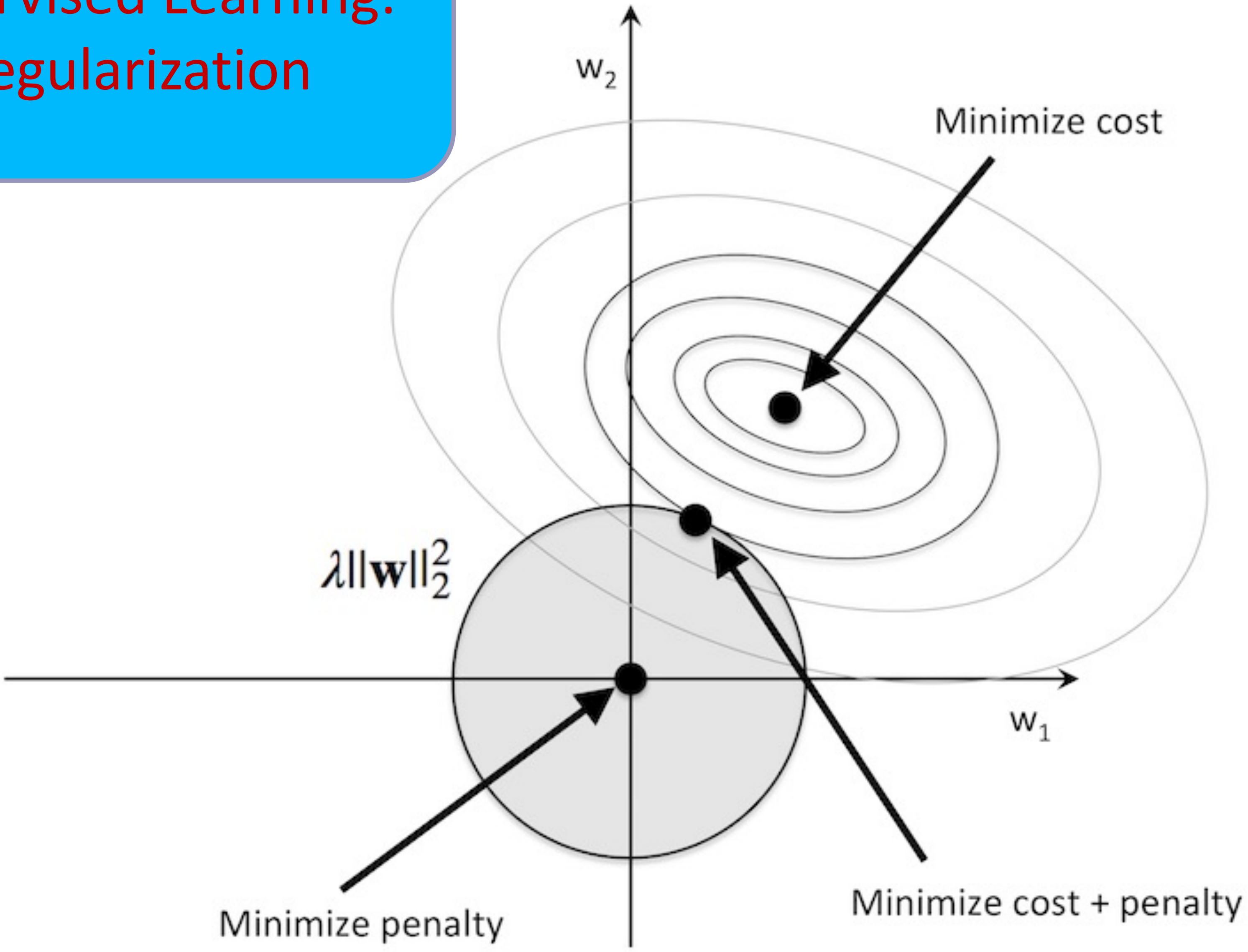
Supervised Learning: Regularization



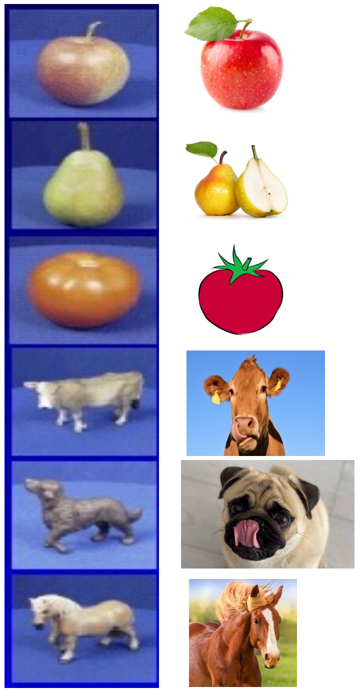
Roadmap

- Generalization Concepts
  - Bias-Variance Tradeoff
- Regularization
  - Overfitting and Generalization
  - Regularization Tricks
    - Dropout, Data Augmentation and Early Stopping
- More about Generalization Error
  - Cross Validation and Validation Set
- Take-home messages

Generalization

Slide credit: L. Lazebnik

Training set (labels known)


Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

Generalization Error

- **"True" distribution:** P(x,y)
  - Unknown to us

- **Train:** h(x) = y
  - Using training data $S = \{(x_1,y_1),\ldots,(x_n,y_n)\}$
  - Sampled from probability distribution P(x,y), i.e., a "snapshot" of the true distribution

i.e., including the unseen data here!

Wiki: Expected value

- **Generalization Error:**
  - $\mathcal{L}(h) = E_{(x,y)\sim P(x,y)}[\ f(h(x),y)\ ]$
  - E.g., squared error $f(a,b) = (a-b)^2$

True Data: Unknown to us

Person	Age	Male?	Height > 55”
James	11	1	1
Jessica	14	0	1
Alice	14	0	1
Amy	12	0	1
Bob	10	1	1
Xavier	9	1	0
Cathy	9	0	1
Carol	13	0	1
Eugene	13	1	0
Rafael	12	1	1
Dave	8	1	0
Peter	9	1	0
Henry	13	1	0
Erin	11	0	0
Rose	7	0	0
Iain	8	1	1
Paulo	12	1	0
Margaret	10	0	1
Frank	9	1	1
Jill	13	0	0
Leon	10	1	0
Sarah	12	0	0
Gena	8	0	0
Patrick	5	1	1

⋮

Training Data: Known to us

Person	Age	Male?	Height > 55”	
Alice	14	0	1	✓
Bob	10	1	1	✓
Carol	13	0	1	✓
Dave	8	1	0	✓
Erin	11	0	0	✗
Frank	9	1	1	✗
Gena	8	0	0	✓

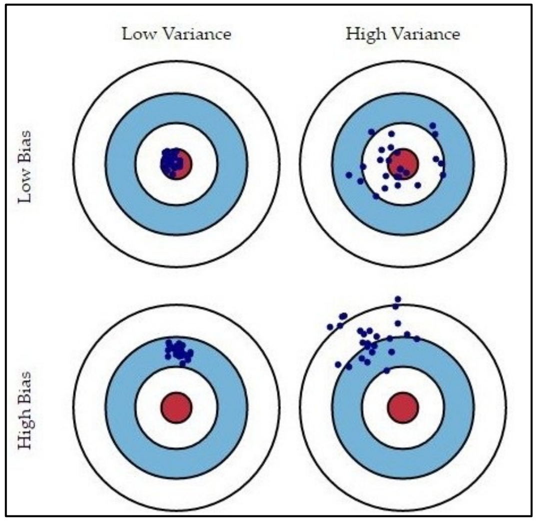
y h(x)

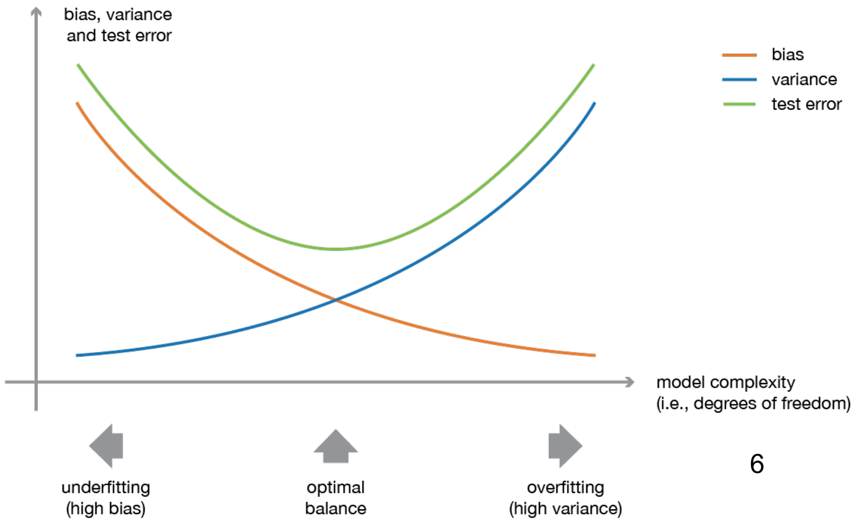
Generalization Error:

$$\mathcal{L}(h) = E_{(x,y)\sim P(x,y)}[\, f(h(x),y)\,]$$

Bias, Variance and their Trade-off

- The bias is the error which results from missing a target.
- For example, if an estimated mean is 3, but the actual population value is 3.5, then the bias value is 0.5.
- The variance is the error which results from random noise.
- When the variance of a model is high, this model is considered unstable.
- A complicated model tends to have low bias but high variance.
- A simple model is more likely to have a higher bias and a lower variance.

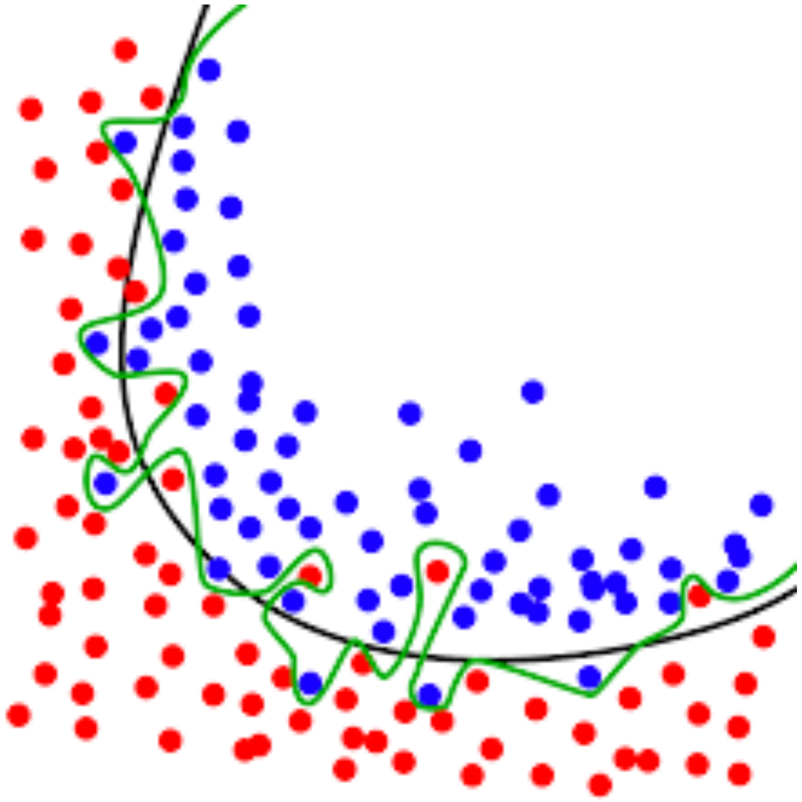


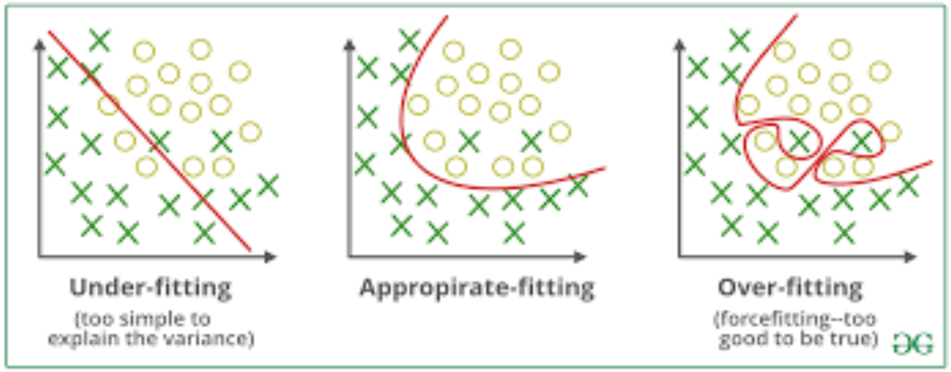


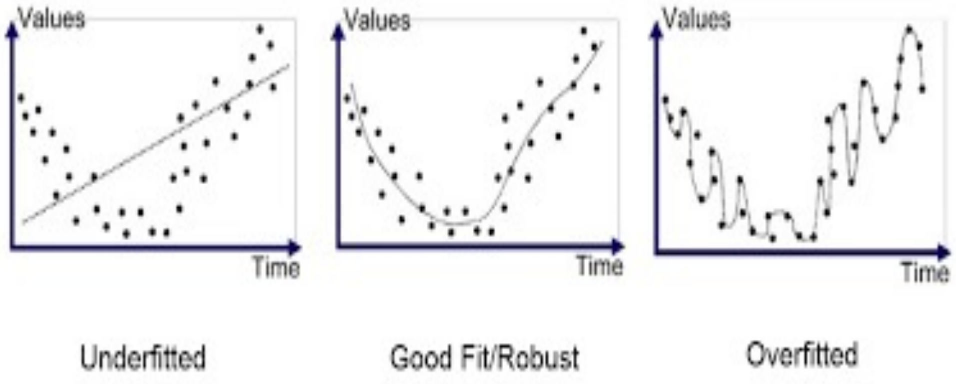
Bias-Variance Tradeoff

- Used to analyze how much selection of any classifier/training set affects performance. For any learning scheme,
  - Bias = expected error of the classifier on new data (high bias refers to underfit)
  - Variance = expected error due to the particular training set used (high variance refers to overfit)
- Total expected error ≅ bias + variance
- Underfitting: Model is too “simple” to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- Overfitting: Model is too “complex” and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

Problem of Overfitting



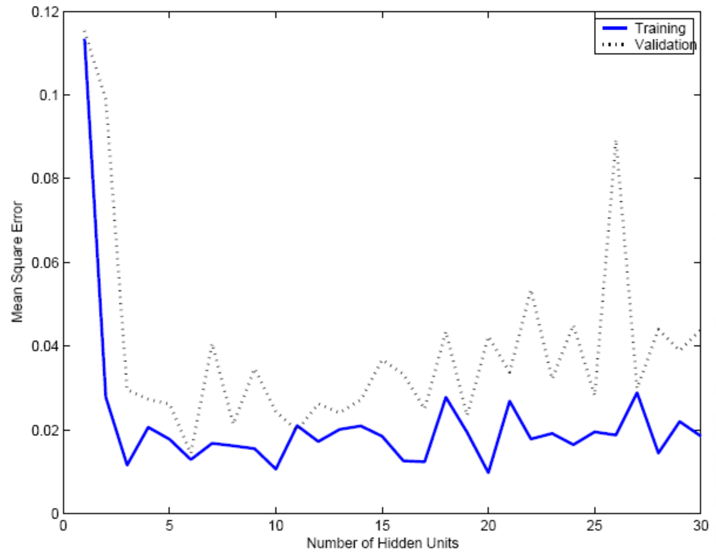


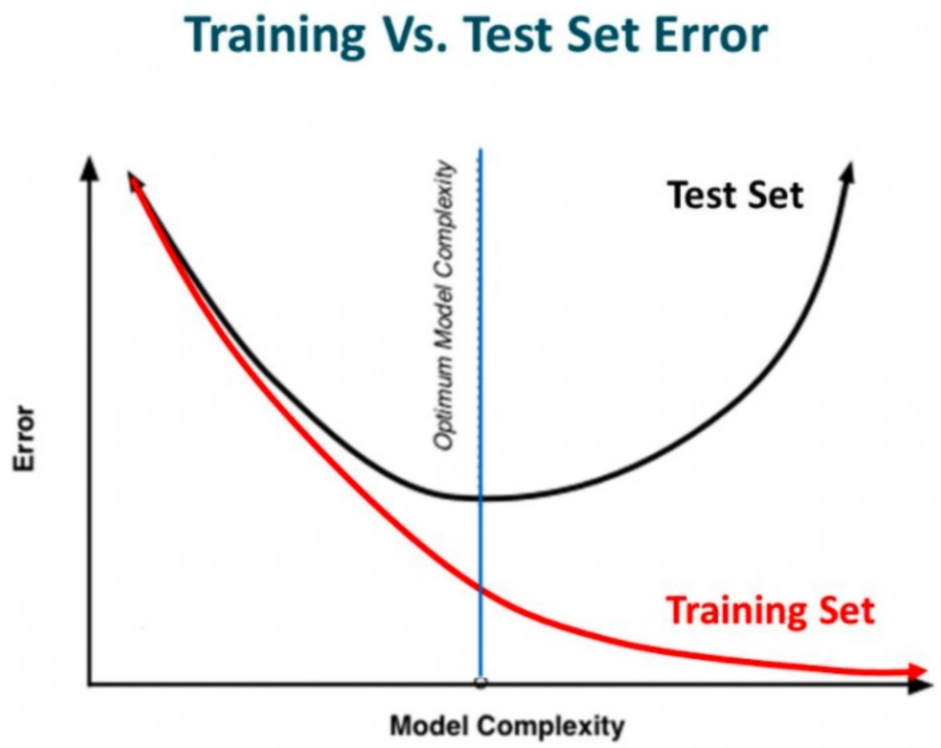


So, what's our main concern?
- Better fit? Then, how good is it?

Overfitting/overtraining vs Generalization

- **Generalization** refers to your model's ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model.





How to avoid overfitting and generalize better? Regularization

- A technique that discourages complex models, typically through constrained optimization.
  - E.g. optimize $E(\mathbf{W},\mathbf{v} \mid \mathcal{X}) = \frac{1}{2}\sum_t \left(r^t - y^t\right)^2$ subject to "smallest network" or "simplest model"
- Occam's razor
  - "Entities are not to be multiplied without necessity"
  - Prefers simpler functions over more complex one
  - So, add a regularization term to the original error/loss function *E*

$$Cost\ function = Loss + Regularization\ term$$

$$Cost\ function = E(w|X) + \lambda \sum \|w\|^2$$

encourage small weights!

Regularization parameter – requires finetuning!

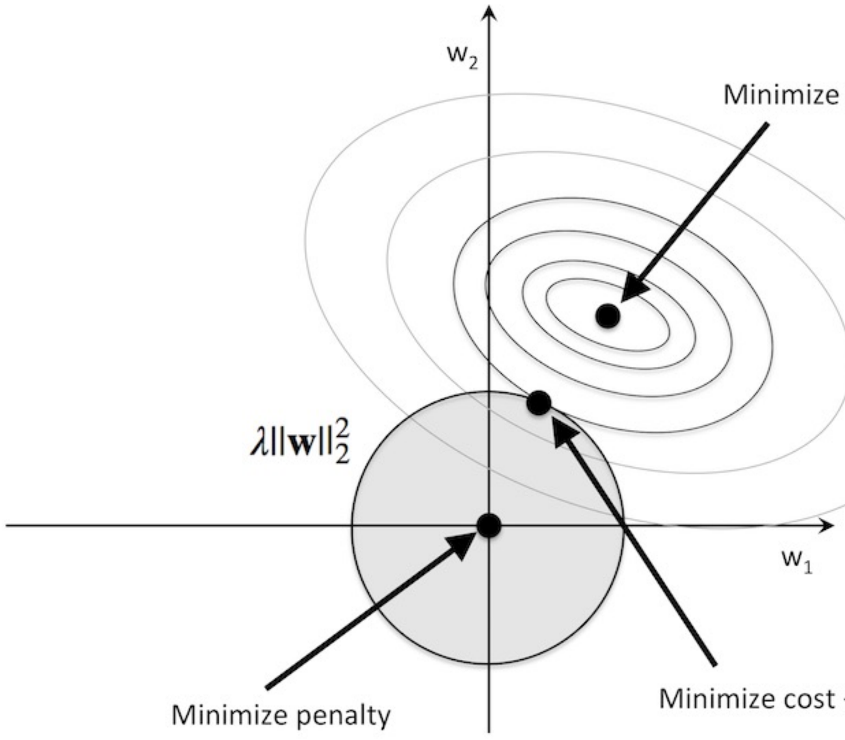
Regularization (cont.)

- *Modified Optimization Function:*

$$Cost\ function = E(w|X) + \lambda \sum \|w\|^2$$

encourage small weights!

Regularization parameter – requires finetuning!



(when λ =0)

(for a specific λ)

(when λ is very large)

Common regularizers with p-norm

L_1 (1-norm): sum of the absolute weights

i.e. sum of the absolute values of the weight vector

$$\|w\|^1 = \sum_i |w_i|$$

L_2 (2-norm): sum of the squared weights

i.e. square root of the sum of the squared vector values

$$\|w\|^2 = \sqrt{\sum_i |w_i|^2}$$

Squared weights penalize large values more
Sum of abolute weights will penalize small values more

L_p (p-norm):

$$\|w\|^p = \sqrt[p]{\sum_i |w_i|^p}$$

Smaller values of p (p < 2) encourage sparser vectors
Larger values of p discourage large weights more

Regularization in CNN

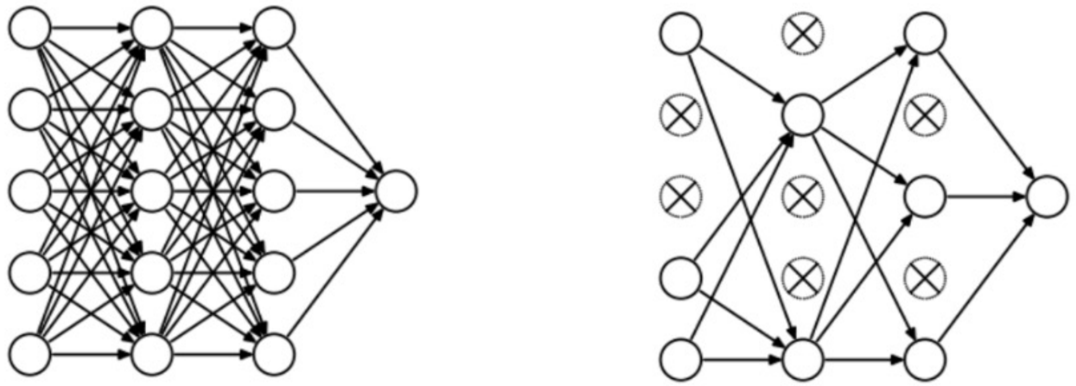
Some words about the design choices of neural networks

- Number of layers.
  - Too many: slow to train, risk of overfitting.
  - Too few: may be too simple to learn an accurate model.
- Number of units per layer.
  - We have a separate choice for each layer.
  - Too many: slow to train, dangers of overfitting.
  - Too few: may be too simple to learn an accurate model.
  - We have no choice for input layer (number of units = number of dimensions of input vectors) and output layer (number of units = number of classes).
- Connectivity: what outputs are connected to what inputs?

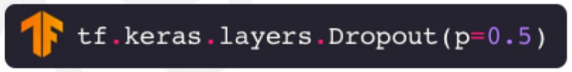
Regularization tricks in NN or CNN:

Dropout

- During training, randomly set some activations to 0 (randomly select some nodes and removes them along with their incoming and outgoing connections).



- Force the model not to rely on any node. So, each training iteration has a different set of nodes (different shared models), resulting to generalize better to unseen data.



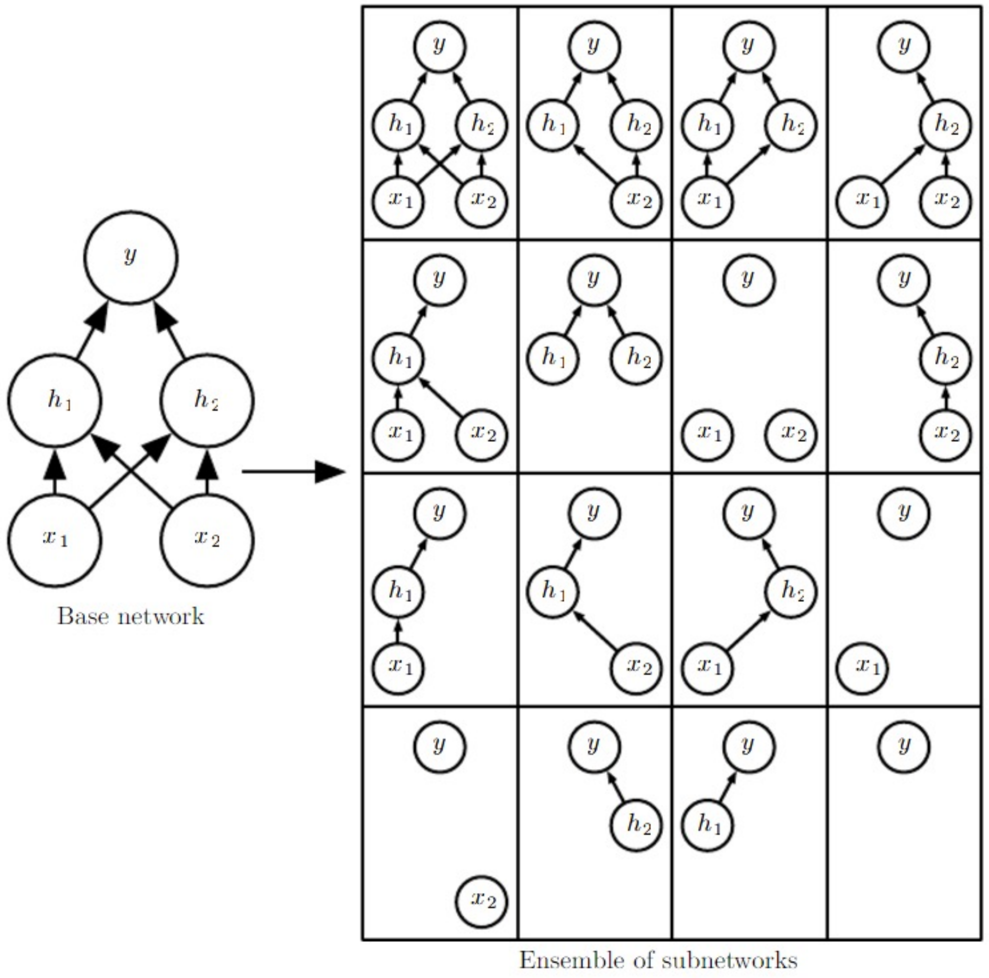
```
tf.keras.layers.Dropout(p=0.5)
```

- Parameter of dropout: probability *p* of training a given node in a layer. (Suggested p values: 0.5 for hidden nodes and 0.8 for input nodes, i.e., keeping more input nodes than hidden nodes)
- Hence, dropout is usually preferred in large NN training.

Dropout

- Dropout provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- It provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks (concept of ensemble methods).
- Specifically, dropout trains the (ensemble) NN model consisting of all sub-networks that can be formed by removing non-output units from an underlying base network.
- More practice tips can be found from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

An example of Dropout



Ways to drop 1 node: $\binom{4}{1}$=4

Ways to drop 2 nodes: $\binom{4}{2} = 6$

Ways to drop 3 nodes: $\binom{4}{3}$=4

Ways to drop 4 nodes: $\binom{4}{4}$=1

Training with Dropout

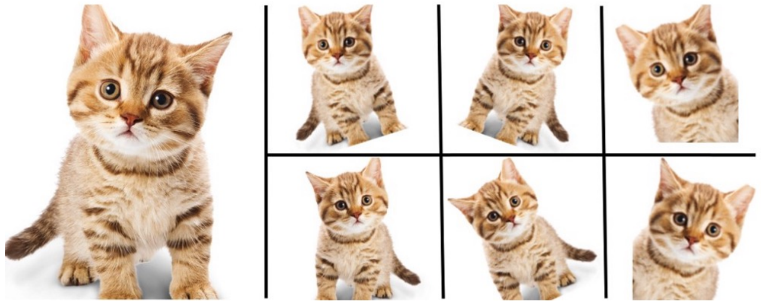
- To train with dropout, a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent (SGD), is used.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units (but NOT the output units) in the network.
- The mask for each unit is sampled independently from all of the others.
- Typically, the probability of including a hidden unit is 0.5, while the probability of including an input unit is 0.8.

Remarks on Dropout

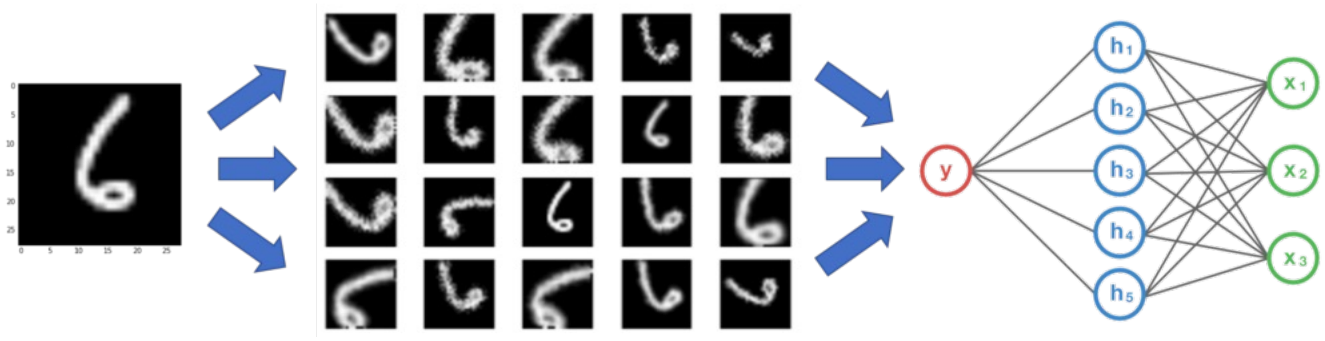
- Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.
- Dropout simulates a sparse activation from a given layer, which interestingly, in turn, encourages the network to actually learn a sparse representation (simpler model!) as a side-effect.
- Dropout is that it is very computationally cheap, using dropout during training requires only O(*n*) computation per example per update, to generate *n* random binary numbers and multiply them by the state. (*** Cost per step could be negligible but dropout assumes a wasteful larger network which implies more resources.)
- Dropout does not significantly limit the type of model or training procedure that can be used. It works well with nearly any model that uses a distributed representation and can be trained with stochastic gradient descent.
- Dropout can also be applied to other large learning networks, e.g. Autoencoder and LSTM (Long-Short Term Memory) Model.
- Some studies say that it is not intuitive to apply it to convolutional layers.

Regularization tricks in NN or CNN:
Data Augmentation

- One way to address overfitting is to increase the training data size – data augmentation.



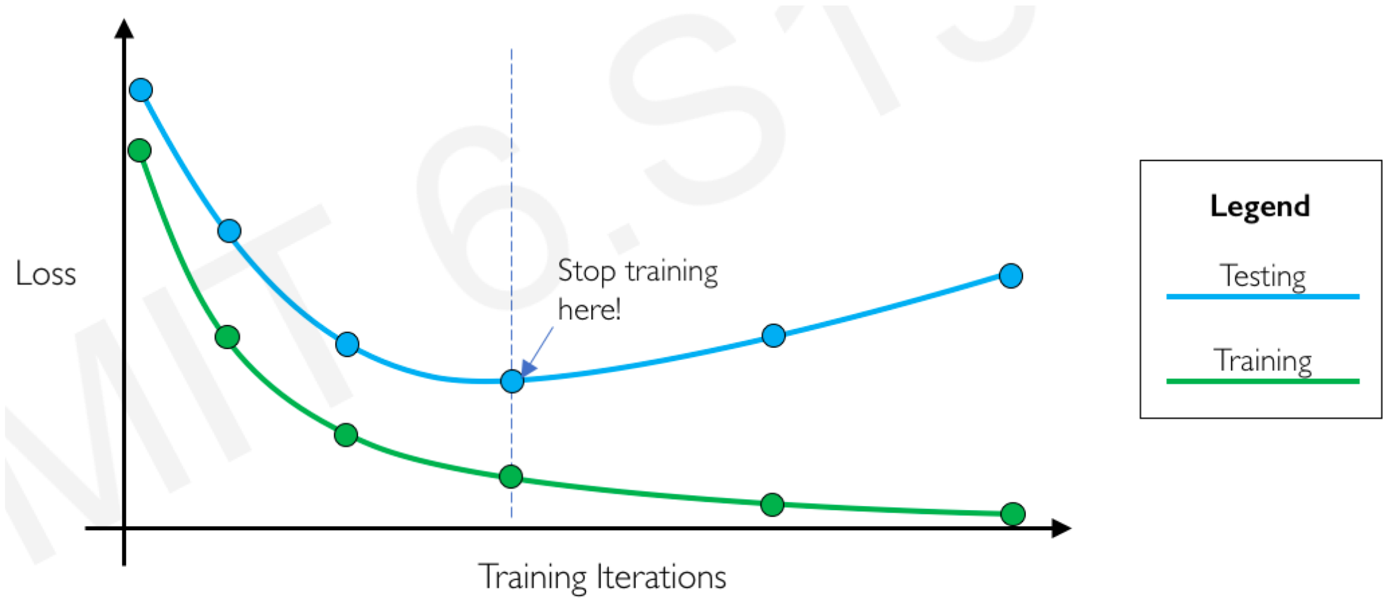
Enlarge your Dataset



- A highly effective technique! But not straightforward for some applications.

Regularization tricks in NN or CNN:
Early stopping

- Basic idea is to stop training before overfitting occurs.



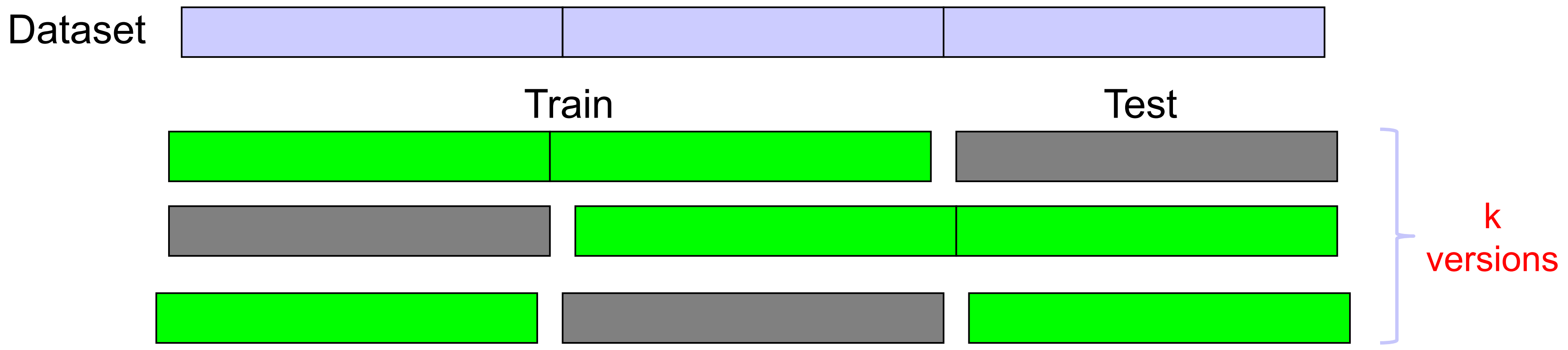
- Basically, it is a “how” question.
  - Keep track of the performance/loss difference between training and testing/validation
  - When such difference becomes ”larger”, stop and undo the overtraining work.

More about Generalization Error

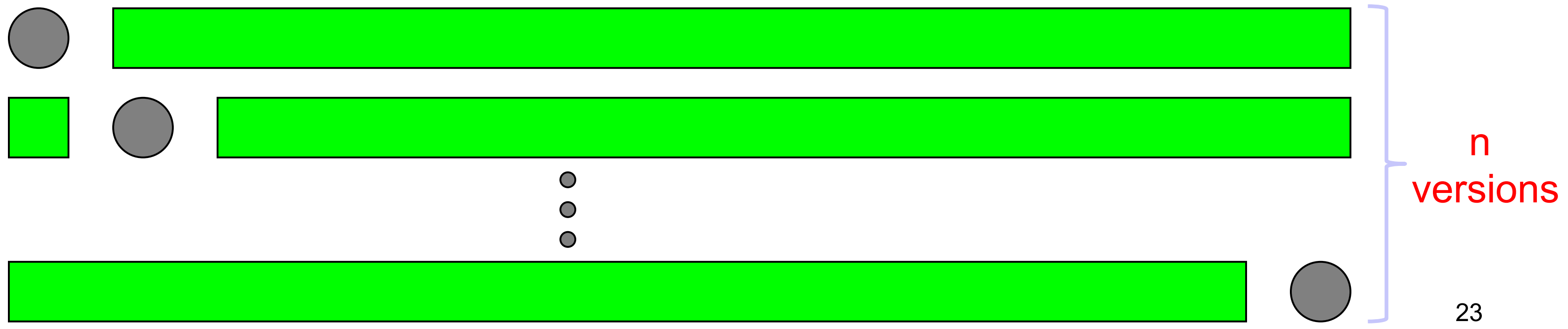
- No classifier is inherently better than any other: you need to make assumptions to generalize!
- Three kinds of error
  - Inherent: unavoidable
  - Bias: due to over-simplifications
  - Variance: due to inability to perfectly estimate parameters from limited data
- How to reduce variance?
  - Choose a simpler classifier (like regularizer)
  - Get more training data (like data augmentation)
  - Cross-validate the parameters

Cross Validation

- k-fold cross-validation

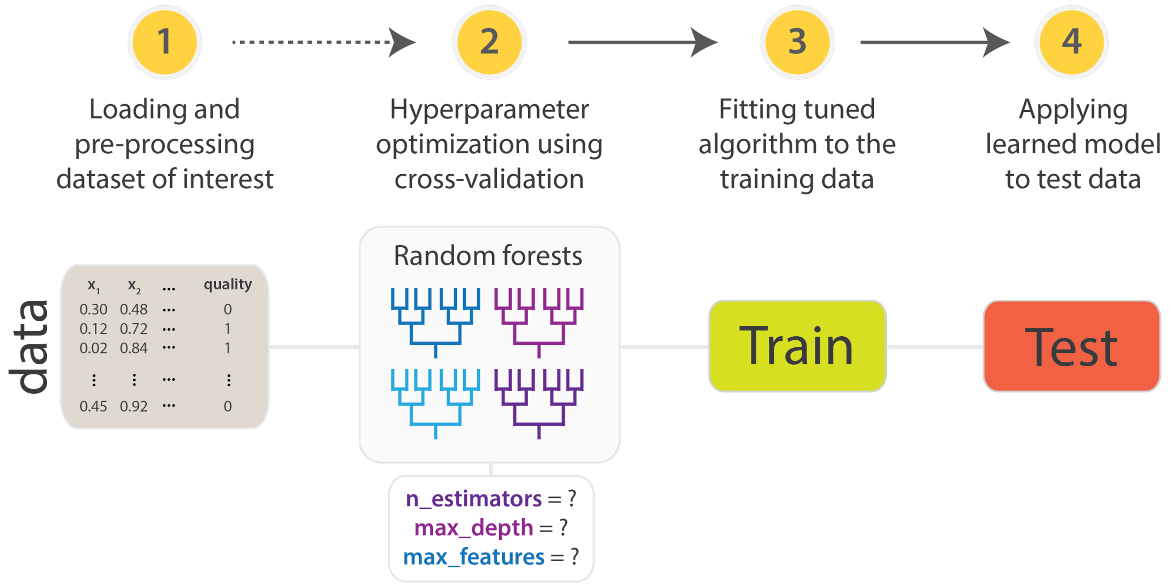


- Leave-one-out (n-fold) cross validation (LOOCV)



Training, Validation and Testing

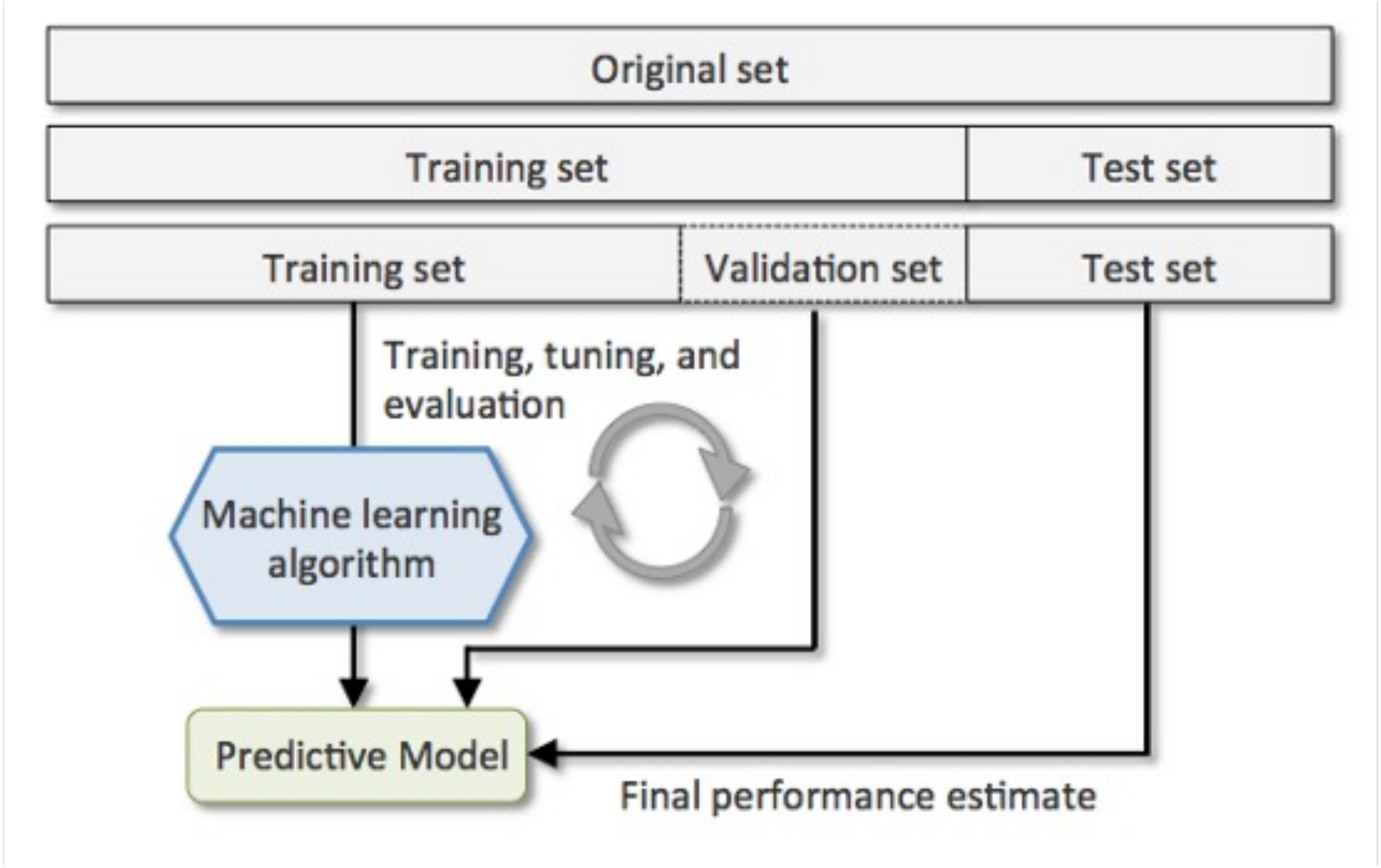
- Training set: Data to train up the model
- Testing set: Data to test the trained model (understand how the trained model predicts unseen data)
- Validation set: Data to help the model from overfitting and choose hyperparameters



1
Loading and pre-processing dataset of interest
2
Hyperparameter optimization using cross-validation
3
Fitting tuned algorithm to the training data
4
Applying learned model to test data
data
x1 x2 … quality
0.30 0.48 … 0
0.12 0.72 … 1
0.02 0.84 … 1
0.45 0.92 … 0
Random forests
Train
Test
n_estimators = ?
max_depth = ?
max_features = ?

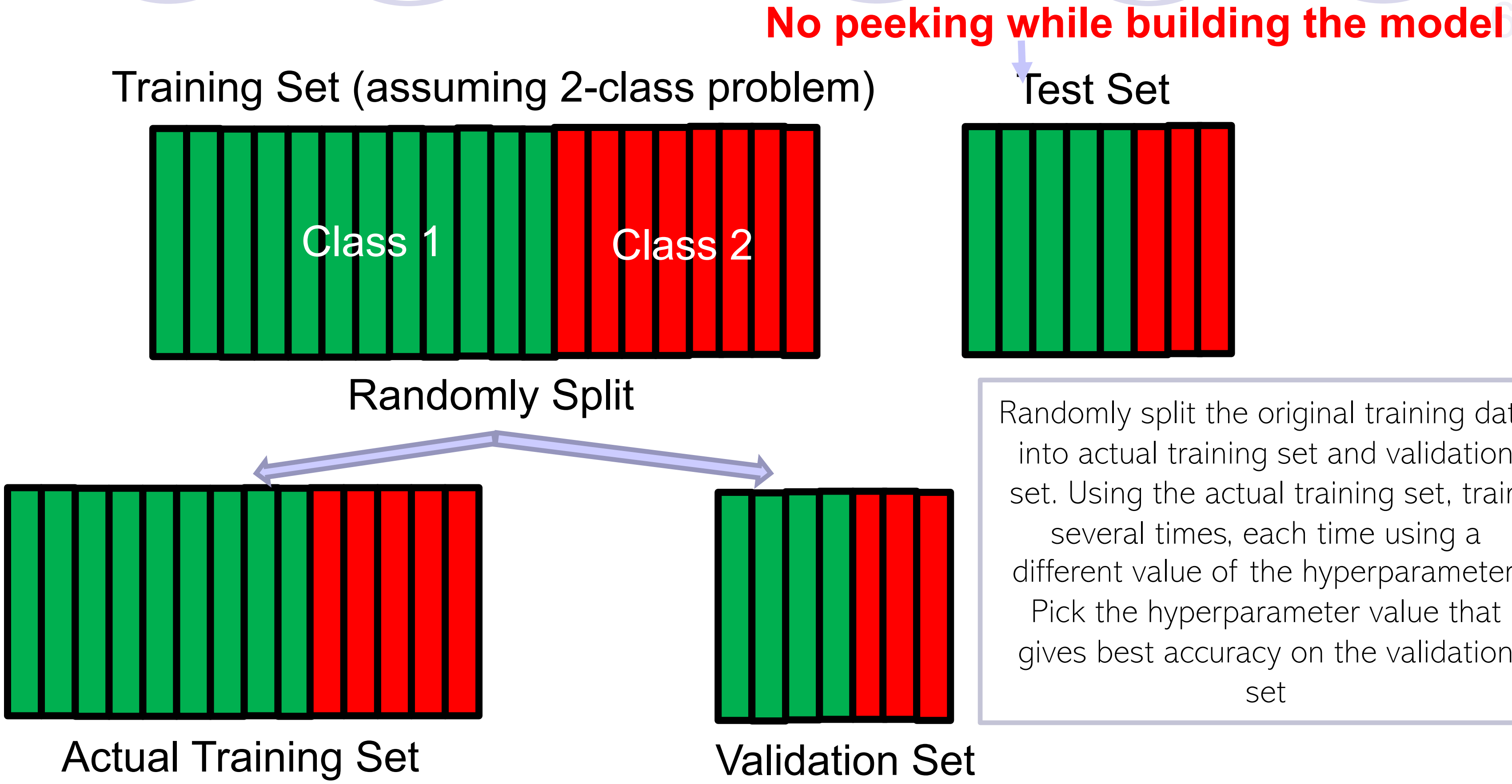
Training, Validation and Testing

- Note the iterative process involving the validation set



Original set
Training set
Test set
Training set
Validation set
Test set
Training, tuning, and evaluation
Machine learning algorithm
Predictive Model
Final performance estimate

Validation Set and k-fold CV


No peeking while building the model
Training Set (assuming 2-class problem)
Class 1
Class 2
Test Set
Randomly Split
Actual Training Set
Validation Set

Randomly split the original training data into actual training set and validation set. Using the actual training set, train several times, each time using a different value of the hyperparameter. Pick the hyperparameter value that gives best accuracy on the validation set

What if the random split is unlucky (i.e., validation data is not like test data)?

If you fear an unlucky split, try multiple splits. Pick the hyperparameter value that gives the best average CV accuracy across all such splits. If you are using k splits, this is called k–fold cross validation

Take-home Messages

- Goal of machine learning model – generalize better (better performance on unseen data)!
- Various concepts involved: generalization error, bias-variance, overfitting and underfitting, etc.
- Need the concept of regularization!
- Regularizer approach is fundamental in ML.
- Dropout is a very famous regularization trick for deep neural networks.
- To generalize better, there involve lots of trials and intelligent picks.

Acknowledgement

- Slides from following sources:
  - https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/
  - Ali Harakeh, U of Waterloo
- Images from Google search of Internet