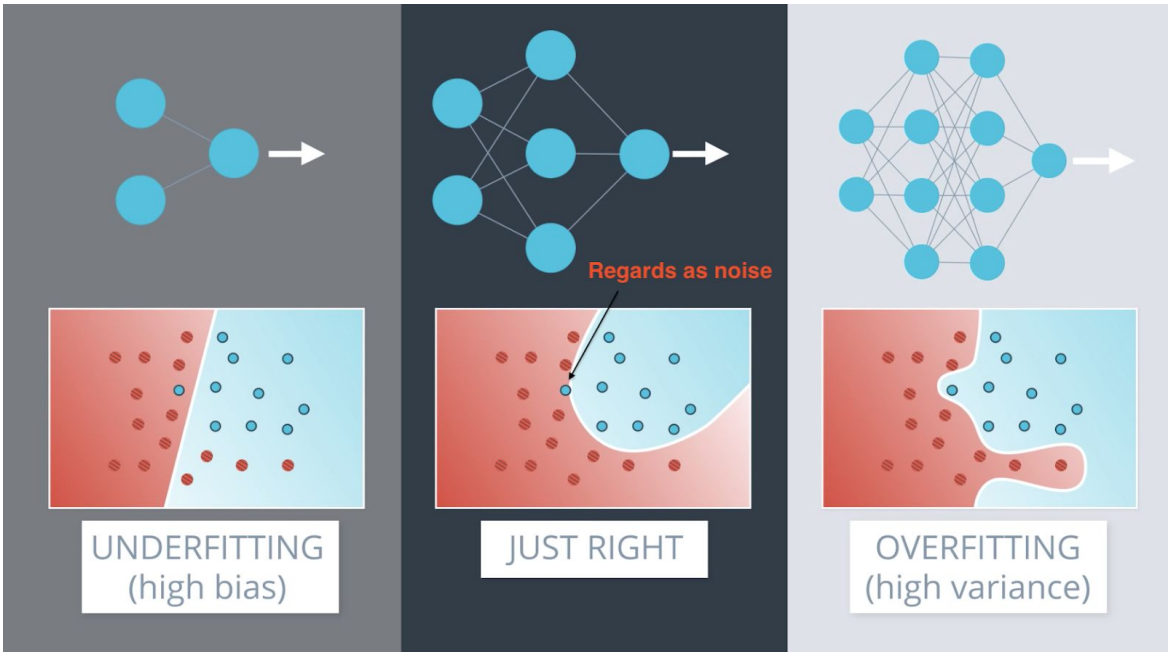
- Need optimization when:
    - Poor architecture
    - Data noisy
    - Model run too slow
    - Etc

## Training and Testing

- To know if the model is good or bad
- Train model with training set
- Validate results with testing set
- Simple model is a choice unless the result is bad

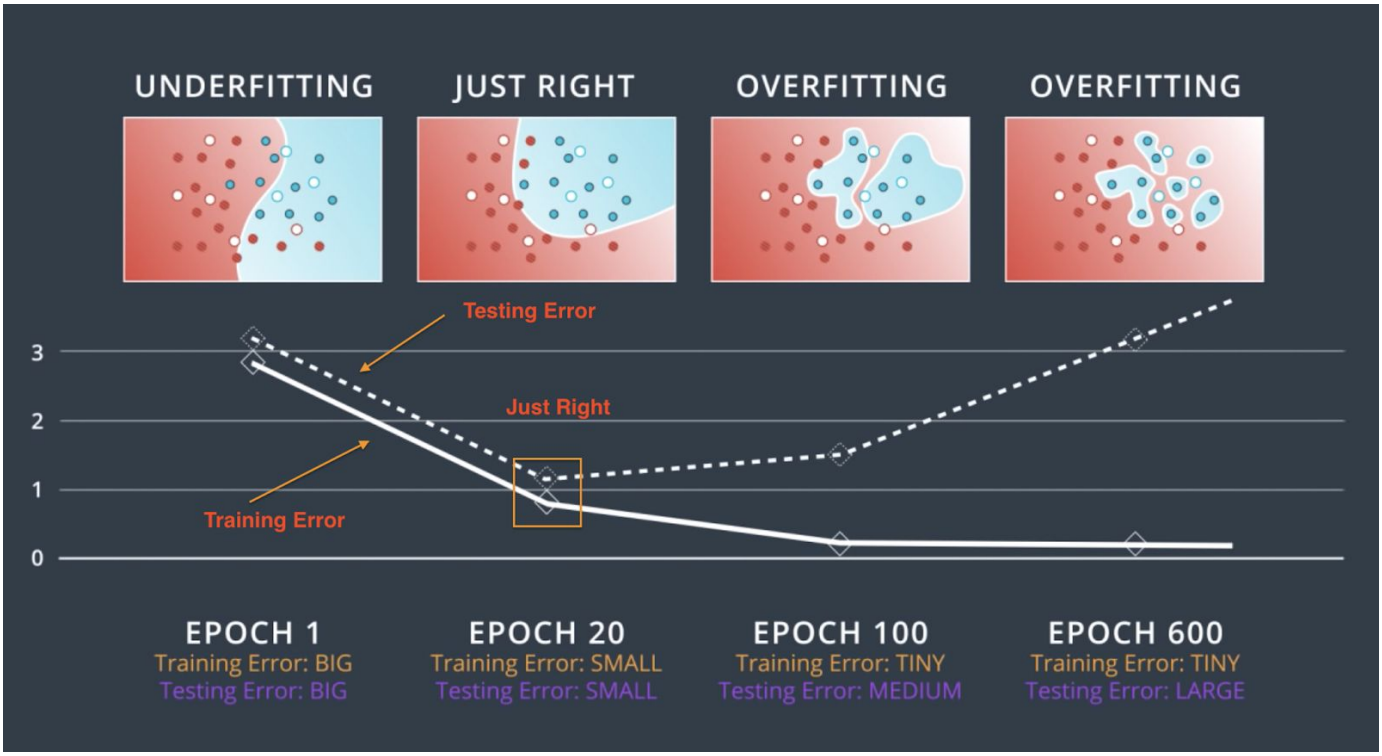
## Overfitting and Underfitting

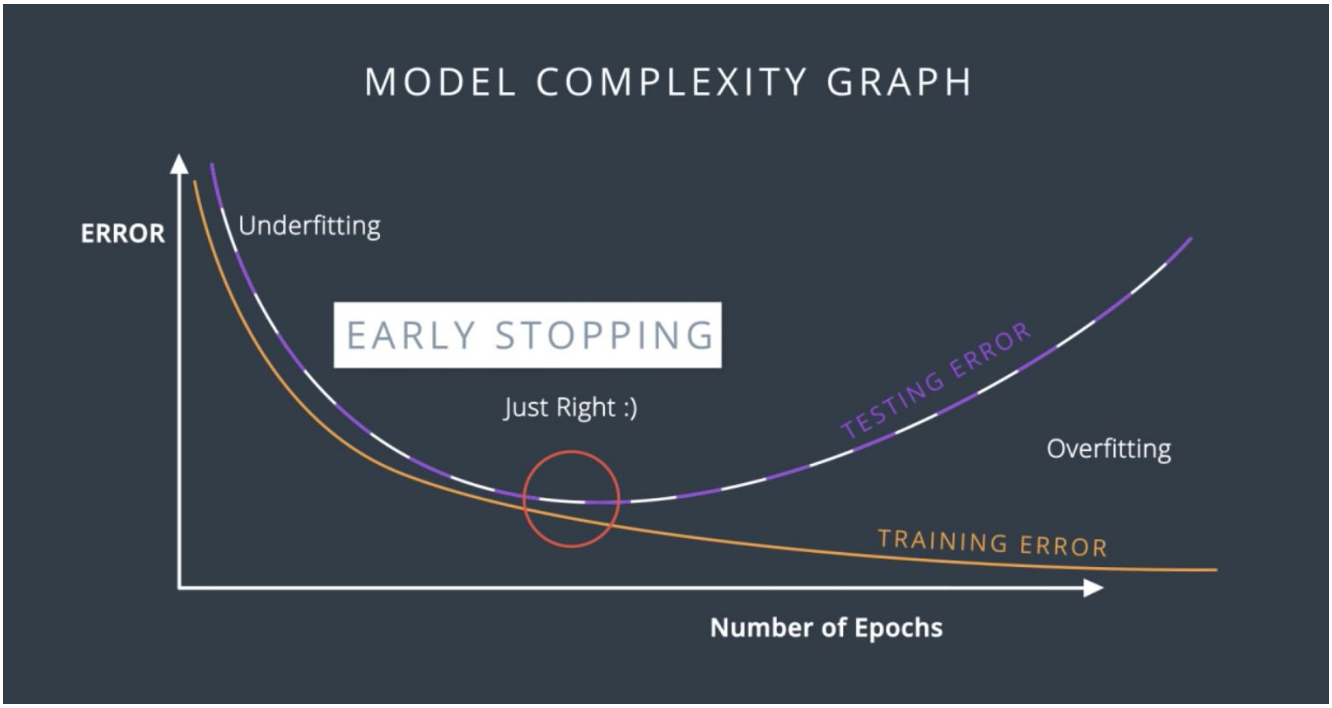
- Rules/models that is too simple could cause **underfitting** = error due to bias
- Rules/models that is too specific could cause **overfitting**
    - Introducing testing set
    - Fit data well, fail to generalize
- Analogy: studying for an exam
    - Good model, studying well and doing well in the example
    - Underfitting: not studying enough and failing
    - Overfitting: memorized text book word by word, can't generalize properly and answer the questions in the test



## Model Complexity Graph

- Problem: how to find the JUST RIGHT above?
- Solution: determine the number of Epoch ?? (look at the picture below)



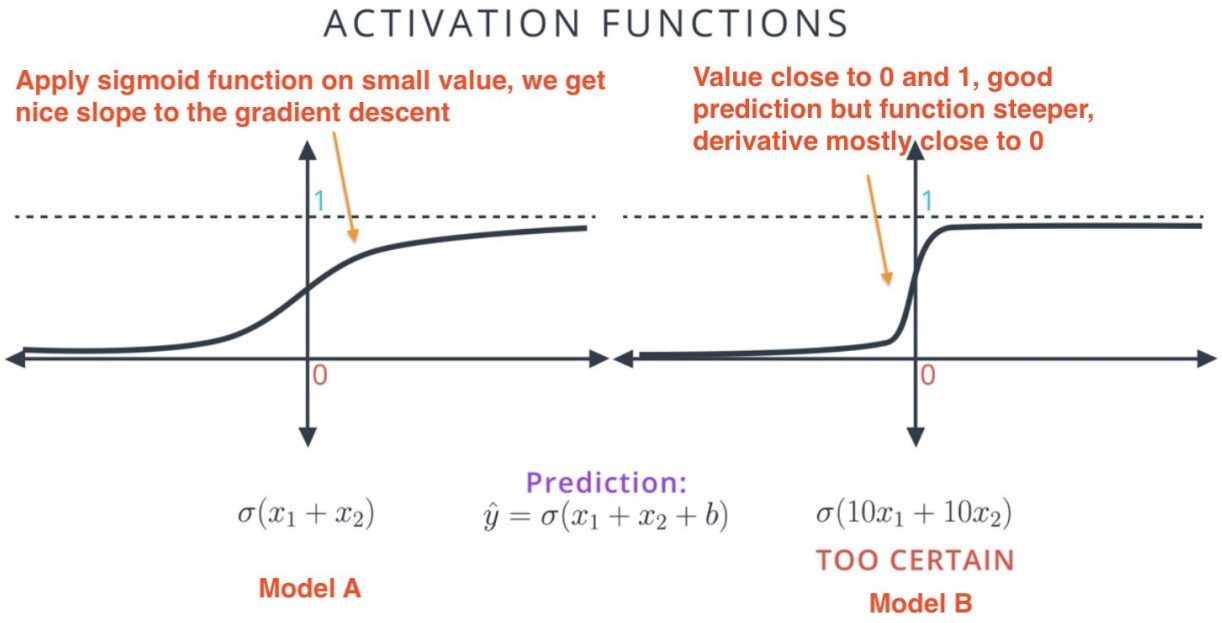



- Decent until the **testing error** stops decrease and start increase, that moment we stop.
- This procedure is called Early Stop Algorithm

## Prevent overfitting:

### 1. Regularization

- Use sigmoid function to calculate the error vai prediction function
- Some time the function give low error could be the cause of overfitting
- “The whole problem with AI is that bad models are so certain of themselves and good model so full of Doubts” -- BertrAIND Russel




- Model A is better than model B
- Model B is too certain:
    - Give little room to apply gradient descent
    - The misclassified point generate large error that make it hard for us to tune the model to correct them

- How do prevent overfitting to happening?
    - Regulatorization -- tweak error function and punish high coefficients / weights

# Solution: Regularization

LARGE COEFFICIENTS ⟶ OVERFITTING

PENALIZE LARGE WEIGHTS

$$(w_1, ..., w_n)$$

Lambda

**L1** ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i)+y_i ln(\hat{y}_i)+\lambda(|w_1|+...+|w_n|)$

Lambda

**L2** ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i)+y_i ln(\hat{y}_i)+\lambda(w_1{}^2+...+w_n{}^2)$

- Lambda is large when the weights are large
- Lambda tell us how much we want to penalized the coefficients
- The larger the value of lambda, the more we want to penalize the coefficients
- The smaller the value of lambda, the less we want to penalize the coefficients

- **When to use L1 and L2 ?**
    - Depends on goals / applications
    - L1 => sparse vector => small weight tends to go to 0
        - Want to reduce the number of weights and end up with small set
        - Feature selection, L1 helps decide which features are important and turn the rest of features to 0
    - L2 => not favor sparse vector => try to maintain all the weights homogeneously small
        - Better result for training model

## 2. Dropout

- Another way to prevent overfitting
- Problem: when training the network, some time one part of network has very large weights and it ends up dominating all the training, while another part of network not much so it doesn't get train
- Solution:
    - During training turns one part off, and let the rest of the network train
    - In each epoch, we don't allow the network to use certain node so that others. We'll give the algorithm a parameter, the probability that each node gets dropped at a particular epoch.
        - For example: if the parameter = 0.2 means that each node gets turned off with a probability of 20 percent
        - Will all nodes get turned off - not really -- but on average each node has the same treatment

==========

**Mini Summary**

- Why need training optimization ?
- What are the roles of training and testing set ?
- Overfitting vs underfitting
- How to find the JUST RIGHT (not overfitting or underfitting)
- How to prevent overfitting?