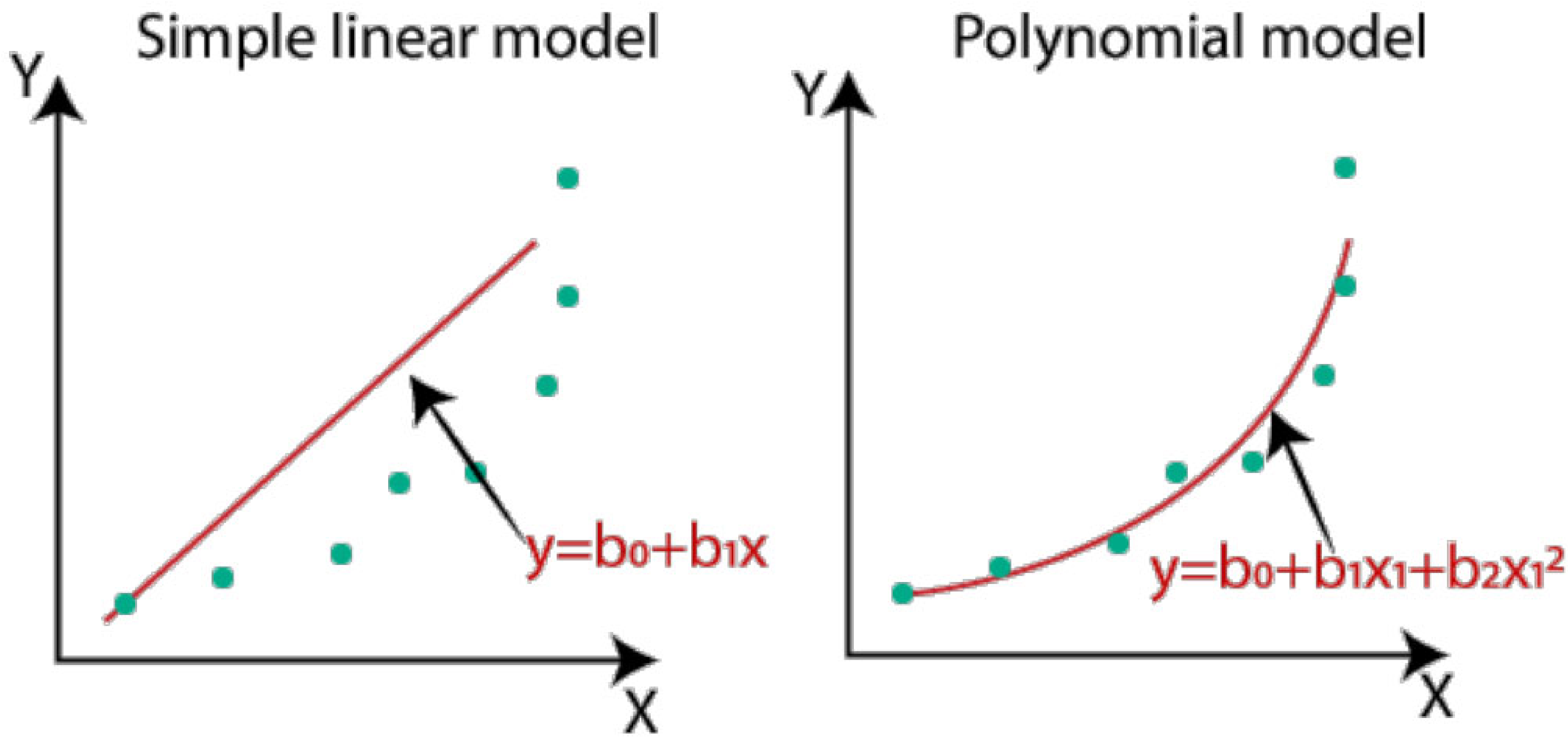
Deep Learning

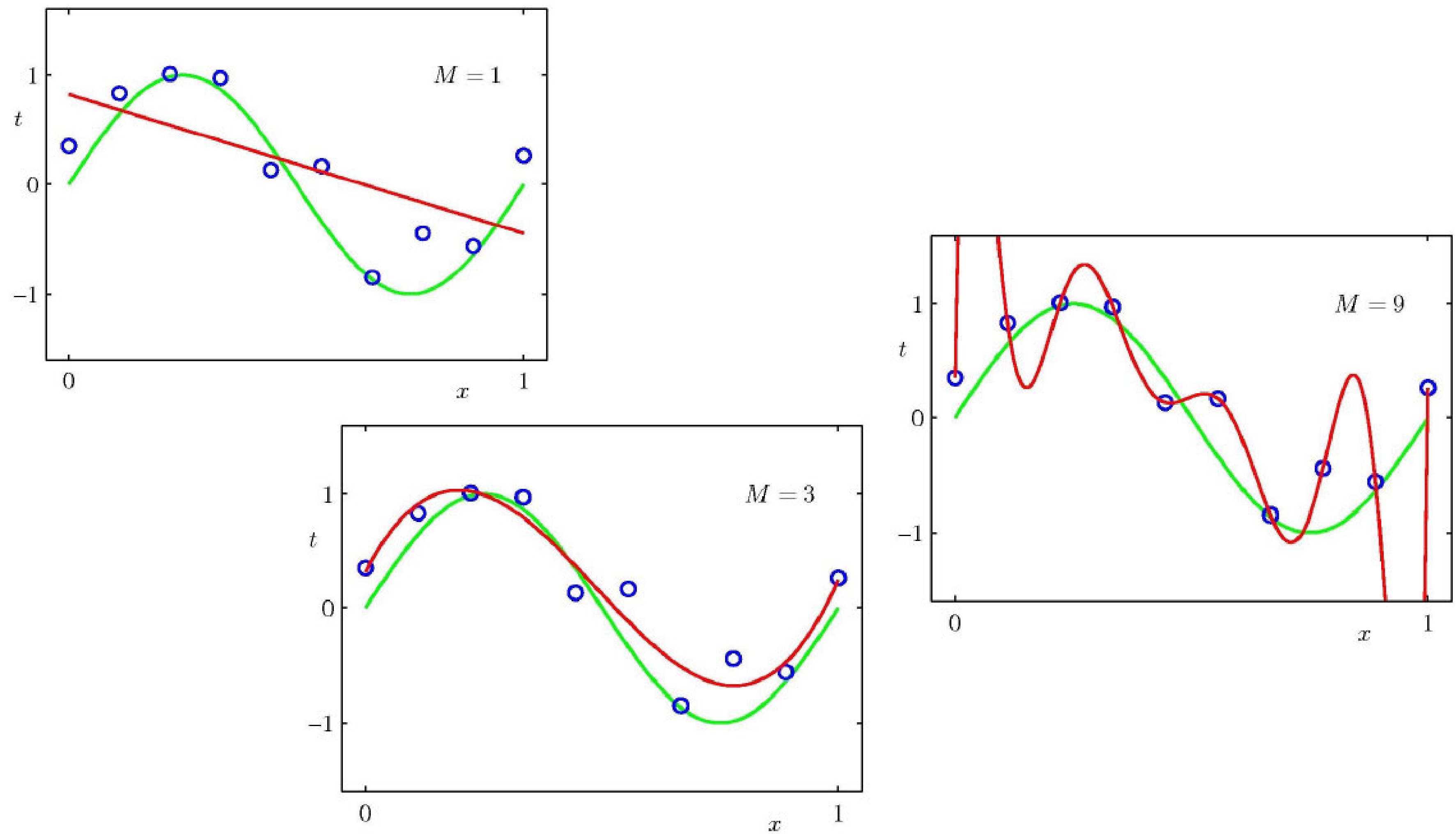
Winter 2026

Regularization

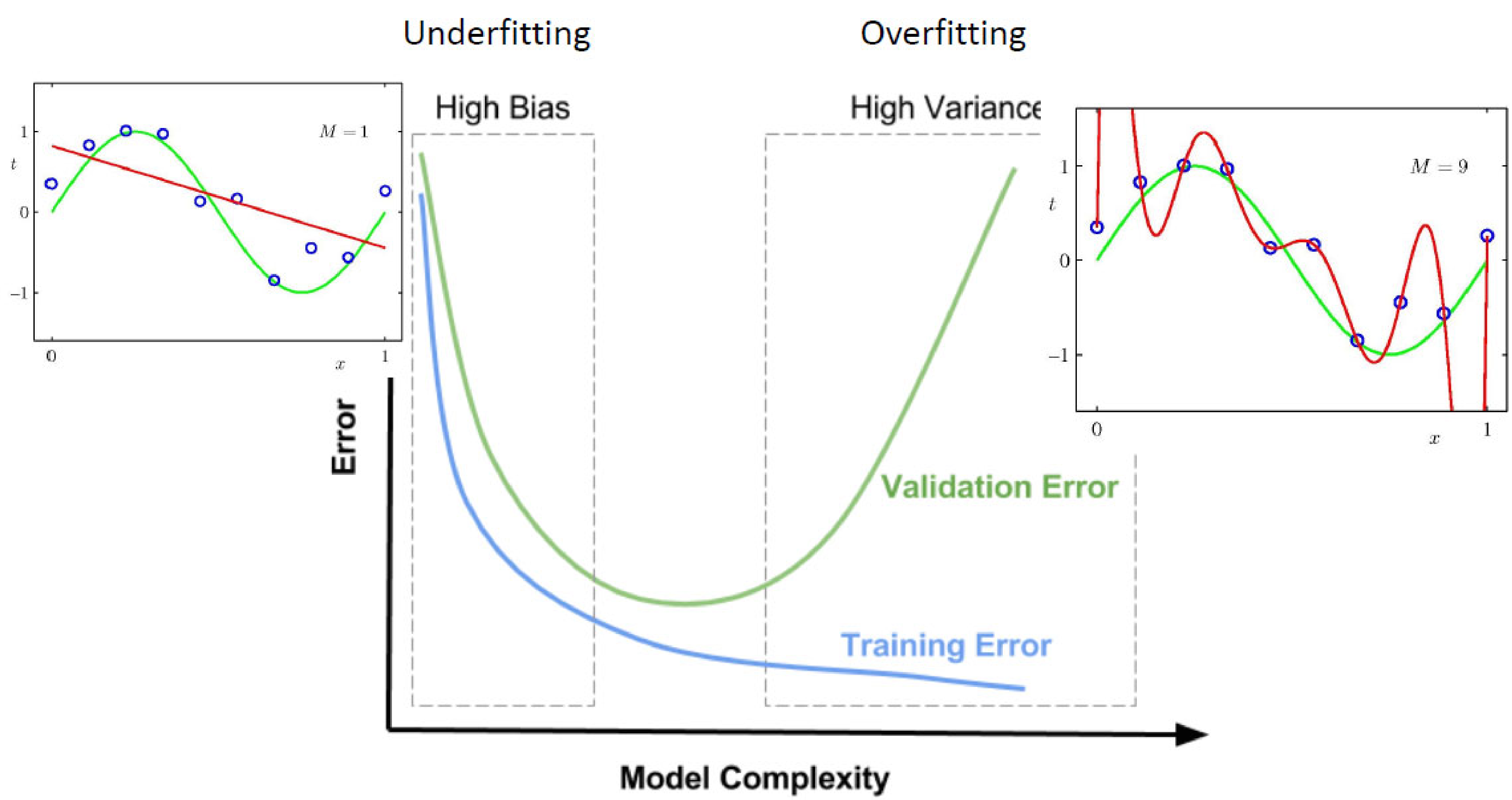
Polynomial Regression



Under/Over Fitting



Bias/Variance Trade-off

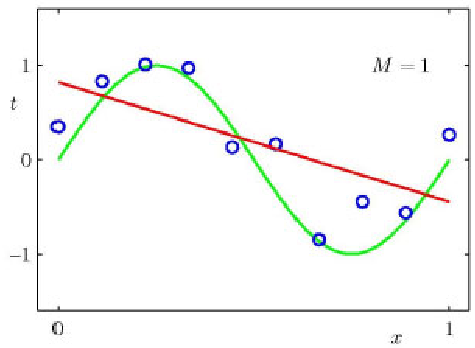


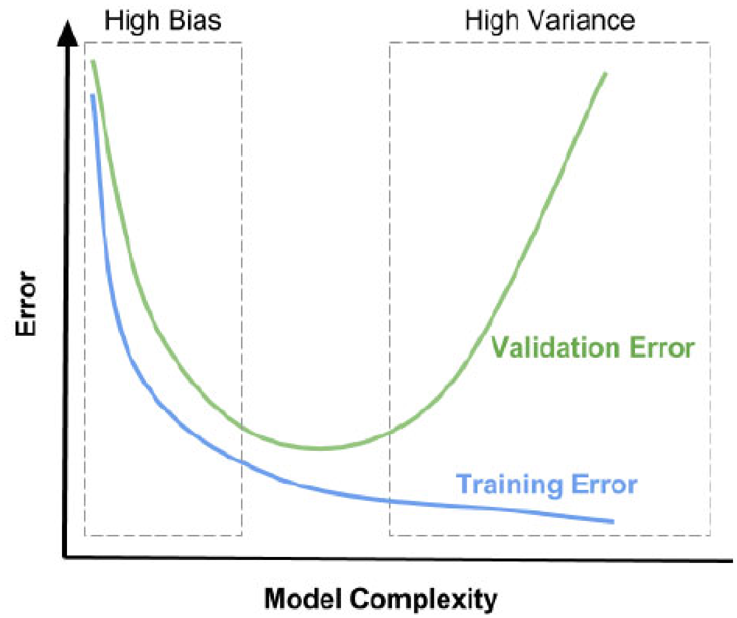
What Goes Wrong?

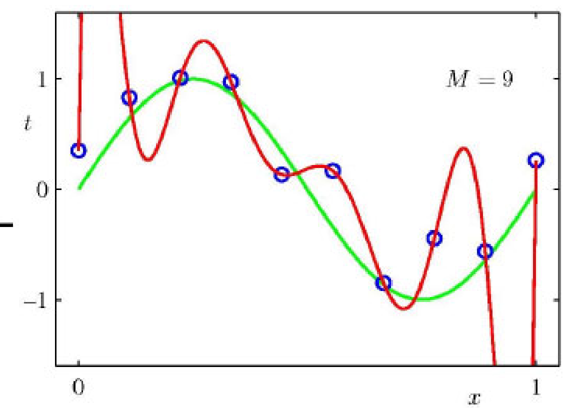
- Underfit: too many assumptions
  - Hypothesis class doesn't contain the optimal hypothesis
    - (or even a "good" hypothesis)
  - Data representation discards essential information

- Overfit: not enough assumptions
  - Hypothesis class is "hard to search"
  - Learning algorithm is inefficient, can't optimize parameters
  - Many hypotheses perfectly fit training data

How Do We Stop Overfitting



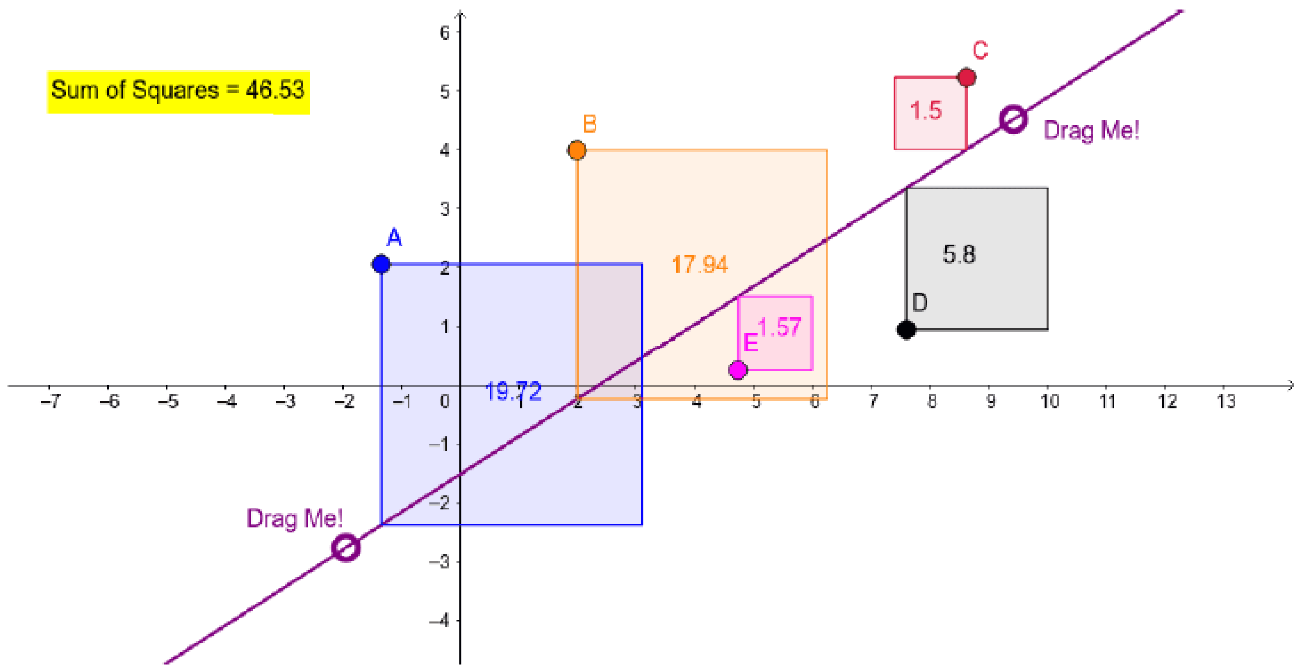




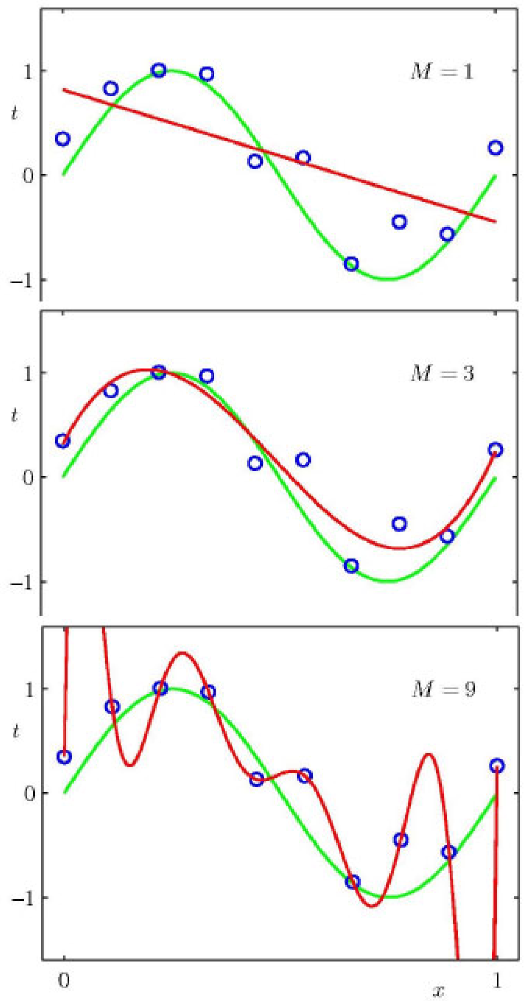
Motivating Regularization

- Our predictor f with parameters w has loss L:

$$L(y,\ f(x;\ w)) = (y - w \cdot x)^2$$



Motivating Regularization

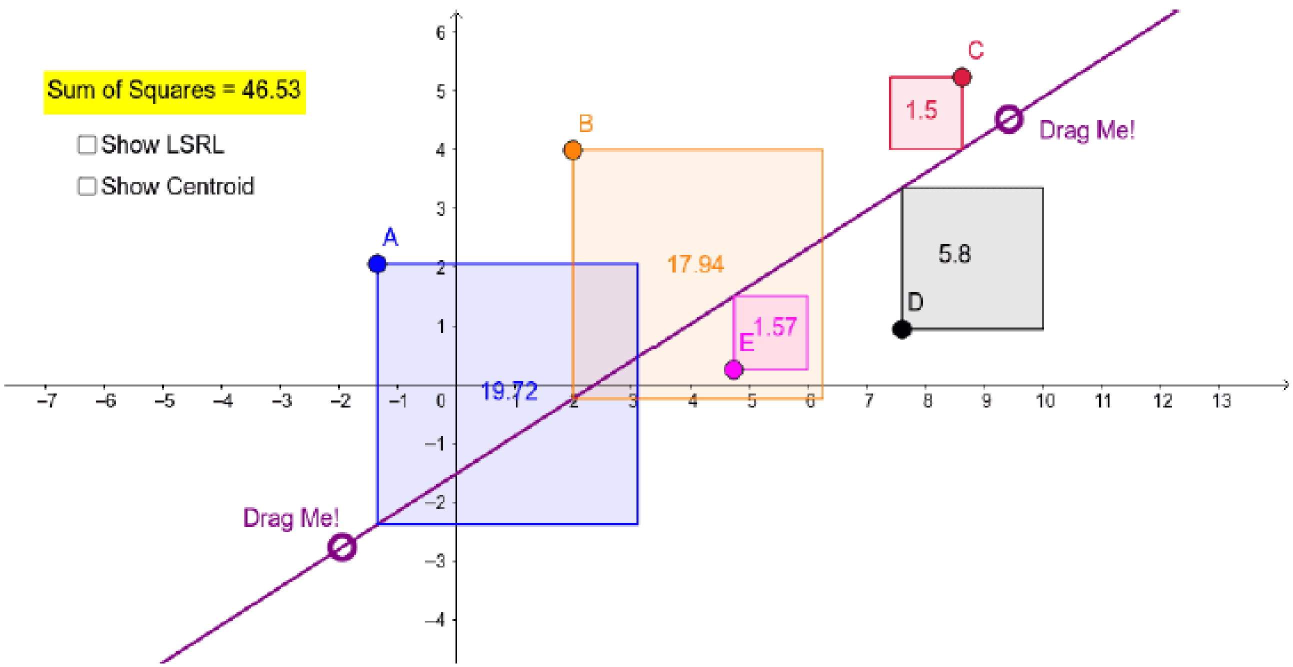


	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.82	0.31	0.35
$w_1^\star$	-1.27	7.99	232.37
$w_2^\star$		-25.43	-5321.83
$w_3^\star$		17.37	48568.31
$w_4^\star$			-231639.30
$w_5^\star$			640042.26
$w_6^\star$			-1061800.52
$w_7^\star$			1042400.18
$w_8^\star$			-557682.99
$w_9^\star$			125201.43

Motivating Regularization

- What if we add a term to our loss?

$$L(y, f(x; w)) = (y - w \cdot x)^2 + \lambda w^2$$



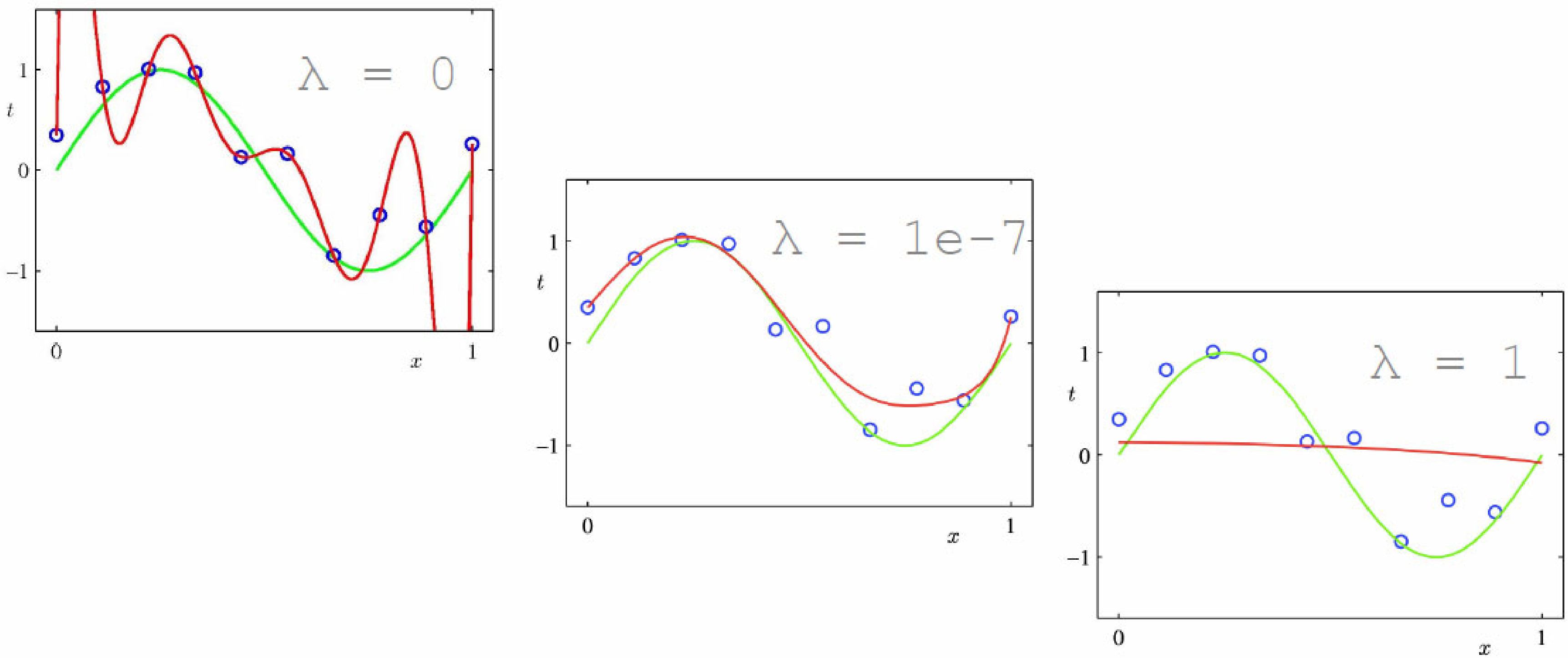
How Do We Stop Overfitting?

$$L(y, f(x; w)) = (y - w \cdot x)^2 + \lambda w^2$$

	$\lambda = 0$	1e-7	1
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

How Do We Stop Overfitting?

$$L(y, f(x; w)) = (y - w \cdot x)^2 + \lambda w^2$$



Regularized Least Squares

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\{y_i - \mathbf{w}^T \cdot \mathbf{x}_i\}^2$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\{y_i - \mathbf{w}^T \cdot \mathbf{x}_i\}^2 + \lambda\frac{1}{2}\mathbf{w}^T\mathbf{w}$$

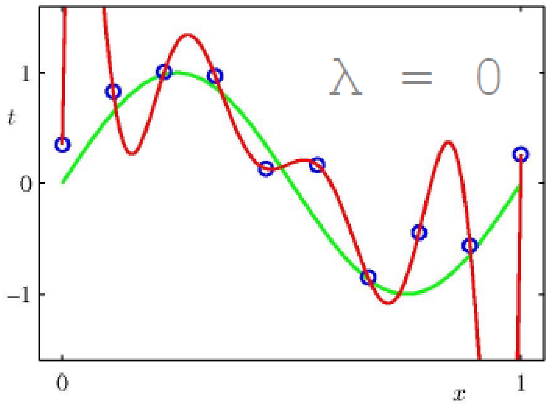
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

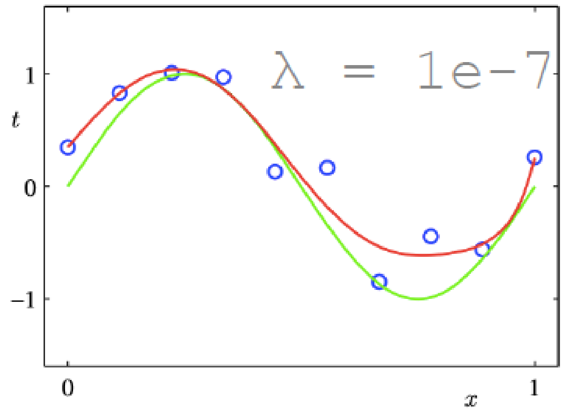
$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

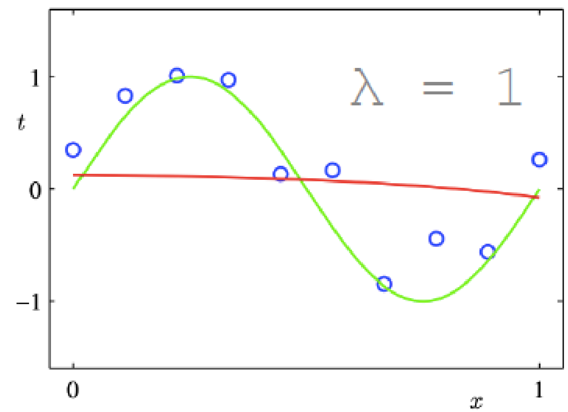
How Do We Stop Overfitting?

$$L(y, f(x; w)) = (y - w \bullet x)^2 + \lambda w^2$$

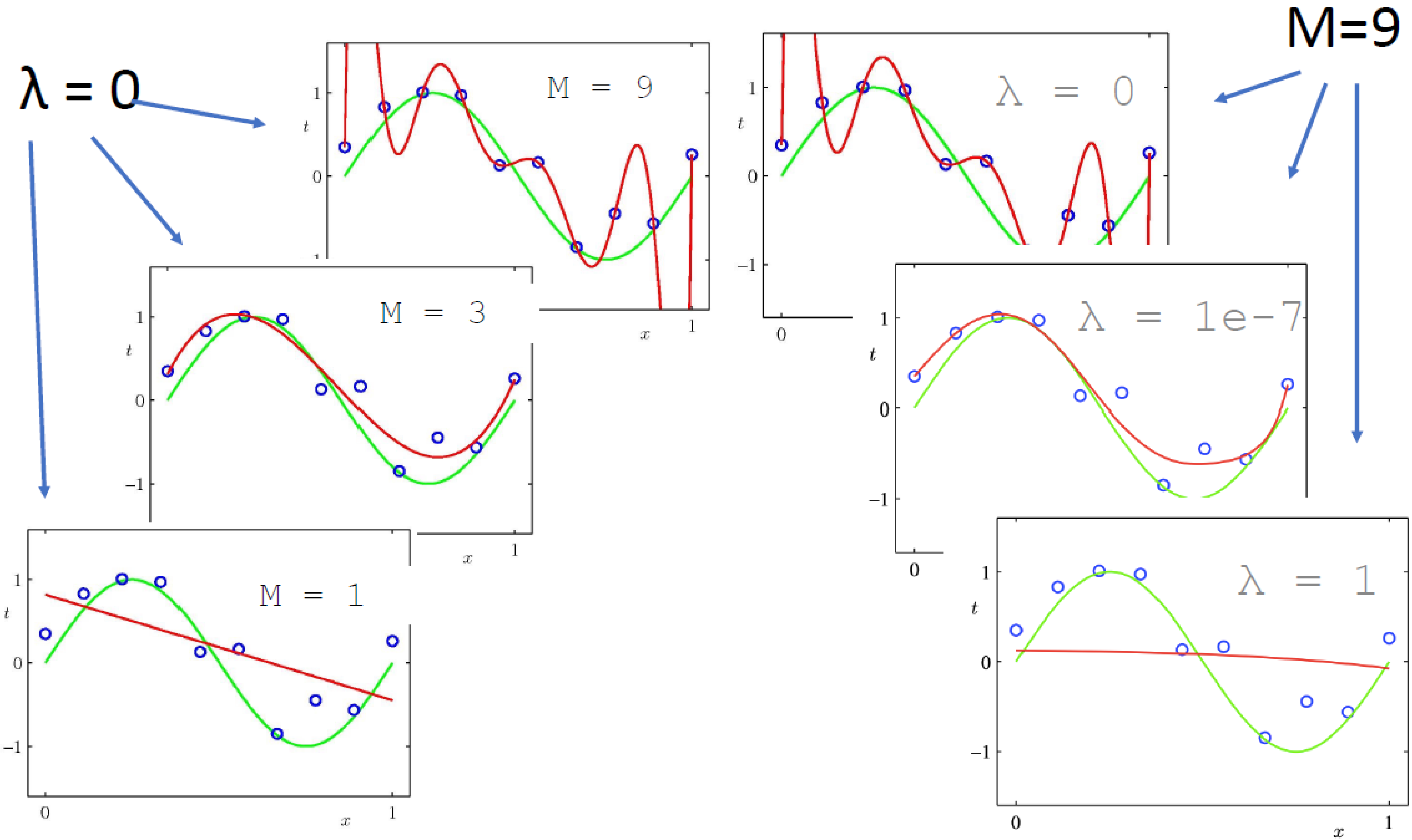
$$\mathbf{w} = \left(\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{Y}$$





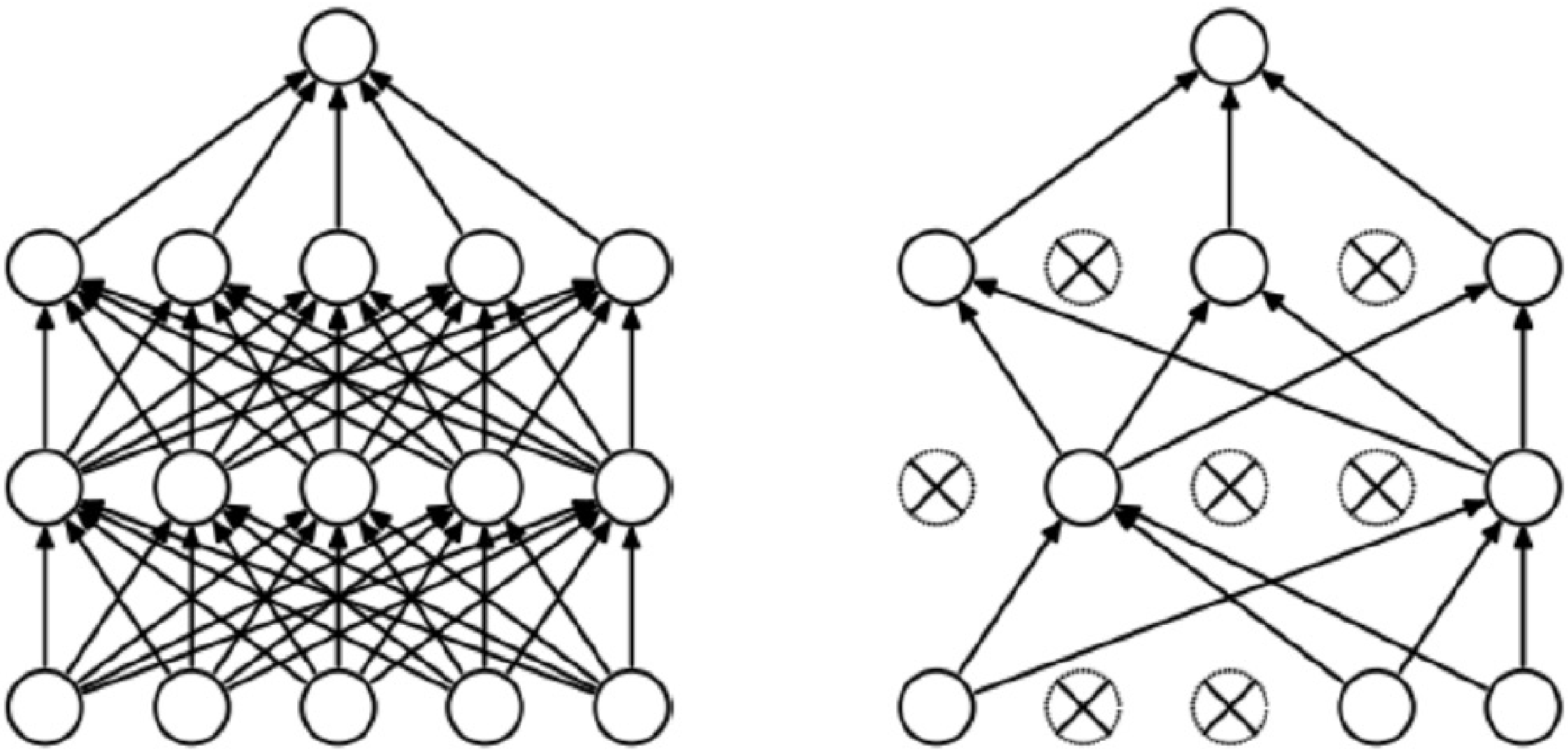


Summary of Regularization



Dropout

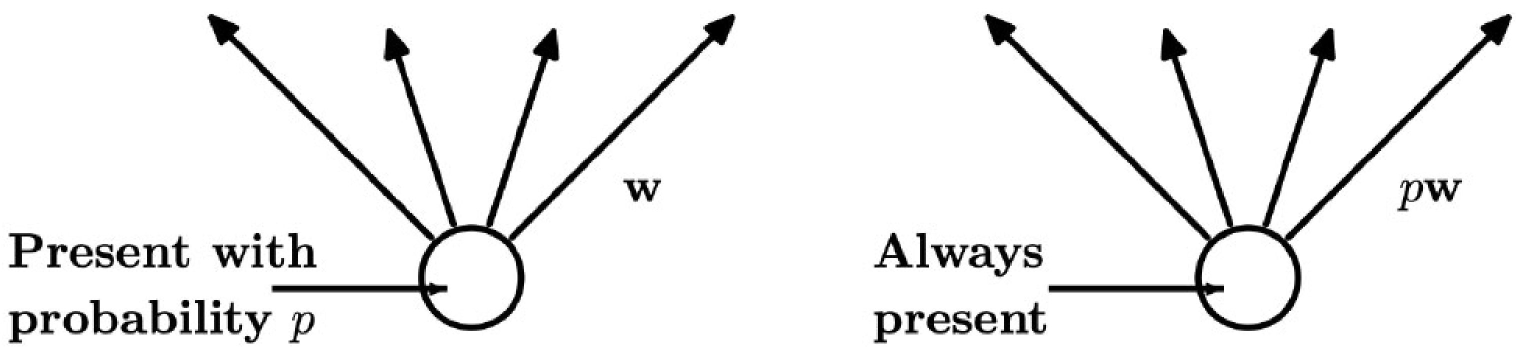
- Instead of modifying objective function (regularization), modify the **network structure**



(a) Standard Neural Net (b) After applying dropout.

Dropout

- Randomly "dropout" some neurons with probability p
- The chosen neurons change on each mini-batch
- All other learning remains unchanged



(a) At training time (b) At test time

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research* 15.1 (2014

Why Does Dropout Work?

- **Noise robustness**
  - Dropout is similar to making random perturbations in the input
  - These small changes shouldn't change the prediction for an input
  - Think "reduces variance"

Why Does Dropout Work?

- Prevents neuron co-adaptation
  - Neurons, especially near the top of the network, suffer from co-adaptation
  - The neurons take on values that rely on other neurons for correct predictions
  - Multiple points of failure: any single neuron failing can invalidate the others
  - Dropout prevents co-adaptation by randomly removing some of these neurons
  - Network is forced to learn more robust features that are useful with many different random subsets of other neurons

Why Does Dropout Work?

- Ensemble training
  - Similar to random forests or boosted models: we can combine many classifiers to produce better output
  - During training, each random permutation is one of 2^n possible neural networks
  - All networks share weights so number of parameters is still $O(n^2)$
  - Dropout training is like training a collection of 2^n "thinned" networks with extensive weight sharing
  - Test time: approximate the average prediction of all networks by using a single network without dropout

Dropout Takeaways

- Works really well for a wide variety of network types and domains
- Very easy to implement with no modifications in training
  - Some minor details (omitted) are important

- Drawbacks
  - Need to select dropout rate
  - This can depend on types of layers and network structure

Data Augmentation

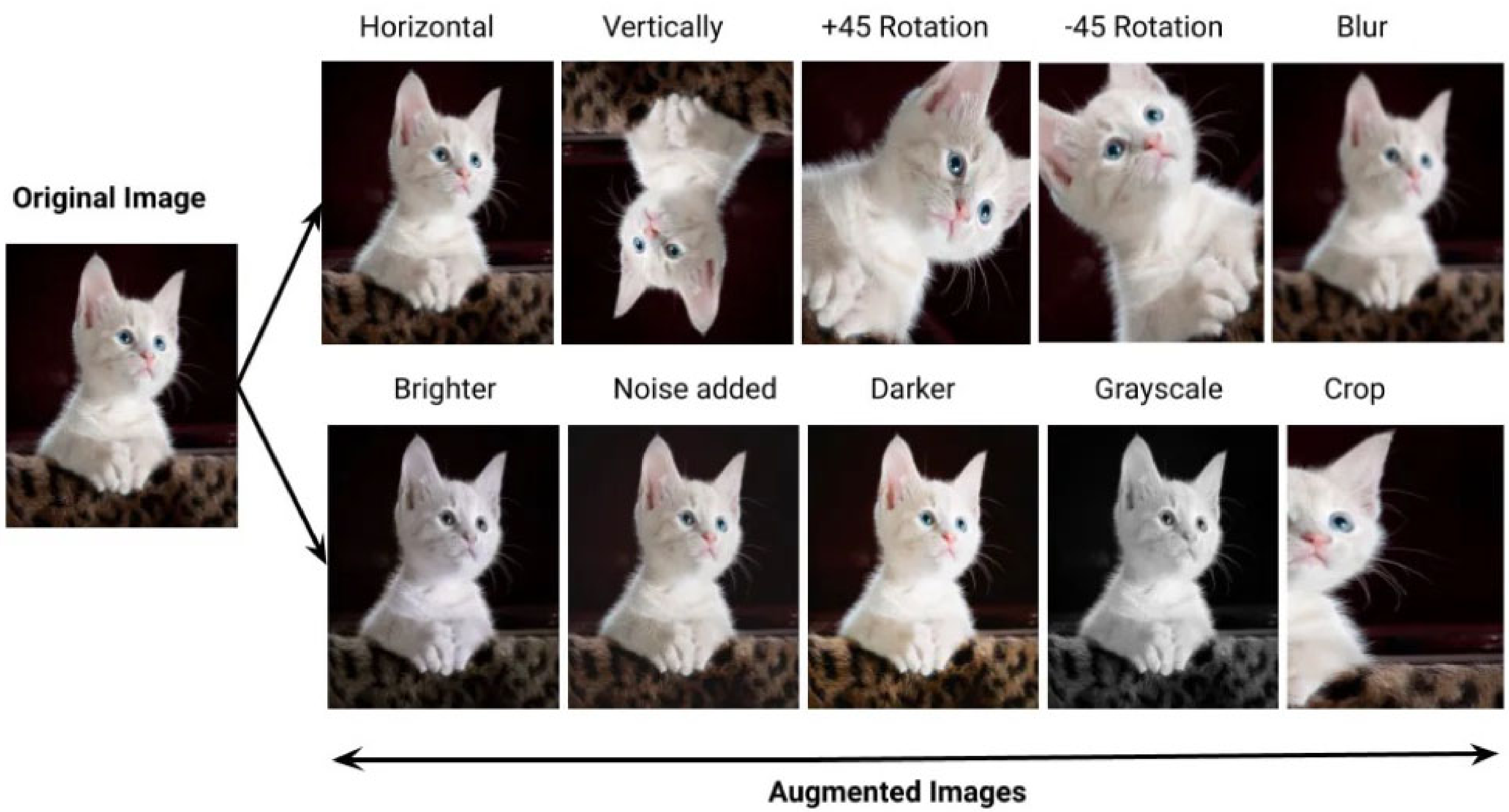
- Make perturbed copies of your data that vary in ways that should not change the value nature of the output function.

- This can help prevent spurious correlations between data and output.

- Example: Distinguishing clarinet sounds from flute sounds
  - Vary the pitch of each note by + or – 1%, 2%, 3%, 4%....
  - Add background noise of different kinds and at different dB
  - Time-stretch each note a bit
  - Delay or advance the onset of the note

- This can turn 1000 data points into 100,000.

Image Augmentation



More Image Augmentation








Image Augmentation -- What if….

Your training data looks like this?

High dynamic range
Very bright

Your testing data looks like this?

Low dynamic range
Very dark

Image Augmentation -- What if…

- Standardize your data:
  - Make sure that you have unit variance in your batch/dataset.

- Give your data the same range overall (e.g. center your values around the same center point)

- Decorrelate your variables (can be harder for images, if every pixel is a variable)

Batch Normalization

- Normalize the data scale input to each node
- Subtract the mean value of the data
- Do this on a dimension-by-dimension basis
- Do this at every training step in gradient descent

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization