

Regularization, Bias, Variance, etc...

Olivier Dubrule and Navjot Kukreja

Objectives of the Day

- Underfitting and Overfitting in Relation to Bias and Variance
- Regularization to Control Variance/Overfitting
- Batch Normalization for Acceleration of Training and Regularization
- Machine Learning Diagnostics for Hyperparameters Optimization
- Training set, Validation Set and Test Set for Machine Learning Diagnostics
- k-Fold Validation as a Machine Learning Diagnostics
- Approaches for Hyperparameter Search

Regularization, Bias and Variance

- 1. Overfitting and Underfitting, Bias and Variance**
 - 2. Regularization**
 - 3. Batch Normalization**
 - 4. The Need for Machine Learning Diagnostics**
 - 5. Training Set, Validation Test and Test Set**
 - 6. K-Fold Validation**
-

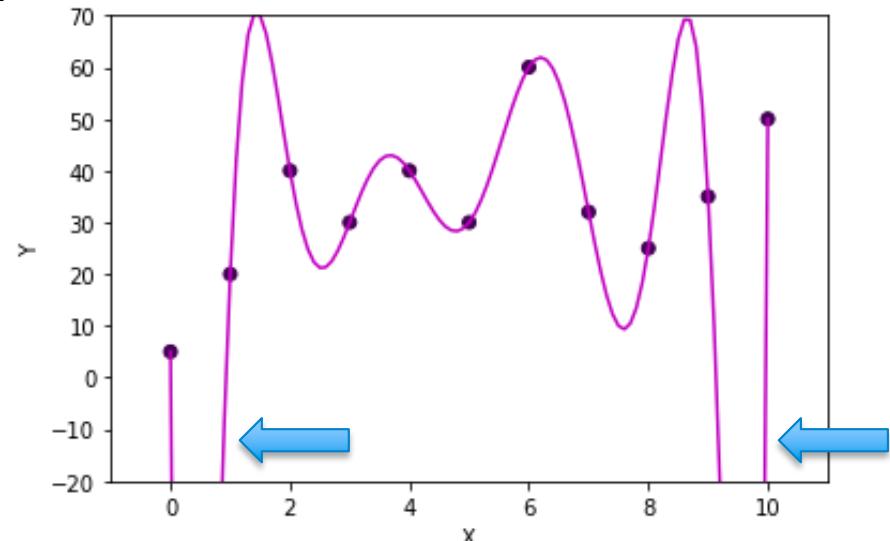
The Training Set Error is not an indicator of how well the fitted model is going to work on other data

If we have a large number of fitting parameters, the trained model may fit the training set very well:

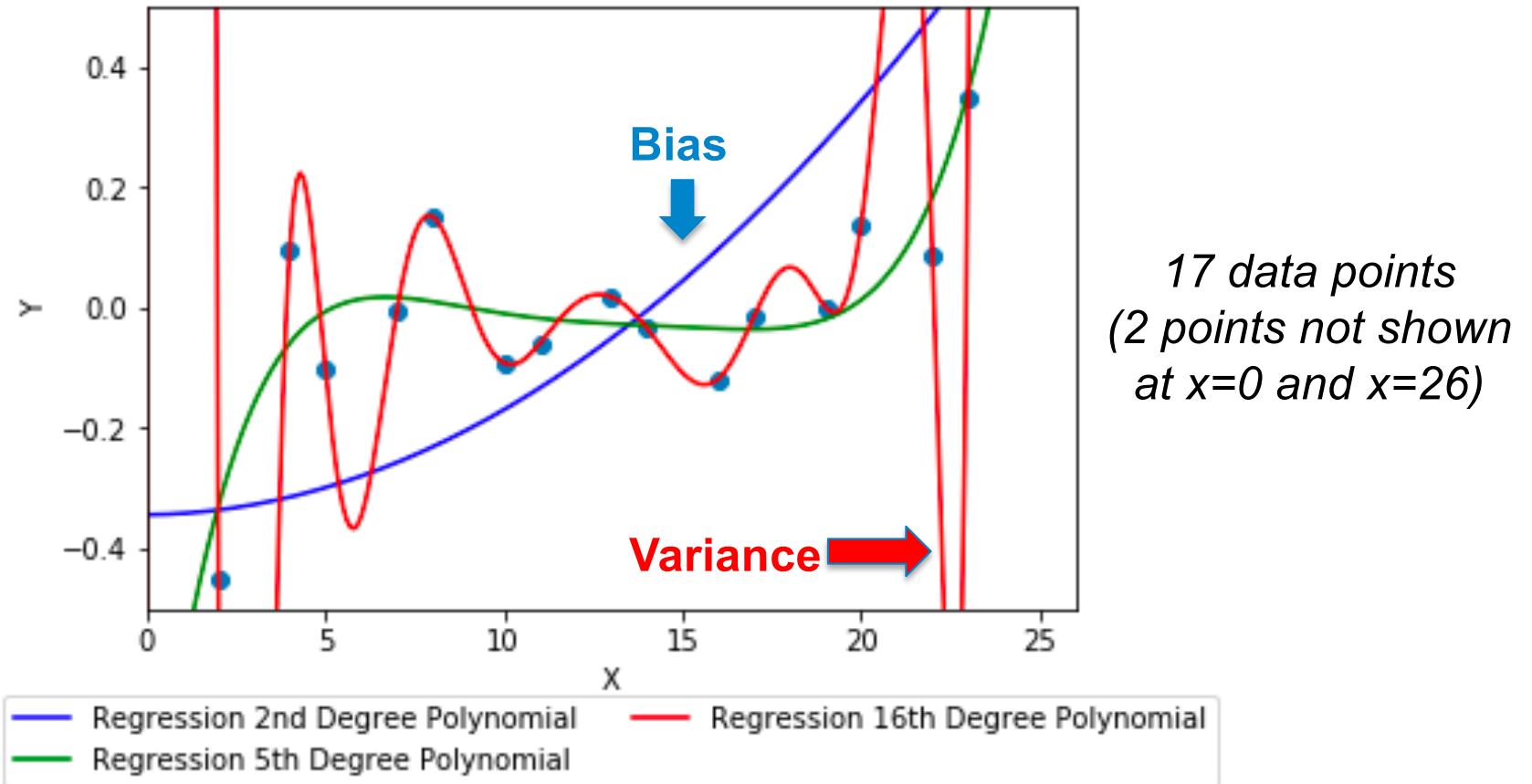
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 = 0$$

But fail to generalize to new data! This is called *Overfitting*.

Example of Polynomial Overfitting



Bias (or Underfitting) vs Variance (or Overfitting)

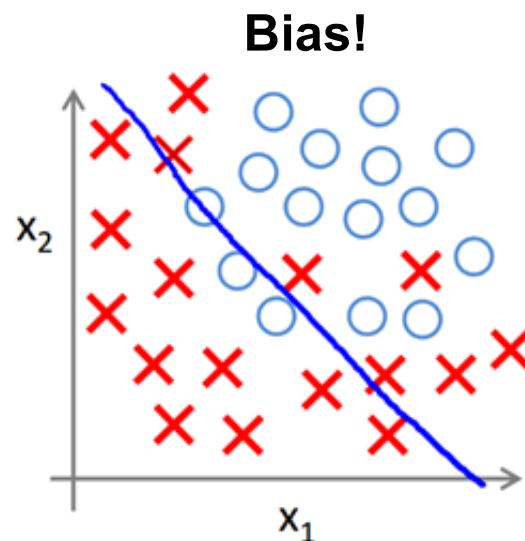


Why the terms « Bias » and « Variance »?

Bias: the hypothesis function $h_\theta(x)$ has a « pre-conception » or an « a priori » idea of the data variations which is too simple, which is biased from the start, considering the actual variability of the data (for instance it is a polynomial of too low degree).

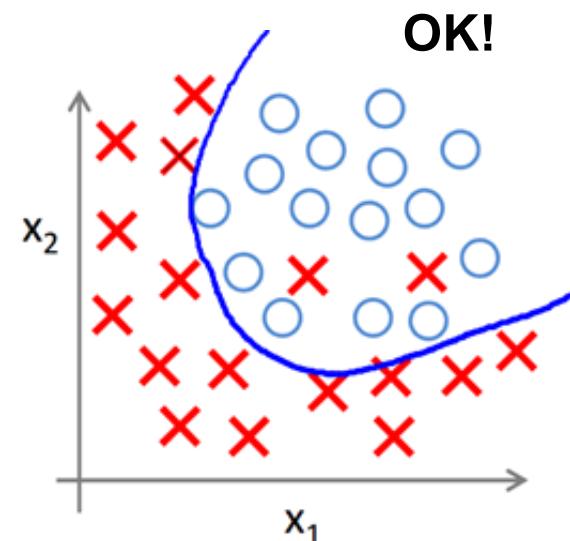
Variance: the hypothesis function $h_\theta(x)$ has too many degrees of freedom – or parameters - and, as a result, can fit too many possible functions, with too much variance, considering the actual variability of the data. (for instance it is a polynomial of too high degree)

Under/Over-fitting may also apply to Classification

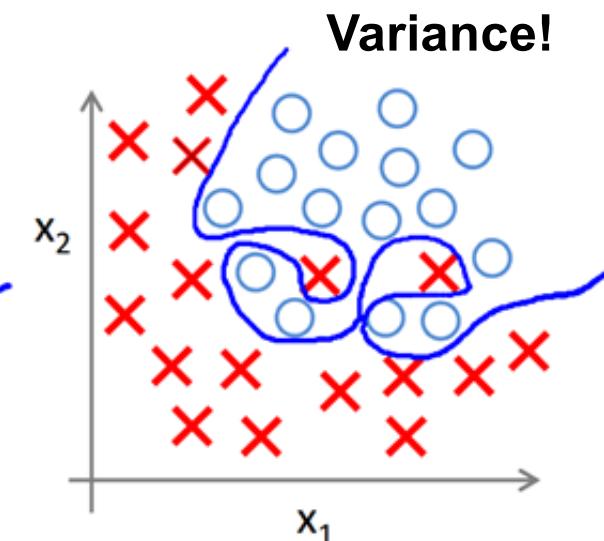


$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

(g = sigmoid function)



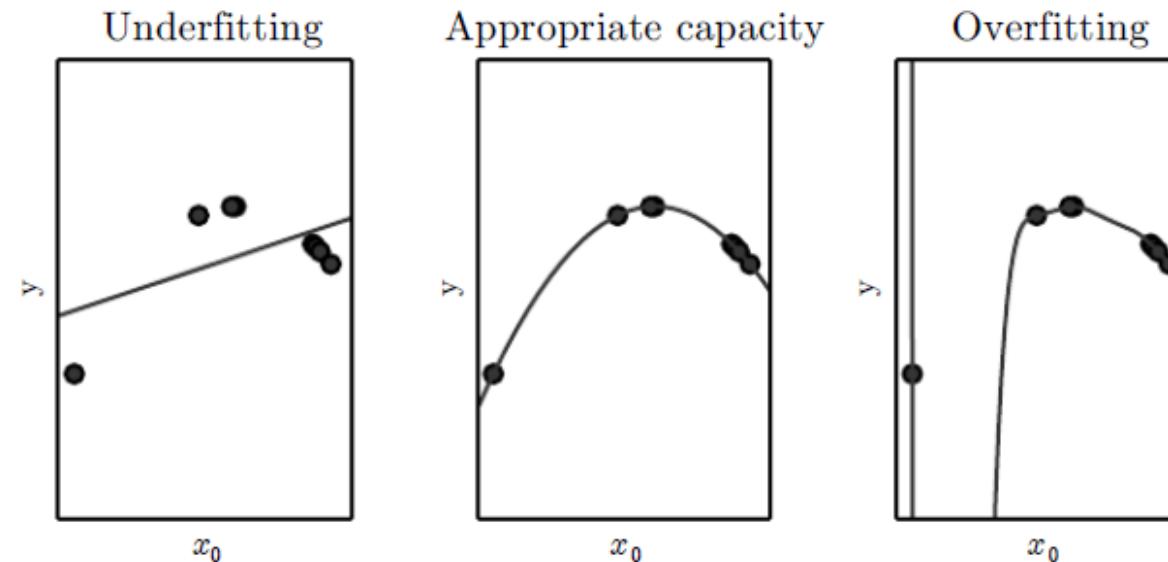
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

Capacity of a Model

The **Capacity** of a model is associated with its likely underfitting or overfitting. Informally a model's capacity is its ability to fit a wide variety of functions. Models with low capacity may tend to **underfit (Bias)** because they do not have enough parameters , models with high capacity may tend to **overfit (Variance)** because they have too many parameters.



The Generalization Error

*The trained model must perform well on new, previously unseen data, not just those on which the model was trained. The ability to perform well on **previously unseen** data – or **Test Data** - is called **Generalization**.*

Underfitting and Overfitting

A good Machine Learning algorithm must:

1. Make the Training Error small. If this is not the case, we have Underfitting.
2. Make the gap between Training and Test – or Generalization - Error small. If this is not the case we have Overfitting.

Regularization, Bias and Variance

- 1. Overfitting and Underfitting, Bias and Variance**
 - 2. Regularization**
 - 3. Batch Normalization**
 - 4. The Need for Machine Learning Diagnostics**
 - 5. Training Set, Validation Test and Test Set**
 - 6. K-Fold Validation**
-

One Way to Avoid Overfitting: Regularization

Regularization is any modification we make to a learning algorithm that is intended to reduce its Generalization Error but not its Training Error...

An effective regularizer is one that reduces Variance significantly while not increasing the Bias.

Goodfellow et al, 2017

(L1 or L2) Regularization for Regression ML

L1 and L2 Regularizations consist of adding a new term to the objective function in order to control the variations of the parameters:

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right) \quad \text{if L2 norm is used}$$



Regularization Parameter,
controlling the “Weight Decay”



$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right) \quad \text{if L1 norm is used}$$

The well-known Ridge & Lasso Regressions

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Ridge Regression (preferred because math derivations simpler)

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right) \text{ if L2 norm is used}$$

Lasso Regression

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n |\theta_j| \right) \text{ if L1 norm is used}$$

Classification with L2 Regularization: Exercise

Take the Case of Binary Linear Logistic Regression in 2-D.

Write $h_\theta(x)$ as a function of x and θ .

$$h_\theta(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}}$$

Assume there are m data points $(x^{(i)}, y^{(i)})$ for $i = 1, \dots, m$

Express $J(\theta)$ using L2 regularization and a regularization parameter λ .

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} (\theta_0^2 + \theta_1^2 + \theta_2^2)$$

Calculate $\frac{\partial J(\theta)}{\partial \theta_j}$ for one of the θ_j parameters:

$$\frac{\partial J(\theta)}{\partial \theta_2} = \frac{1}{m} \sum_{i=1}^m [h_\theta(x^{(i)}) - y^{(i)}] x_2^{(i)} + \frac{\lambda}{m} \theta_2$$

Gradient Descent for Regularized Logistic Regression

Gradient Descent Approach without Regularization Term

$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j} := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Gradient Descent with Regularization Term (if L2 norm is used)

$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j} := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} - \alpha \frac{\lambda}{m} \theta_j$$

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Gradient Descent with Regularized Regression

Consider in more detail the Gradient Descent term in case of Regularization:

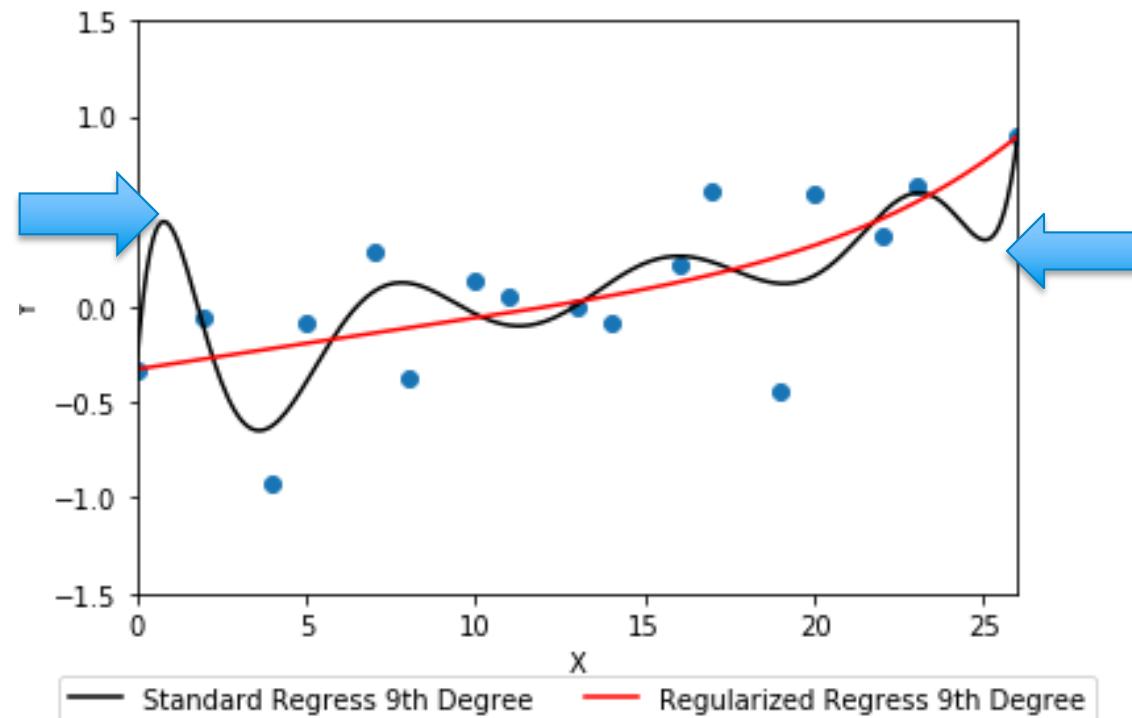
$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$



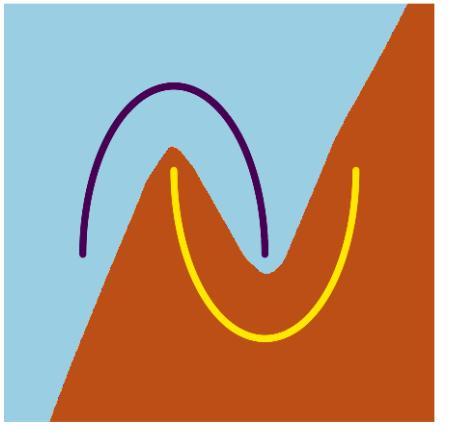
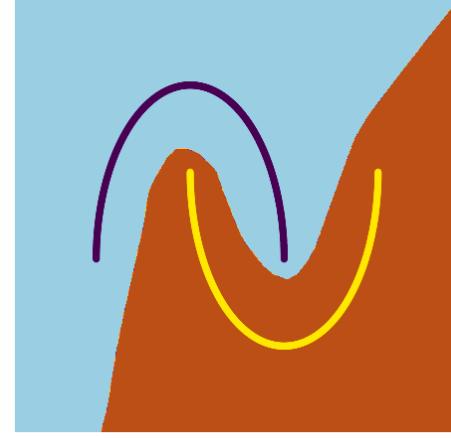
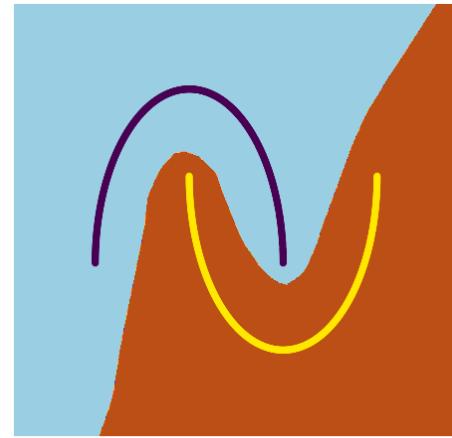
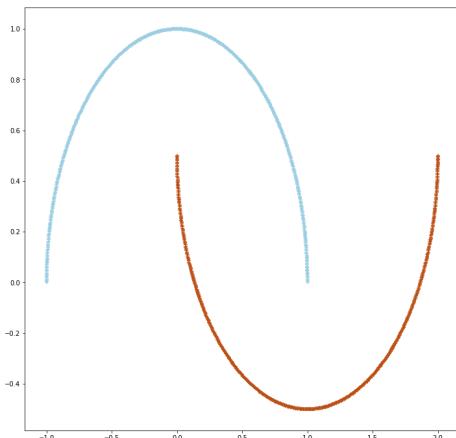
Systematic decrease of absolute value θ_j at each iteration

Same term as for optimization without regularization

The Impact of Applying Regularization to Regression



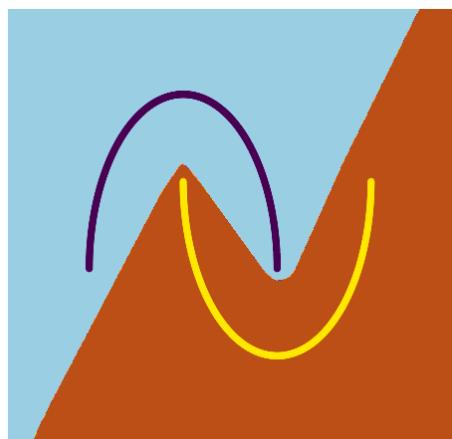
A Simple Neural Network Regularization Example (1)



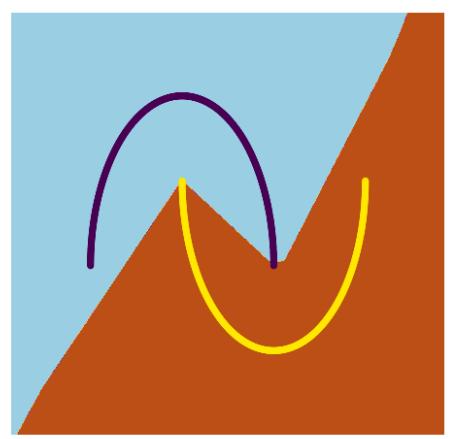
$\lambda = 0$

$\lambda = 0.1$

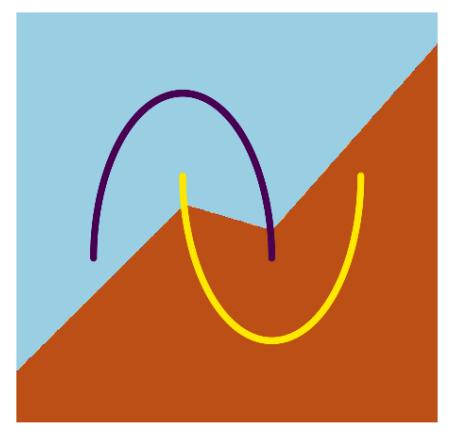
$\lambda = 1$



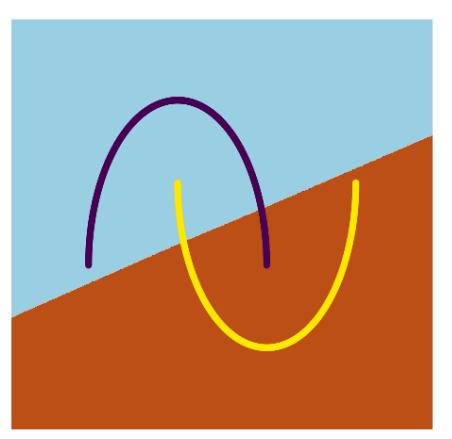
$\lambda = 2.5$



$\lambda = 5$

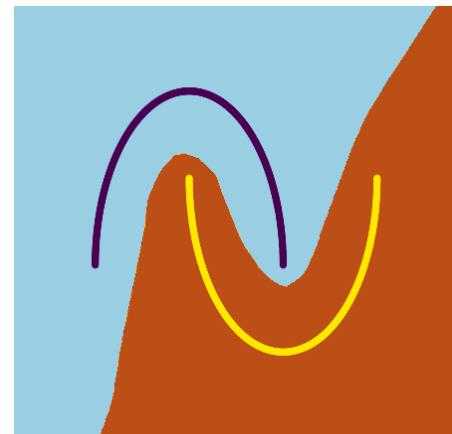
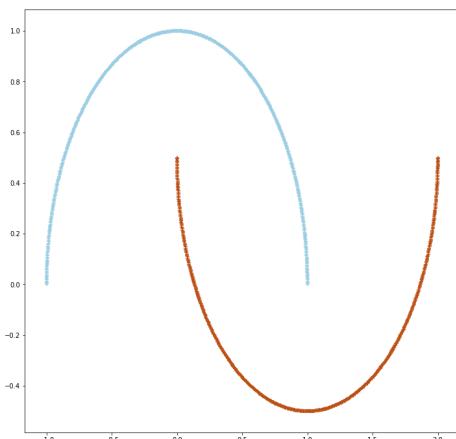


$\lambda = 7.5$



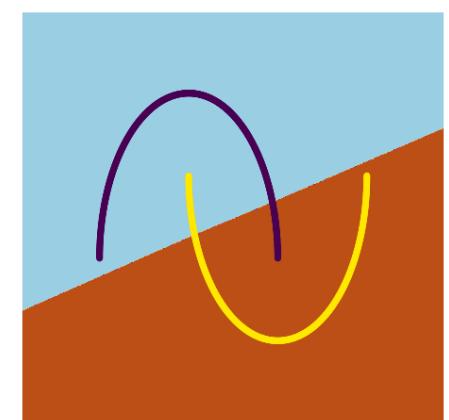
$\lambda = 10$

A Simple Neural Network Regularization Example (2)



$\lambda = 0$

Increasing λ

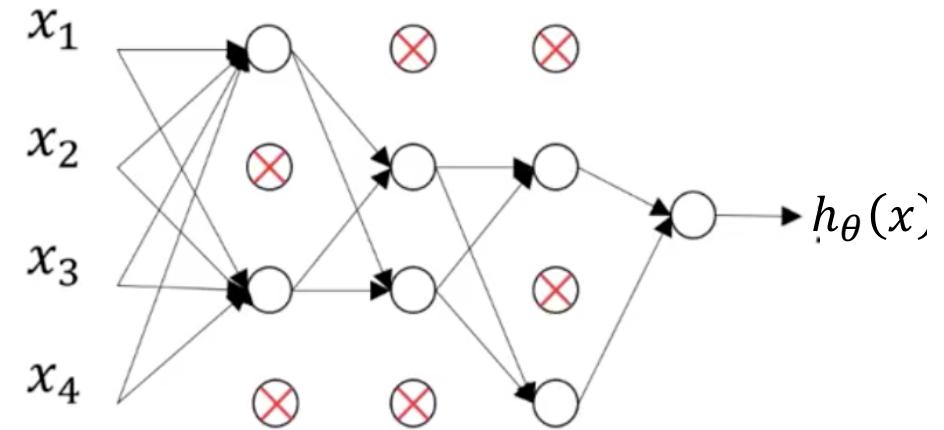
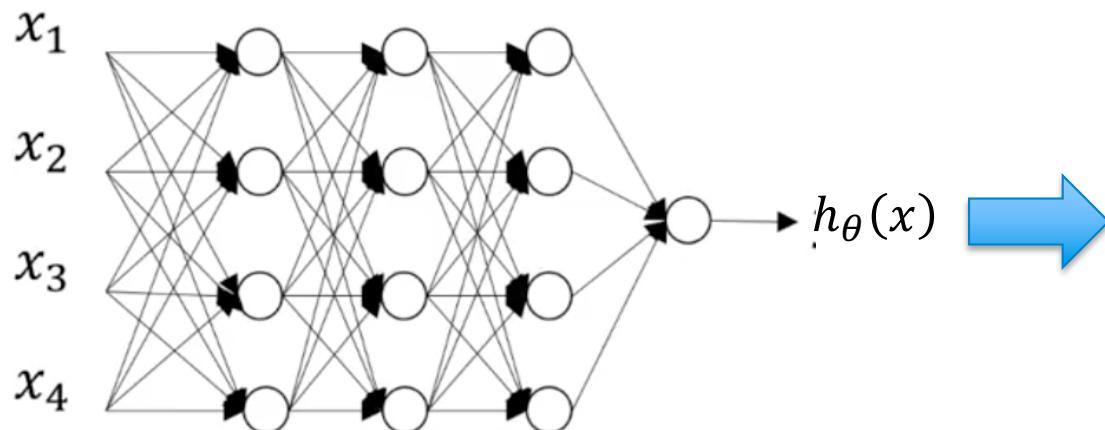


$\lambda = 10$

As λ increases, the values of the weights tend to zero, and the Decision Boundary first becomes a broken Line, then a single Line.

Another Regularization Approach: Drop-Out (1)

At Training time:



For each Training example...

Drop hidden layers units with 0.5 (or p) probability

Same approach for input layer but with p closer to 1. No change in output layer.

We are averaging the results over a set of network configurations!

Classic Dropout Paper (>18000 citations!)

Journal of Machine Learning Research 15 (2014) 1929-1958

Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava

Geoffrey Hinton

Alex Krizhevsky

Ilya Sutskever

Ruslan Salakhutdinov

Department of Computer Science

University of Toronto

10 Kings College Road, Rm 3302

Toronto, Ontario, M5S 3G4, Canada.

NITISH@CS.TORONTO.EDU

HINTON@CS.TORONTO.EDU

KRIZ@CS.TORONTO.EDU

ILYA@CS.TORONTO.EDU

RSALAKHU@CS.TORONTO.EDU

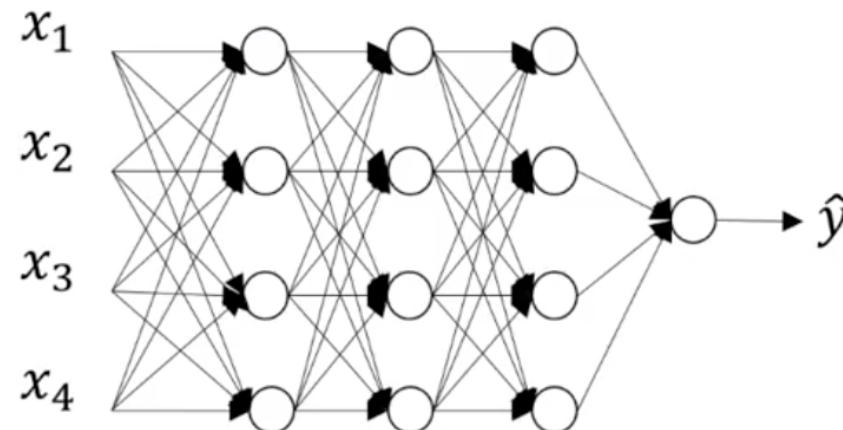
Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many

Another Regularization Approach: Drop-Out (2)

At Test time: no drop-out, use all hidden units but multiply their output by $p = 0.5$



*Drop-out can also be used to quantify Model Uncertainty in Bayesian Neural Networks
(Gal and Ghahramani, 2016)*

Another Regularization Approach: Data Augmentation



Software Tools for Data Augmentation

A number of basic geometrical image transformations are likely to change the appearance of the image, but not its class:

- Rotation (to some reasonable degree)
- Translation (up, down, left, right)
- Zooming
- Cropping
- Added noise
- Changing the Brightness level
- ...

These are readily available in Python code (*imgaug*, *Albumentation* packages...).

Regularization, Bias and Variance

- 1. Overfitting and Underfitting, Bias and Variance**
 - 2. Regularization**
 - 3. Batch Normalization**
 - 4. The Need for Machine Learning Diagnostics**
 - 5. Training Set, Validation Test and Test Set**
 - 6. K-Fold Validation**
-

Normalization vs Standardization Common Definitions

Normalization

$$\frac{x}{x_{min} = 50 \qquad \qquad x_{max} = 500}$$



$$\frac{x'}{0 \qquad \qquad 1}$$

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Standardization

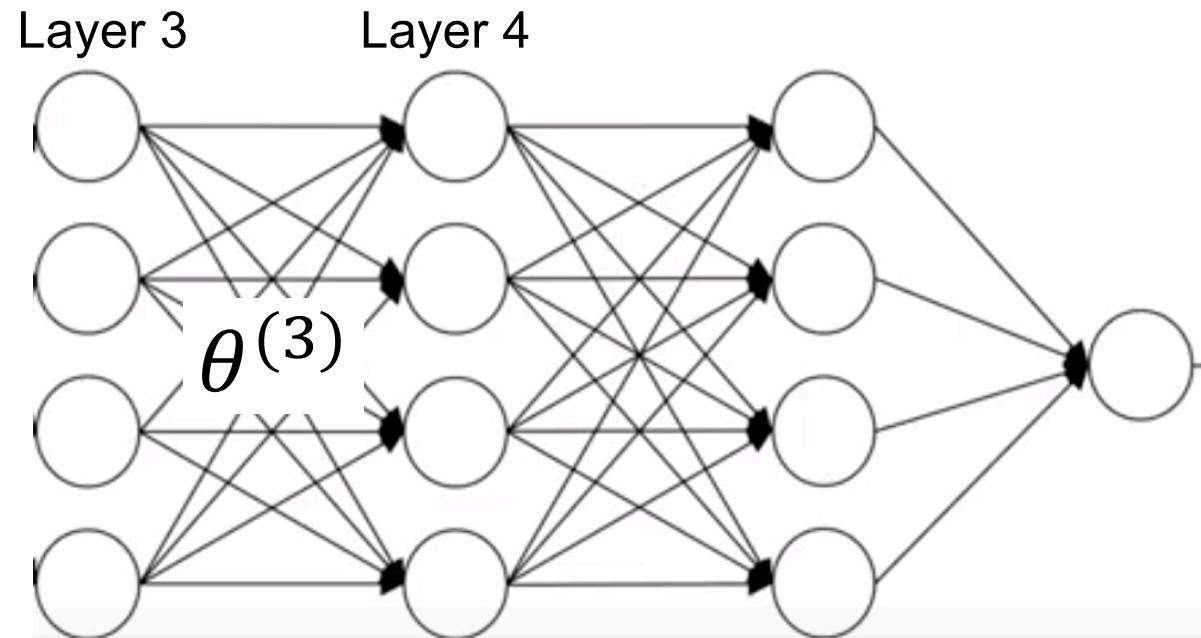
$$x' = \frac{x - \mu_x}{\sigma_x}$$

(where μ_x and σ_x are mean and standard deviation of array x)



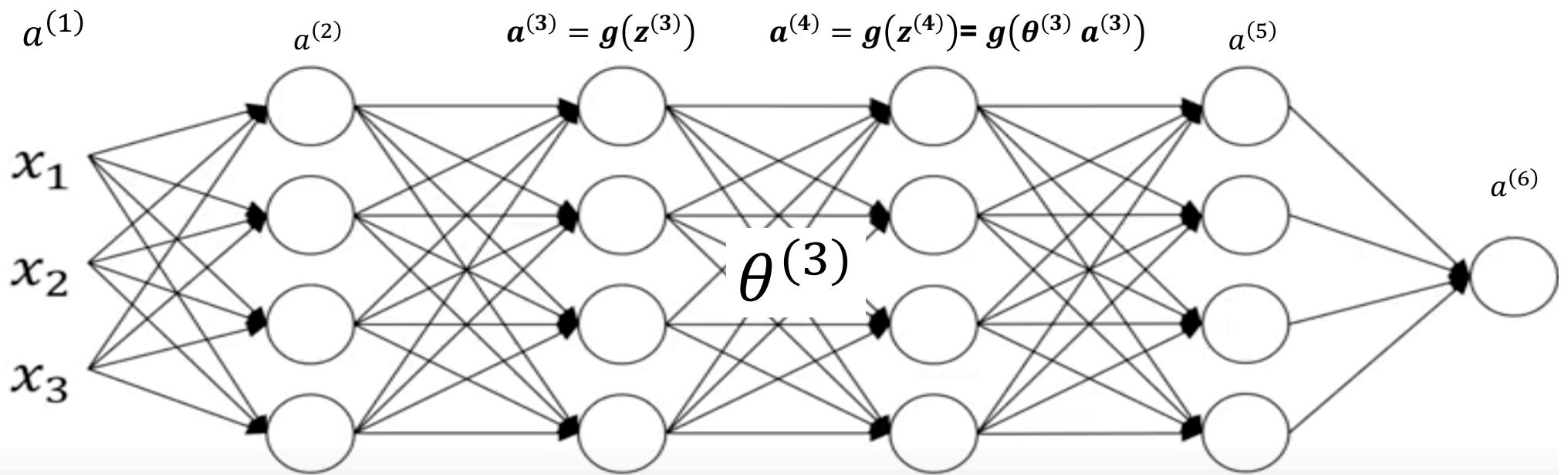
*This is the basis for
“Batch Normalization”*

Batch Normalization: Covariate Shift



Each time the distribution of the input of a neural network becomes different from the input used for training, the weights become invalid (this is like Training a network on Black Cats and applying it to Coloured Cats).

Batch Normalization for Hidden Layers



Can we normalize $z^{(3)}$ (or $a^{(3)}$) in order to stabilize the calculation of weights $\theta^{(3)}$?

Batch Normalization on $z^{(l)}$ vector (could be on $a^{(l)}$)

1. Normalize $z^{(l)}$ vector at layer (l):

$$z_{norm}^{(l)} = \frac{z^{(l)} - \mu_z}{\sqrt{\sigma_z^2 + \varepsilon^2}}$$

(with μ_z and σ_z^2 mean and variance of $z^{(l)}$ calculated separately on each mini-batch,
and ε small parameter useful if $\sigma_z^2 = 0$)

2. Apply Linear Transform to $z_{norm}^{(l)}$, with trainable parameters $\beta^{(l)}$ and $\gamma^{(l)}$. For each component j of $z_{norm}^{(l)}$

$$\tilde{z}_j^{(l)} = \gamma_j^{(l)} z_{j norm}^{(l)} + \beta_j^{(l)}$$



Optimize mean and variance of $\tilde{z}^{(l)}$

3. Use $\tilde{z}^{(l)}$ as input to the activation function: $a^{(l)} = g(\tilde{z}^{(l)})$

Batch Normalization

- Performs pre-processing at each layer of the network, instead of just on input layer
- Accelerates Training by stabilizing the layer changes through the iterations
- Provides some kind of Regularization, as Drop-Out, but not as strong.

Classic Batch Norm Paper (>15000 citations!)

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., sioffe@google.com

Christian Szegedy
Google Inc., szegedy@google.com

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

1 Introduction

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochas-

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers’ inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

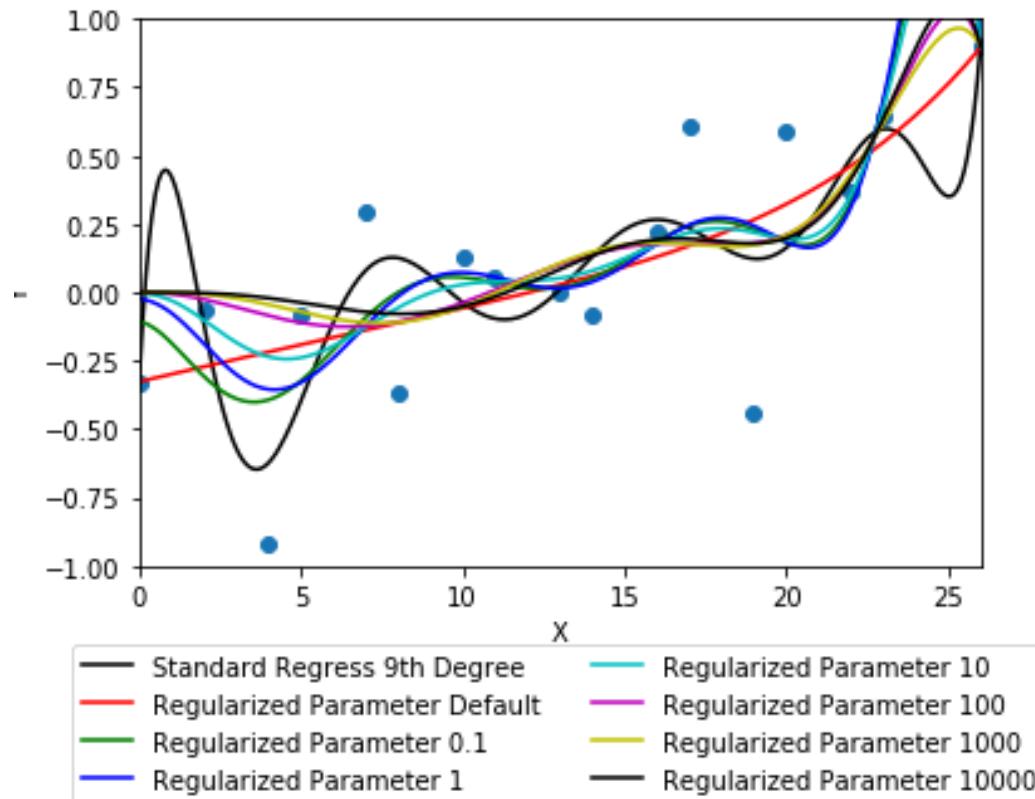
$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize

Regularization, Bias and Variance

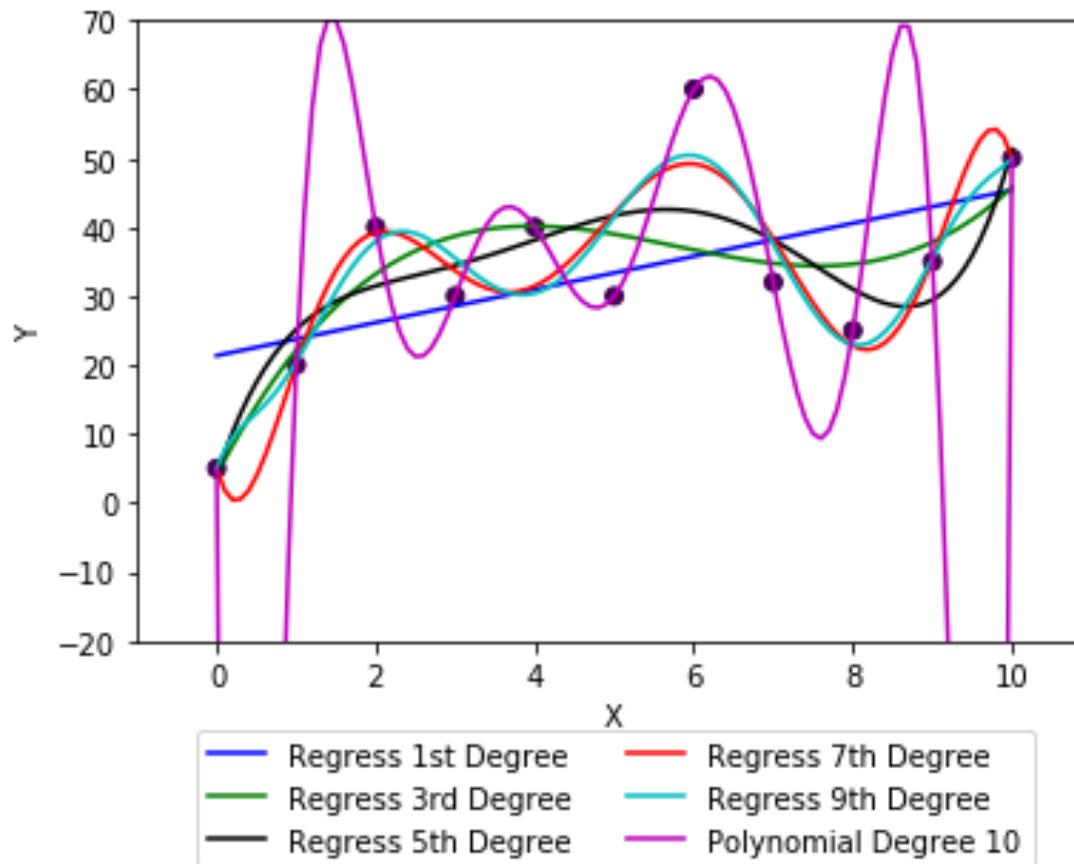
- 1. Overfitting and Underfitting, Bias and Variance**
 - 2. Regularization**
 - 3. Batch Normalization**
 - 4. The Need for Machine Learning Diagnostics**
 - 5. Training Set, Validation Test and Test Set**
 - 6. K-Fold Validation**
-

Changing the Regularization Parameter



*How can we choose the
“Best” Regularization
Parameter?*

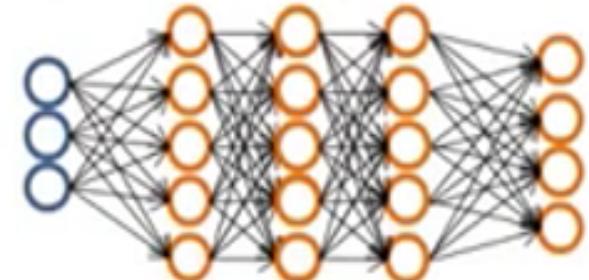
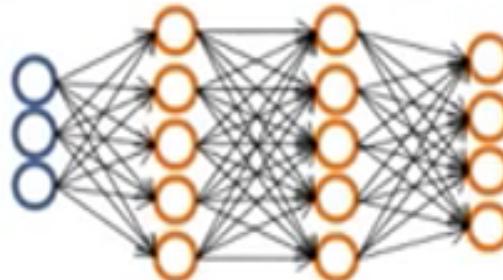
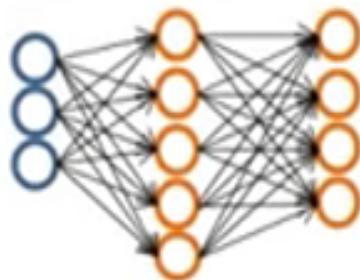
Change of Interpolating Function as Degree Increases



*How can we choose
the “Best” Polynomial
Degree?*

Change of Network Architecture

From one to three Hidden Layers



How can we choose the “Best” Network Architecture?

Machine Learning Diagnostic:

A test to gain insight about the performance of a Training algorithm.

Regularization, Bias and Variance

- 1. Overfitting and Underfitting, Bias and Variance**
 - 2. Regularization**
 - 3. Batch Normalization**
 - 4. The Need for Machine Learning Diagnostics**
 - 5. Training Set, Validation Test and Test Set**
 - 6. K-Fold Validation**
-

What is a Hyperparameter?

A **hyperparameter** is a neural network parameter whose value is set before the Training process begins. It does not change during Training. By contrast, the values of parameters θ are derived via Training.

Exercise: Give Examples of Hyper-Parameters of a Neural Network

Batch Size, Learning Rate, Optimization Approach used, Number of Layers, Number of Hidden Units, Regularization Parameter...

But how can we pick the optimal hyperparameters?

The Need for a Validation Set

In a Supervised Learning context, the Neural Network parameters (or weights) are trained on the Training Set, but tested on the Test Set.

The Test Set constitutes the unseen data, it should not be used to calculate the parameters of the Neural Network ... or to optimize the Hyperparameters.

Hence, to optimize the Hyperparameters, we need a third Set, the Validation Set!

Create Validation Set to Optimize Hyperparameters

Split Data Into Three Sets:

Training Set (~70%) , Validation Set (~15%) , Test Set (~15%)

<i>Training Set</i>	<i>Validation Set</i>	<i>Test Set</i>
----------------------------	------------------------------	------------------------

Example of MNIST: the Training Set contains 60,000 images, the « official » Test Set contains 10,000 images. This ratio is usually appropriate for this size of datasets. For very large datasets (say 100.000's to millions), split between Training and Test Set sizes can be smaller and be as low as 90%-10%.

Note that MNIST does not have a pre-defined Validation Set. This has to be chosen and extracted from the Training Set by the user.

Use Validation Set to Optimize Hyperparameters

For each Possible Choice of Neural Network Hyperparameters:

1. Train Neural Network Parameters on Training Set
2. Test Performance on the Validation Set
3. Pick the Hyperparameters that give the best performance on the Validation Set.
4. Possibly retrain the new Neural Network on Training+Validation Set.
5. Test the Neural Network on the Test Set

The Test Set is the final measure of performance but must never be used in the Training!

Warning

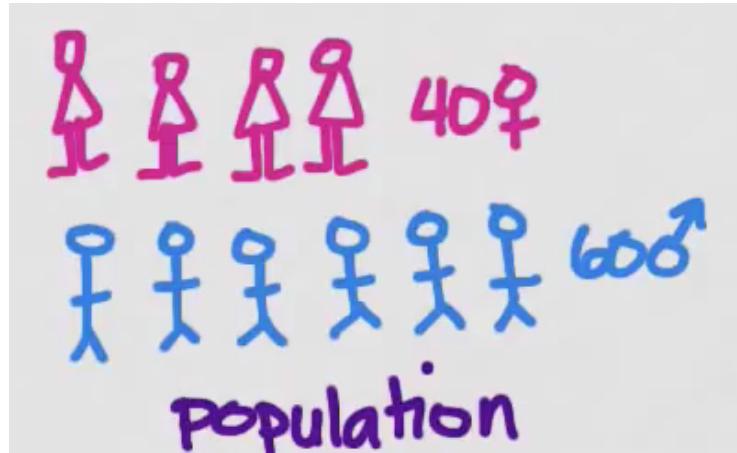
Some studies still use the Test Set to optimize the Hyper-Parameters.

This means that the ultimate Test Set Error is not representative of the Generalization Error!

Sampling the Validation Set (and possibly the Test Set)

In Classification problems, make sure the proportions of each class are the same for Training, Validation and possibly Test Sets. Take the example of a 20% Validation Set.

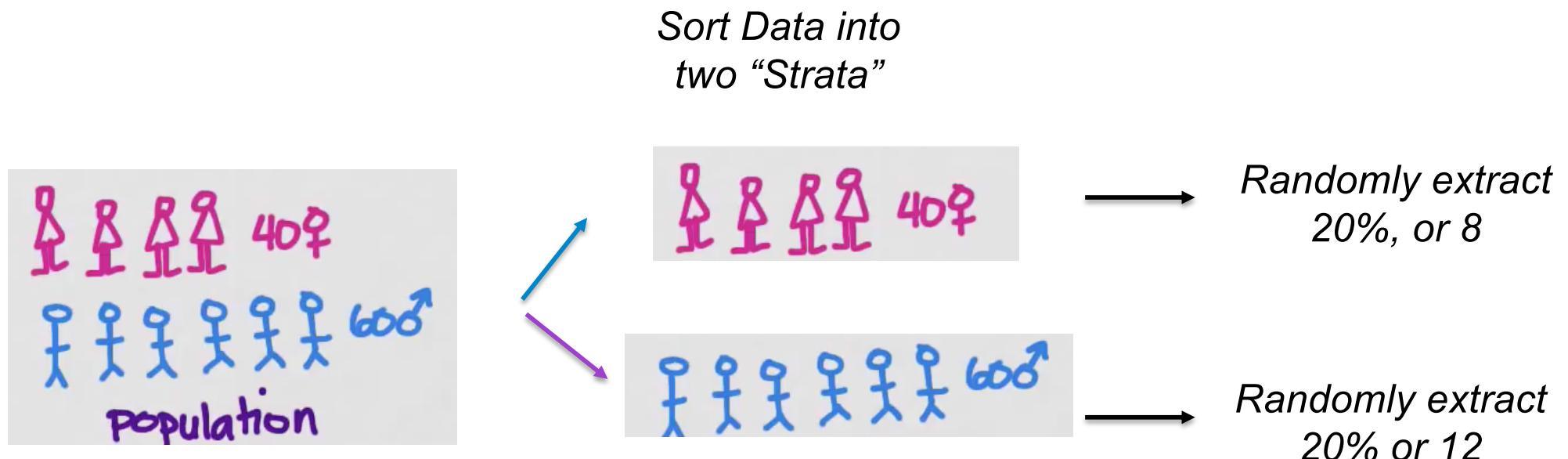
How to create a 20% Validation Set from this population?



If I randomly select 20 individuals (that is 20%) from this group of 40 women and 60 men, it is very unlikely that the group of 20 will be exactly composed of 8 (that is 20%) women and 12 (that is 20%) men!

For smaller data sets, use Stratified Random Sampling (class by class) instead of global sampling over the whole set.

Stratified Sampling: *Create Validation Set of 20%*

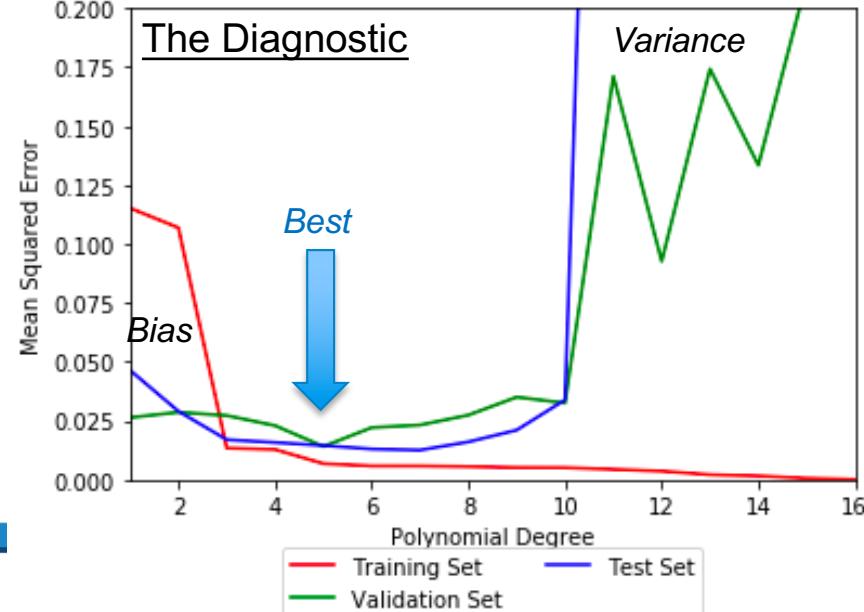
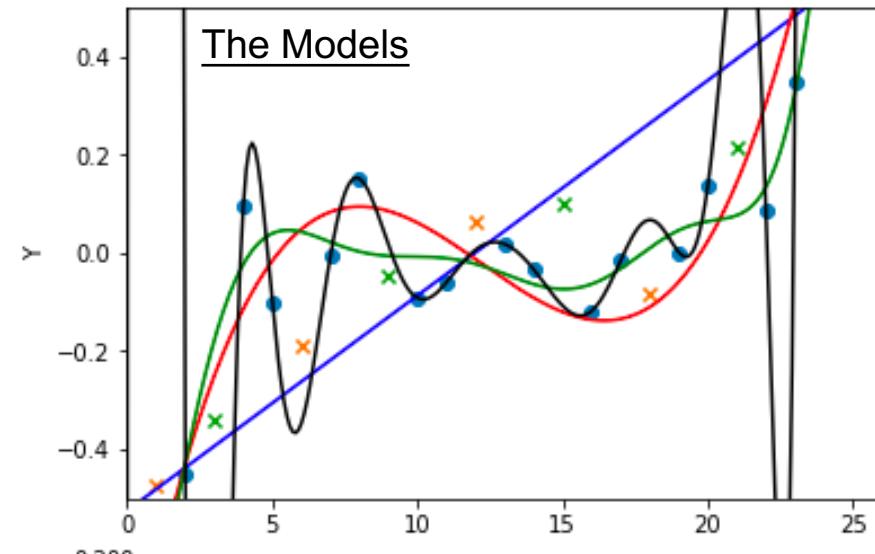


First sort the population into “strata”, then sample from each strata!

Example of Different Sets for Polynomial Fitting



17 Training data
5 Validation data
5 Test Data



$J_{train}(\theta)$ and $J_{val}(\theta)$ for picking the Regularization Parameter

The Training Set has m data points. The optimized Loss Function is (if L2 norm):

$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right)$$

For each tested value of λ , train the network on the Training Set, and evaluate the errors:

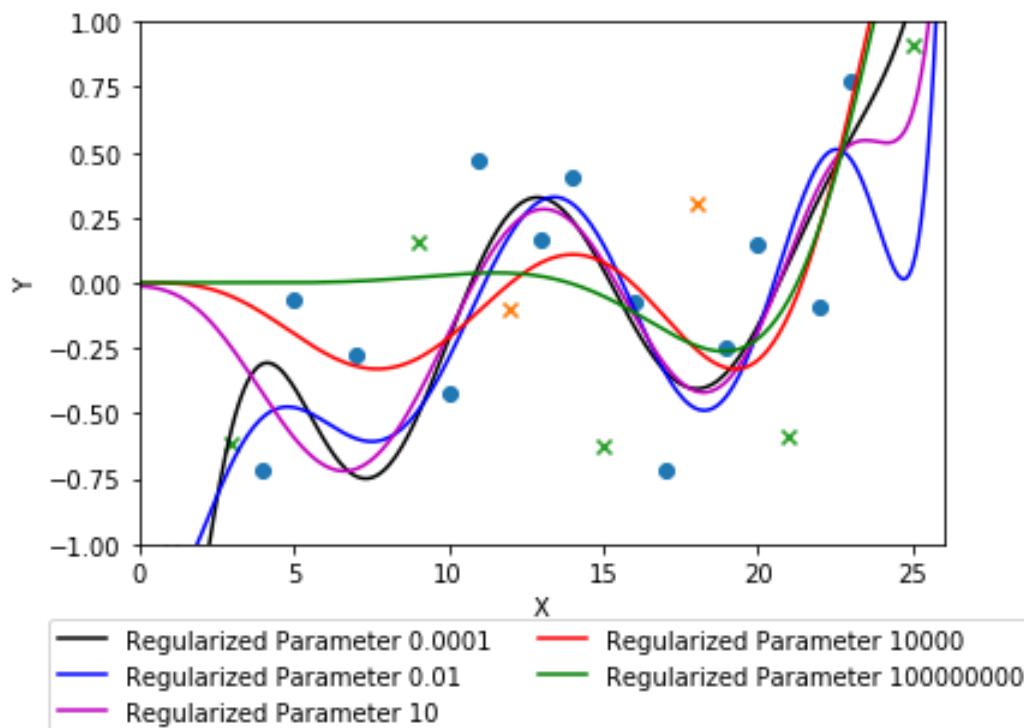
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

And on the Validation Set: $J_{val}(\theta) = \frac{1}{2m_{val}} \sum_{i=1}^{m_{val}} (h_\theta(x_{val}^{(i)}) - y_{val}^{(i)})^2$

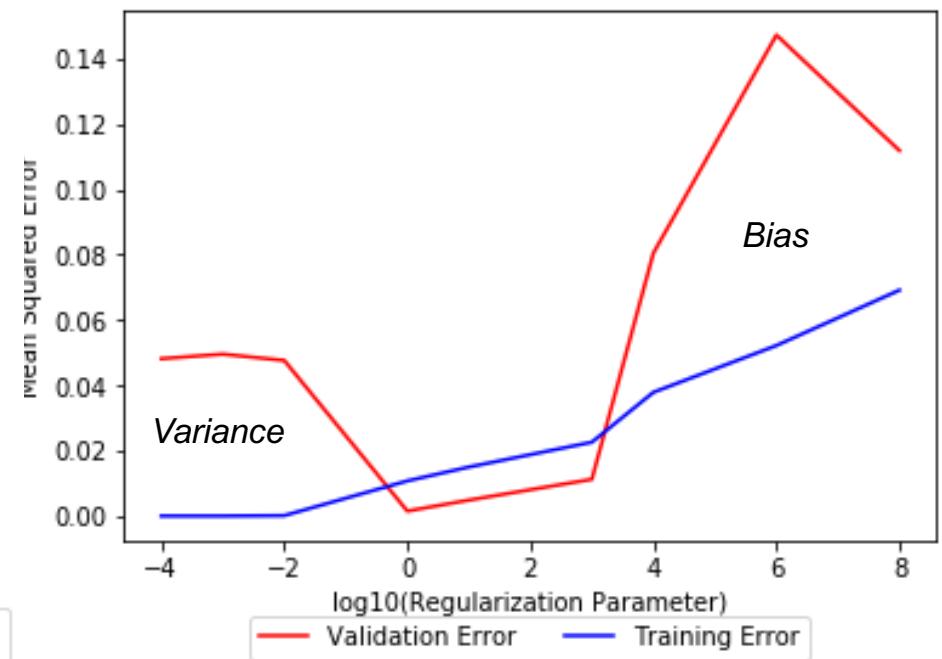
On the Test Set: $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$

Optimizing the Regularization Parameter

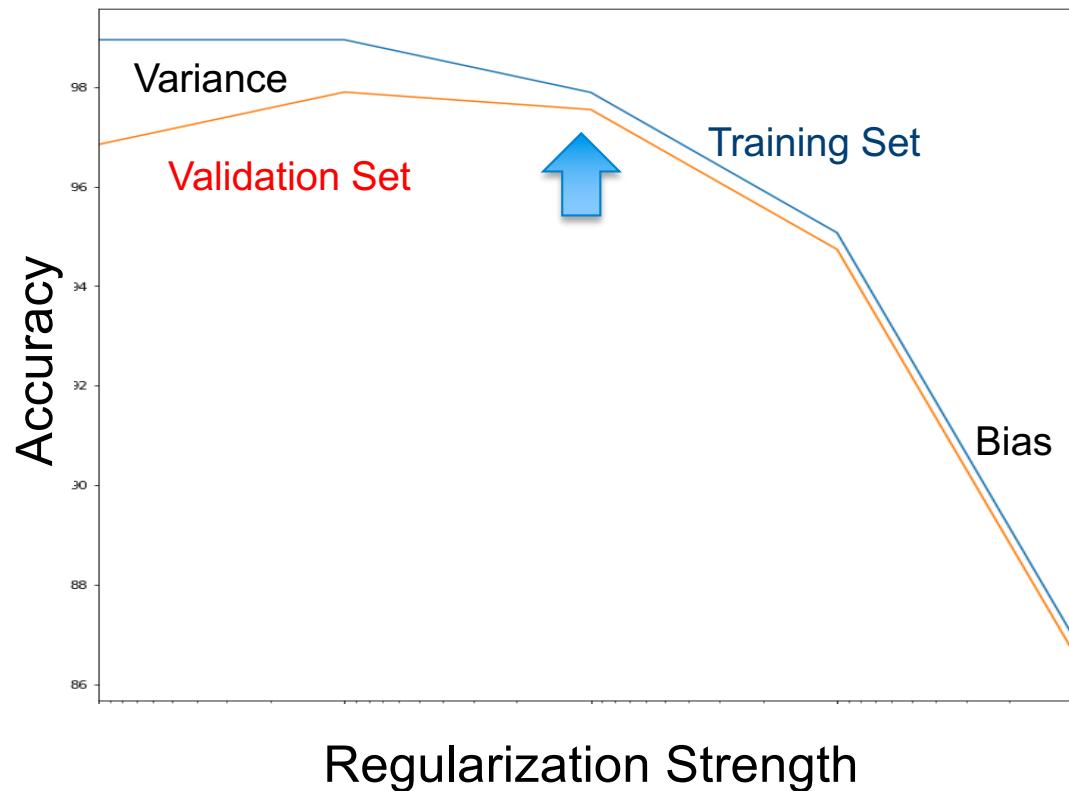
Interpolating Function



Mean Squared Error



Typical Regularization Behaviour in Classification



The Role of the Validation Set: Back to MNIST

The “official” MNIST Training Set is 60000 data. The “official” Test Set is 10000 and unknown to the user.

The user first needs to split the official Training Set into a new Training and Validation sets, for example with 50000 and 10000 data points (as many as in the Test Set) in each (because of large number of data, Random Shuffling is enough).

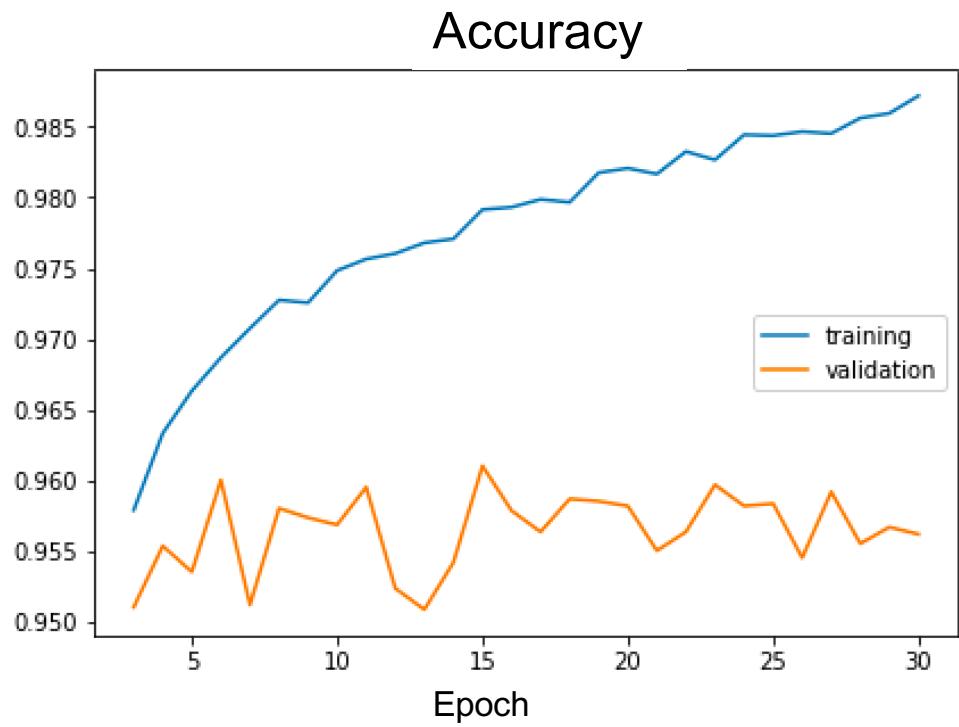
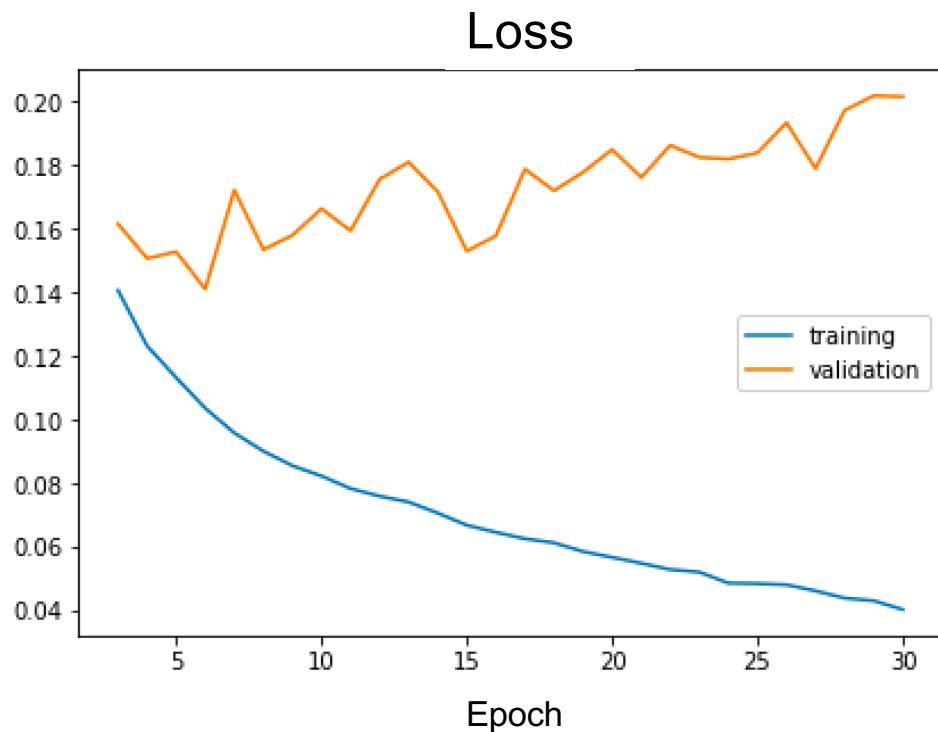
Then for a number of possible Hyperparameters:

1. Train Neural Network on Training Set
2. Test its Performance on Validation Set

Then pick the Hyperparameters that gives the optimal performance on the Validation Set. *Then possibly retrain the network on Training+Validation Sets.*

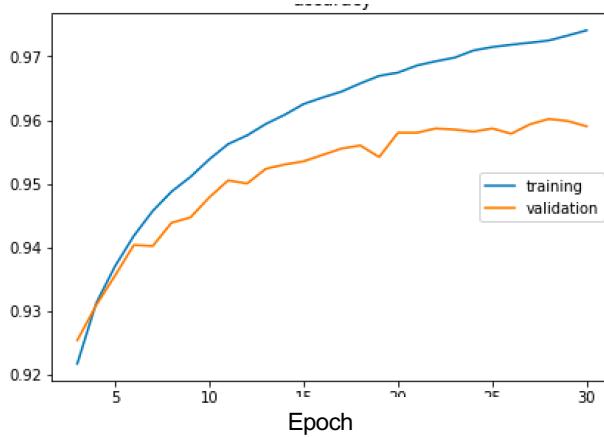
Then finally evaluate the performance of the Neural Network associated with these optimal Hyperparameters on the Test Set.

MNIST: Using the Validation Set during Training

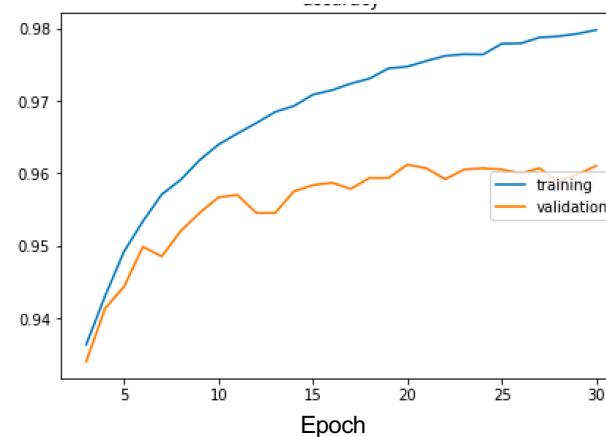


MNIST: Using the Validation Set during Training

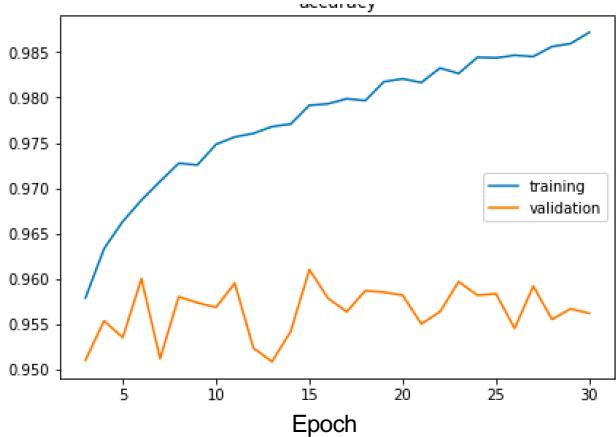
Accuracy for Momentum = 0.1



Accuracy for Momentum = 0.5



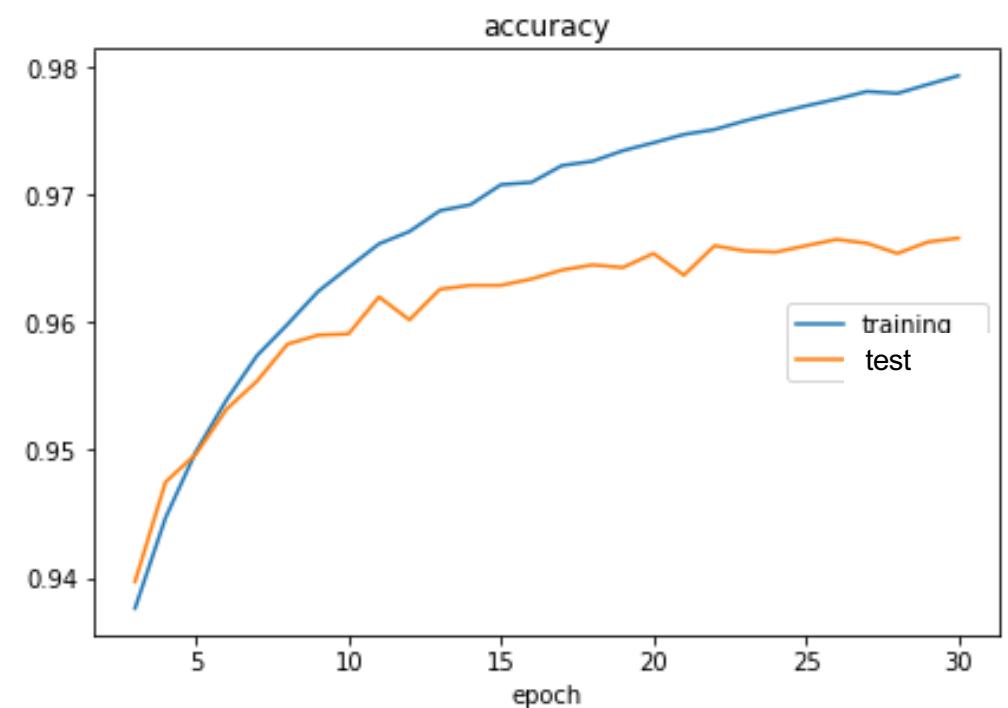
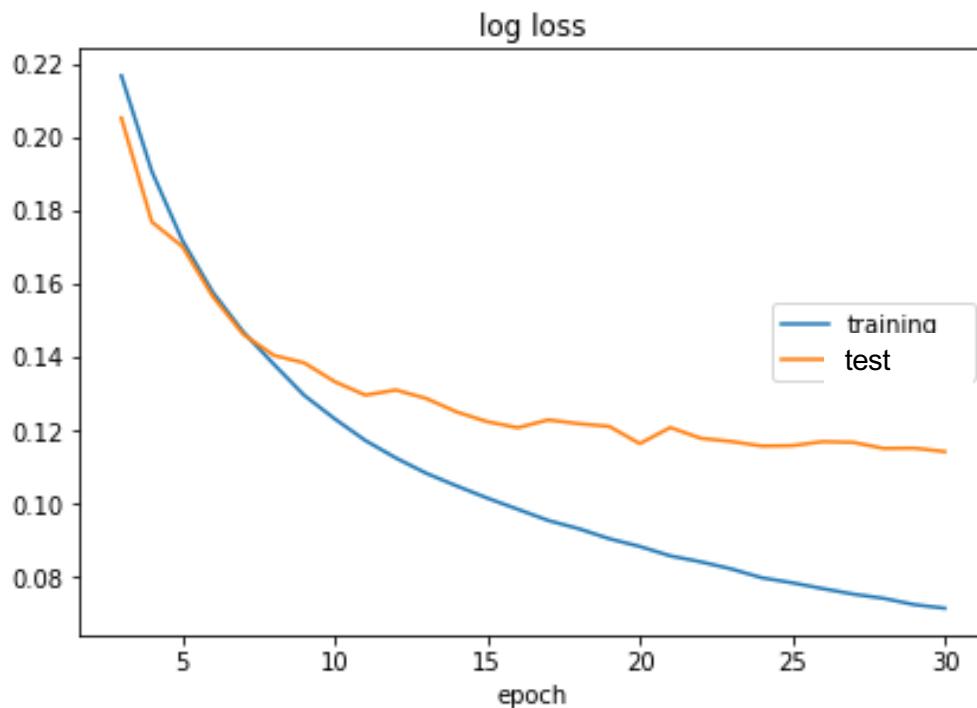
Accuracy for Momentum = 0.9



*Slightly Better Best Performance on Validation Set
(all three Validation Accuracies close to .96)*

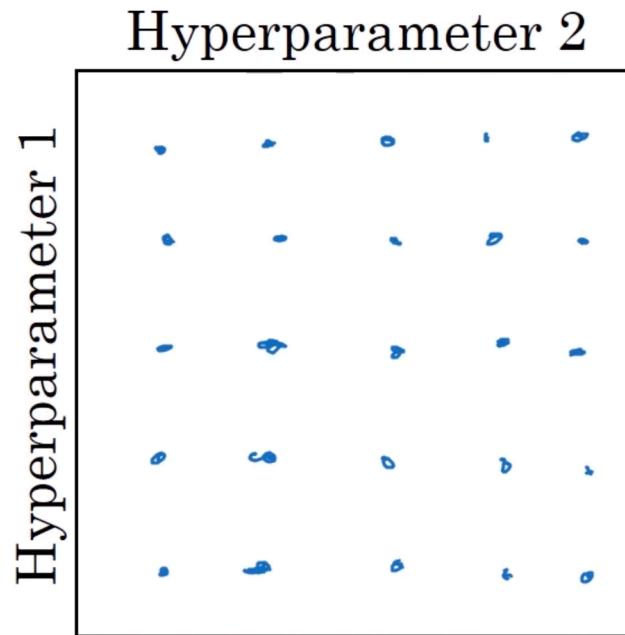
“Grid Search” evaluates Validation Accuracy for a set of Hyperparameter(s) values

MNIST: Final Training using Training & Validation Set



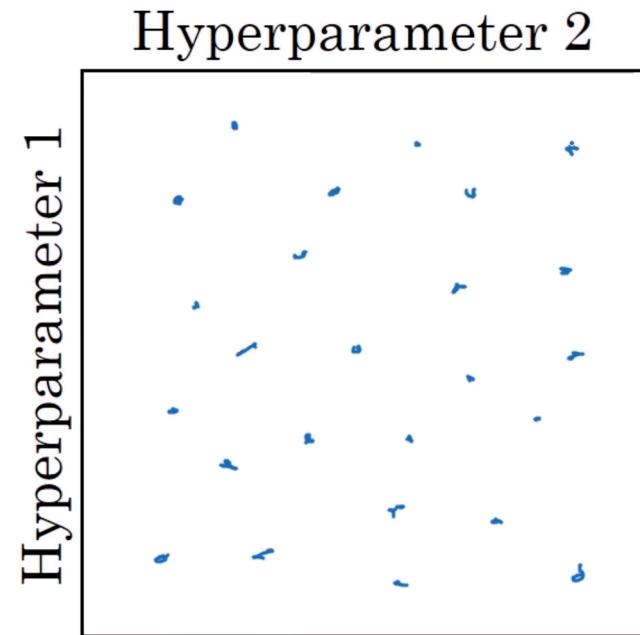
Test Set accuracy at about .97!

Grid Search to Optimize Multiple Hyperparameters



Systematic Grid Search

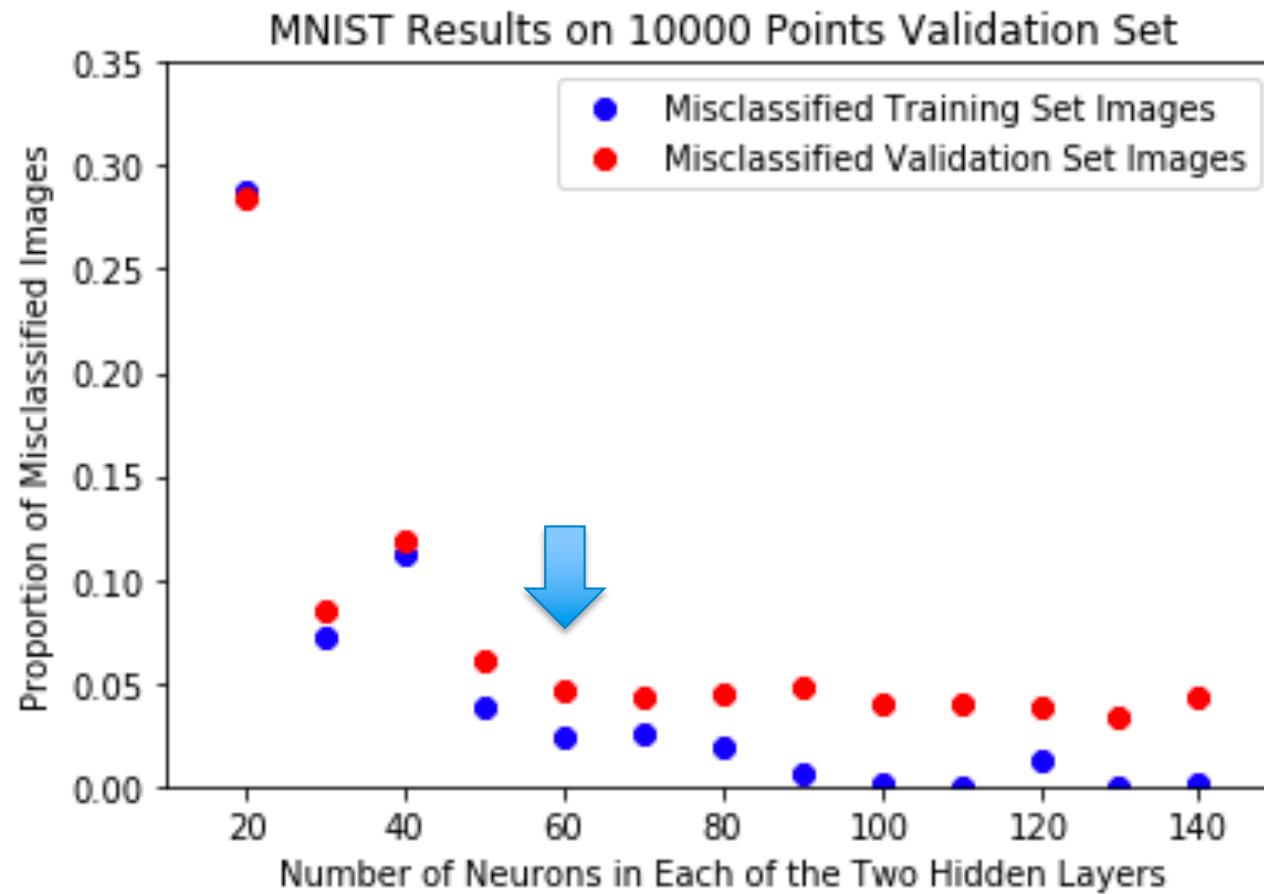
Map the Validation Set Accuracy for each Hyperparameter pair.



Random Grid Search

If one parameter is less sensitive than the other, the random search provides a richer exploration of the values of the sensitive parameter.

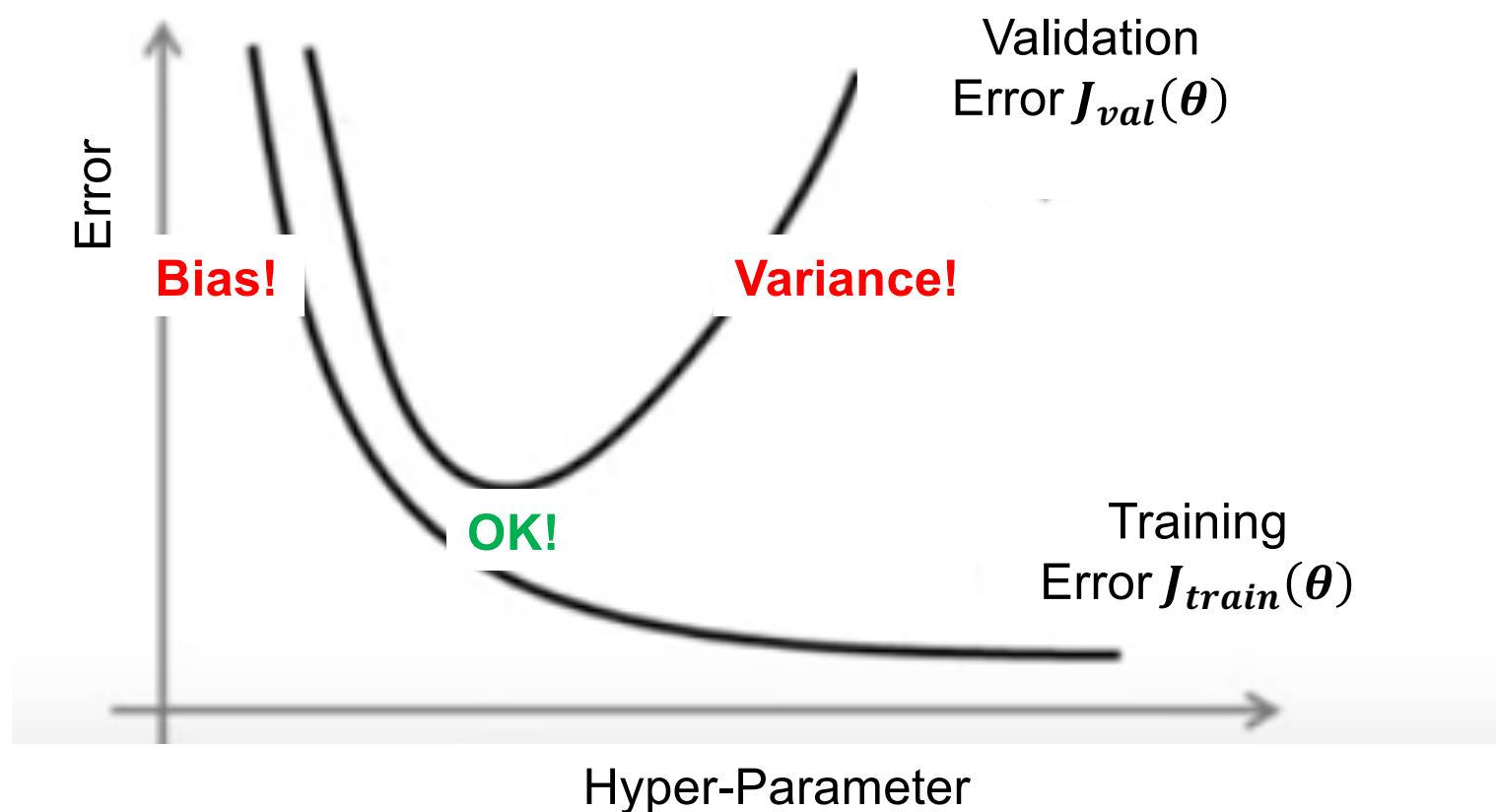
MNIST: Optimizing Number of Neurons in Layers



Optimal value seems to be about 60 neurons in each of the two hidden layers.

This hyperparameter value gives a proportion of 0.04 misclassified images in the Test Set (same as Validation Set!)

How to Identify Bias vs Variance Problems



Bias/Underfitting and Variance/Overfitting in NNs

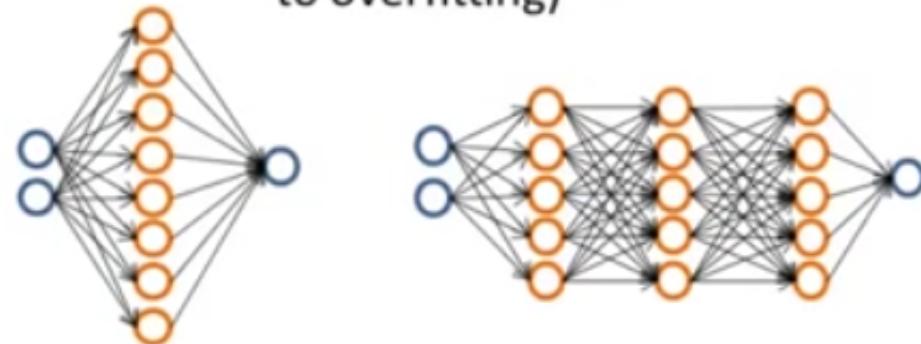
Neural networks and overfitting

“Small” neural network
(fewer parameters; more
prone to underfitting)



Computationally cheaper

“Large” neural network
(more parameters; more prone
to overfitting)



Computationally more expensive.

Use regularization (λ) to address overfitting.

General Guidelines for Improving Training

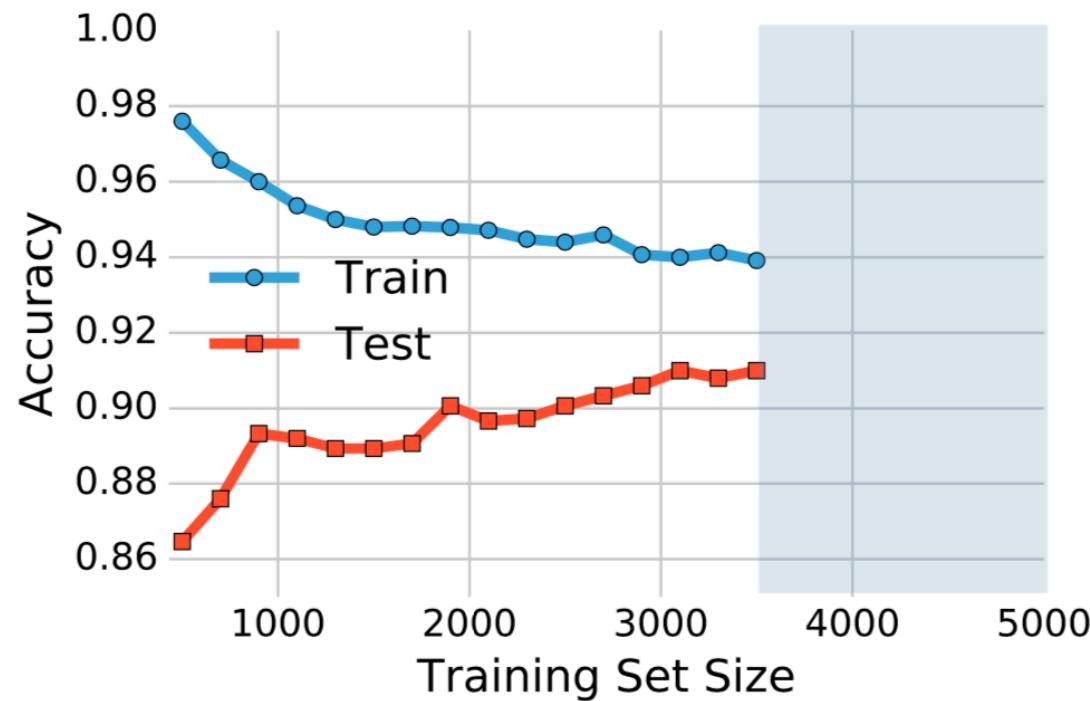
To address Bias problems

- Try using more input features in order to increase the number of parameters
- Try decreasing the regularization

To address Variance problems

- Try decreasing the number of features
- Try increasing the regularization

MNIST : Impact of Number of Training Data



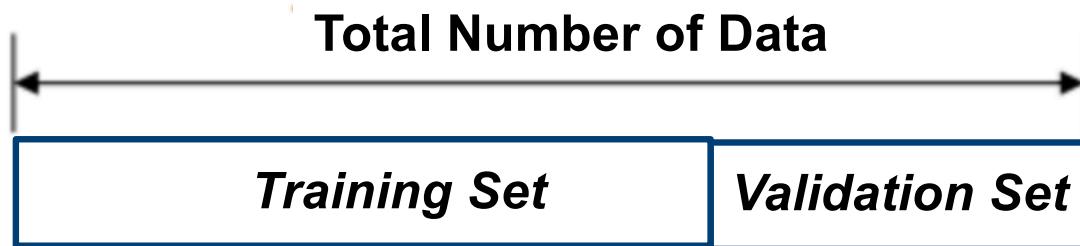
The larger the number of Training Data, the less chance of Overfitting!

Regularization, Bias and Variance

- 1. Overfitting and Underfitting, Bias and Variance**
 - 2. Regularization**
 - 3. Batch Normalization**
 - 4. The Need for Machine Learning Diagnostics**
 - 5. Training Set, Validation Test and Test Set**
 - 6. K-Fold Validation**
-

Limitation of using just one Validation Set (Hold-Out Method)

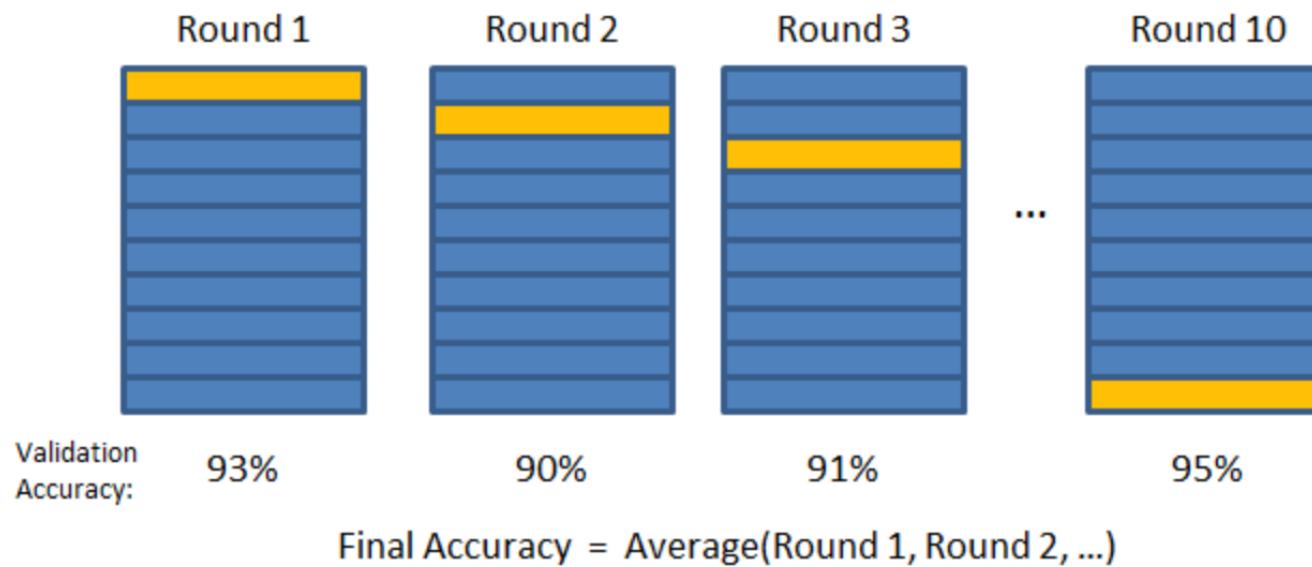
If Data Set is not very big, the role played by Training and Validation sets is not symmetrical. The approach used for the split may significantly affect the results.



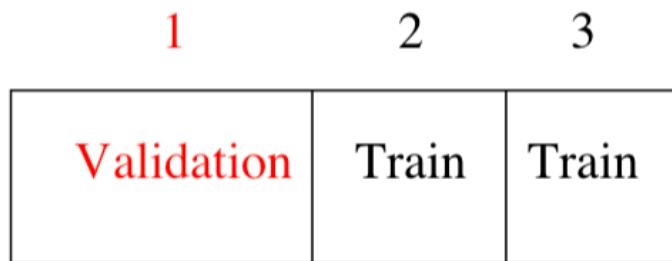
Possible approach: permute between Validation and Training Sets and recalculate. Can do it if size of both is similar.

What is k-Fold Validation?

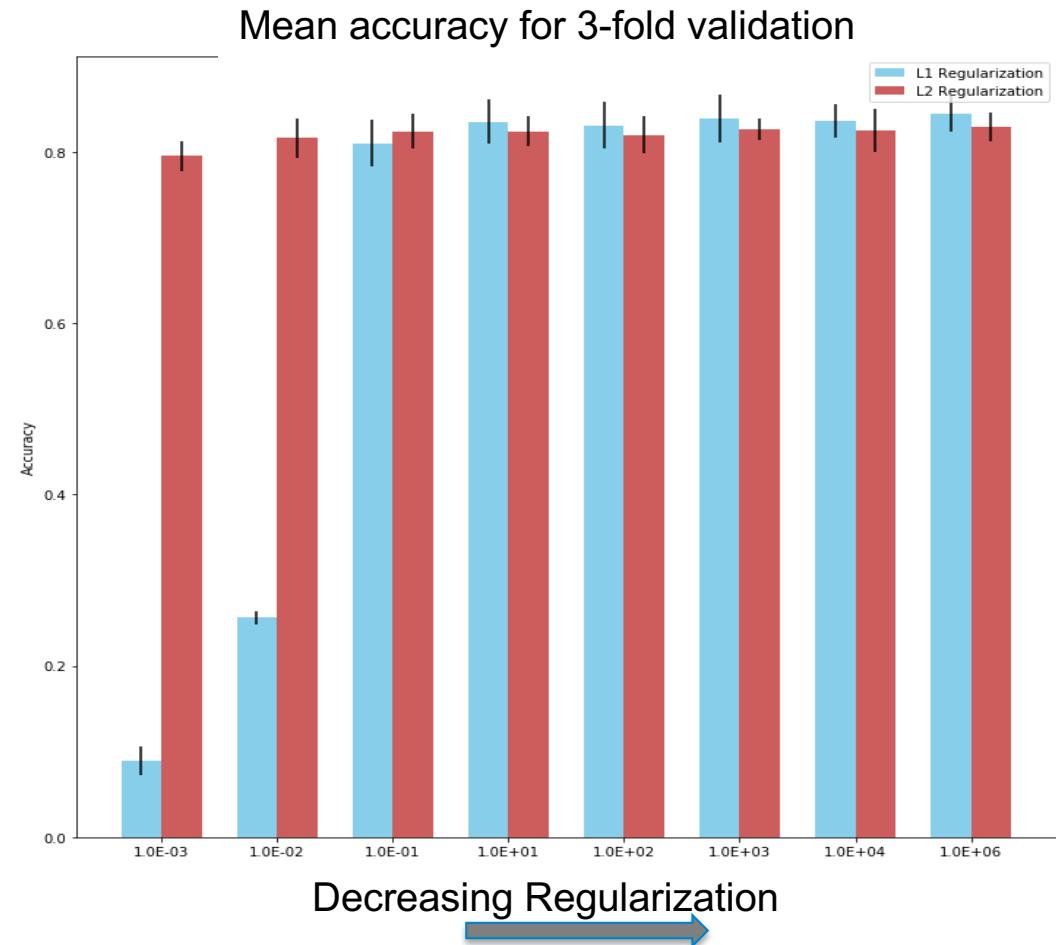
 Validation Set
 Training Set



3-Fold Validation on MNIST Example



For each value of the Regularization Parameter, a 3-fold validation is run: three validation sets are created in turn and predicted using the rest of the data as training set.



Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning

Sebastian Raschka

University of Wisconsin–Madison

Department of Statistics

November 2018

sraschka@wisc.edu

Afternoon Exercise

Abstract

The correct use of model evaluation, model selection, and algorithm selection techniques is vital in academic machine learning research as well as in many industrial settings. This article reviews different techniques that can be used for each of these three subtasks and discusses the main advantages and disadvantages of each technique with references to theoretical and empirical studies. Further, recommendations are given to encourage best yet feasible practices in research and applications of machine learning. Common methods such as the holdout method for model evaluation and selection are covered, which are not recommended when working with small datasets. Different flavors of the bootstrap technique are introduced for estimating the uncertainty of performance estimates, as an alternative to confidence intervals via normal approximation if bootstrapping is computationally feasible. Common cross-validation techniques such as leave-one-out cross-validation and k -fold cross-validation are reviewed, the bias-variance trade-off for choosing k is discussed, and practical tips for the optimal choice of k are given based on empirical evidence. Different statistical tests for algorithm comparisons are presented, and strategies for dealing with multiple comparisons such as omnibus tests and multiple-comparison corrections are discussed. Finally, alternative methods for algorithm selection, such as the combined F -test 5x2 cross-validation and nested cross-validation, are recommended for comparing machine learning algorithms when datasets are small.

Summary for Regularization, Bias and Variance

- **Bias and Variance to characterize Underfitting versus Overfitting.**
- **L1 or L2 Regularizations as a cure against Overfitting.**
- **Drop-out and Data Augmentation are other techniques for Regularization.**
- **Batch Normalization for accelerating Training (and Regularization)**
- **Importance of Training, Validation and Test Set for Diagnostics.**
- **K-Fold Validation for Hyper-Parameters Optimization.**