LG전자 Deep Learning 과정

Regularization

Gunhee Kim

Computer Science and Engineering

서울대학교
SEOUL NATIONAL UNIVERSITY

Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

Regularization

Any modification we make to a ML algorithm that is intended to reduce its generalization error but not its training error

Several strategy

- Extra constraints on a machine learning algorithm
- Extra terms in the objective (based on prior knowledge)
- Ensemble methods (combine multiple hypotheses that explain the training data)

Parameter Norm Penalties

Basic form

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

- $J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$: original objective
- $\Omega(\boldsymbol{\theta})$: a parameter norm penalty
- $\alpha \in [0, \infty)$ a hyperparameter that weights the relative contribution of the penalty

Meaning

- Decreasing $\tilde{J}$: decrease both the original objective J on the training data + some measure of the size of parameter $\boldsymbol{\theta}$
- Different choice of $\Omega(\boldsymbol{\theta})$ results in a different solution

L2 Regularization

Drive the weights closer to the origin by adding the term

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{w}||_2^2$$

- Also known as *ridge regression* or *Tikhonov regularization*

Another option: Penalize the nonzero elements

$$\Omega(\boldsymbol{\theta}) = \alpha||\boldsymbol{w}||_1$$

Norm Penalties for Neural Networks

L2 regularization (weight decay) is a common practice

Constraining the norm of each column of the weight matrix of each layer

- Not for the entire matrix
- Prevents any one hidden unit from having very large weights
- A separate α for each column

L1 regularization is only used if having a strong reason

Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

Data Augmentation

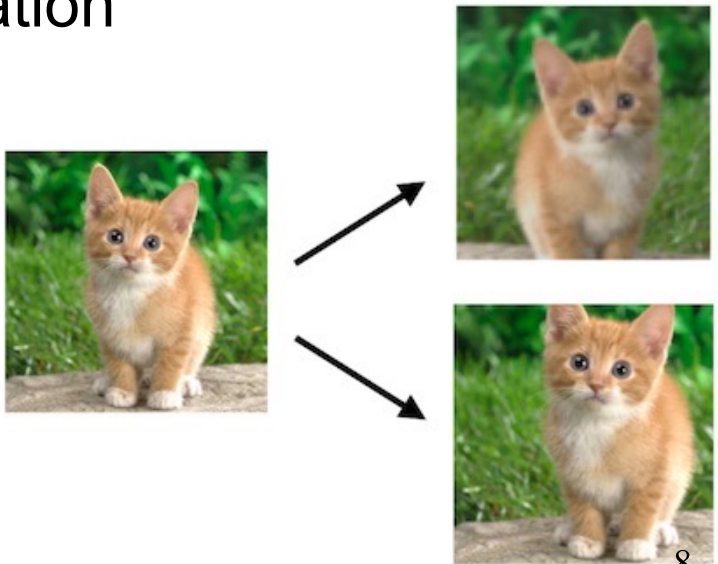
The best way to make ML model generalize better:
More training data

Create fake data from training data!

- For a training sample $(\boldsymbol{x}, y)$, make some transformation $(\boldsymbol{x}', y)$

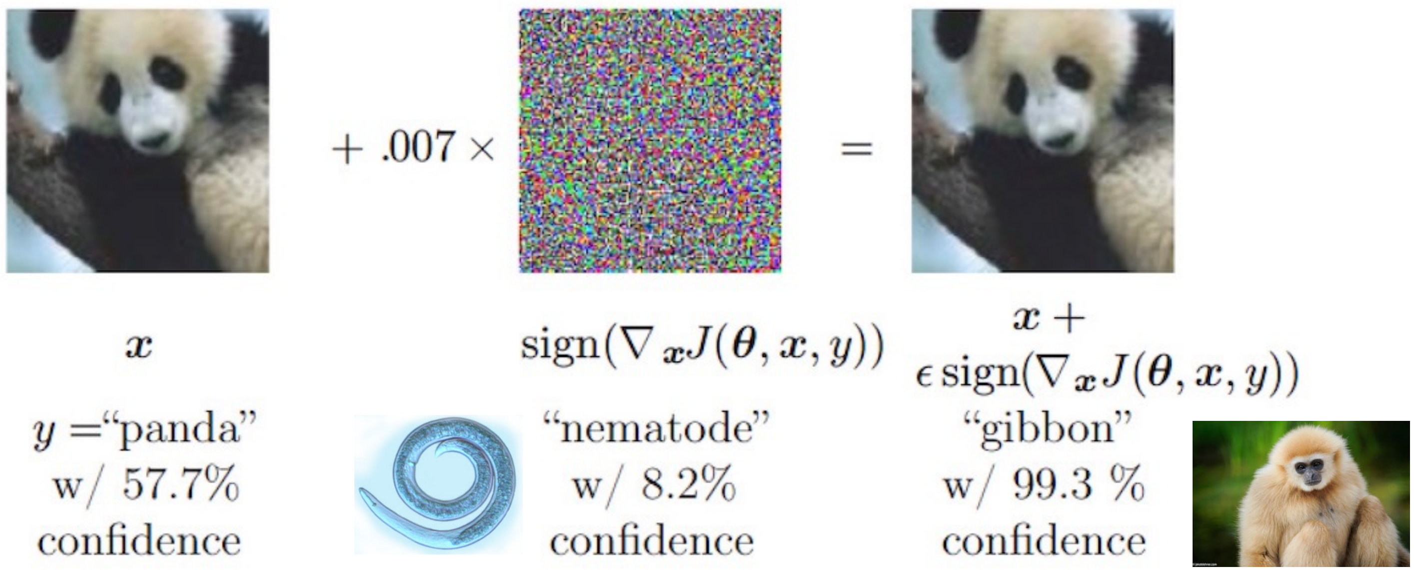
Very effective for image classification

- Images are high-dimensional and have an enormous variety of factors of variation
- (Partial) translating by a few pixels, rotating, scaling, and flipping



Noise injection

Neural networks are not robust against noise



Noise injection

Injecting noise in the input can be seen a form of data augmentation

- Highly effective
- Can be seen as regularization

Different ways of noise injection

- A random noise to input
- Noise injection to parameters (or models) (~ Tikhonov regularization)
- Noise at the output targets (e.g. label smoothing)

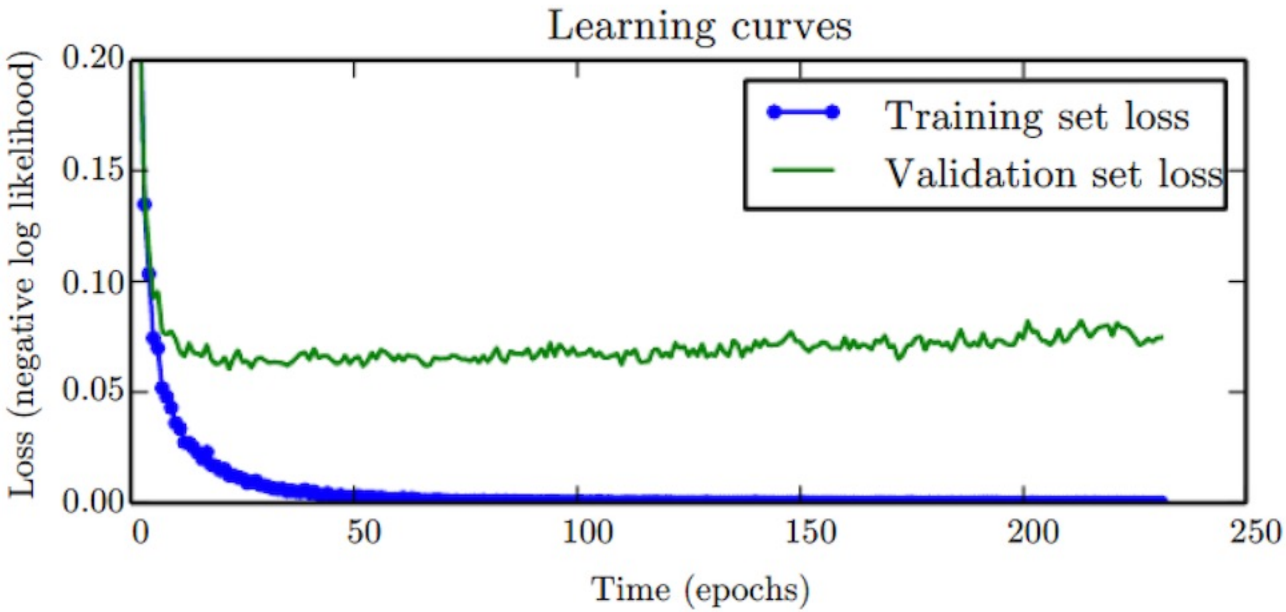
Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

Early Stopping

A conventional learning cover (negative log-likelihood)

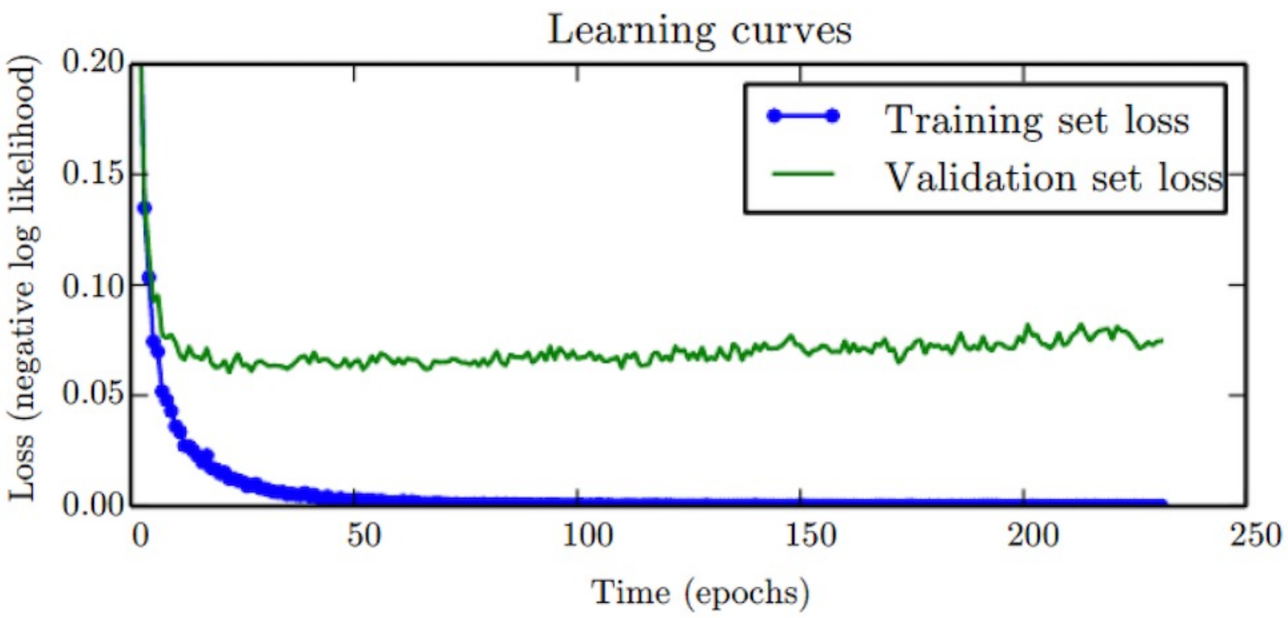
- e.g. Maxout network on MNIST
- The training objective decreases consistently over time
- The validation loss forms an asymmetric U-shaped curve



Early Stopping

Idea: use the parameters at the minimum validation error

- Maintain a copy of model parameters every iteration
- Additional cost is negligible (+ occasional slow writes)
- A kind of hyperparameter selection algorithm (training time)



Properties of Early Stopping

A very unobtrusive form of regularization

- No change in training procedure, objective, etc.
- Can be used jointly with other regularization

Another strategy for using all of the training data

- Early stopping requires a validation set (could seem wasteful)
- Learning the model with training data w/o validation set first + Continue training using all data until the validation loss falls below the training loss

Regularization effect (decreasing generalization error) + reduction of training time

Outline

- Parameter Norm Penalties
    - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

Ensemble Methods

Combine opinions of multiple learning algorithms or models

Does not innovate on base learning algorithm/model

- Decision Trees, SVMs, etc.

Innovates at higher level of abstraction

- Bagging: creating **multiple training sets** via bootstrapping, and then combines by averaging prediction
- Boosting: training **multiple models** (called weak learner), from which a single strong learner is created.

Why Ensemble Methods Work ?

Answer: Bias-Variance Tradeoff!

Bagging reduces variance of low-bias models

- Low-bias models are complex and unstable
- Bagging averages them together to create stability

Boosting reduces bias of low-variance models

- Low-variance models are simple with high bias
- Boosting trains sequence of models on residual error
  ➔ Sum of simple models is accurate

Two Basic Supervised Learning Problems

Classification

$$f(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{b}) = \operatorname{sign}(\boldsymbol{w}^T\boldsymbol{x} + \boldsymbol{b})$$

- Predict which class an example belongs to
- e.g., spam filtering example

Regression

$$f(\boldsymbol{x}|\boldsymbol{w},\boldsymbol{b}) = \boldsymbol{w}^T\boldsymbol{x} + \boldsymbol{b}$$

- Predict a real value or a probability
- e.g., probability of being spam

Both problems are highly inter-related

- Train on regression → Use for classification

Formal Definitions

Given training data and selected model class (a.k.a. hypothesis class)

$$S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}, \qquad \boldsymbol{x} \in \mathbb{R}^D, y \in \{-1, +1\}$$

$$h(\boldsymbol{x}|\boldsymbol{w}, \boldsymbol{b}) = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b} \quad \text{(Linear model)}$$

Goal: find $(\boldsymbol{w}, \boldsymbol{b})$ that predicts **well** on S

Loss function

- ex. The squared loss for regression $\quad L(a, b) = (a - b)^2$
- ex. 0/1 loss for classification, $\quad L(a, b) = \mathbf{1}_{[a \neq b]}$ or $\mathbf{1}_{[\mathrm{sign}(a) \neq \mathrm{sign}(b)]}$

Learning objective (optimization)

$$\mathrm{argmin}_{w,b} \sum_{i=1}^{N} L(y_i, h(\boldsymbol{x}_i|\boldsymbol{w}, \boldsymbol{b}))$$

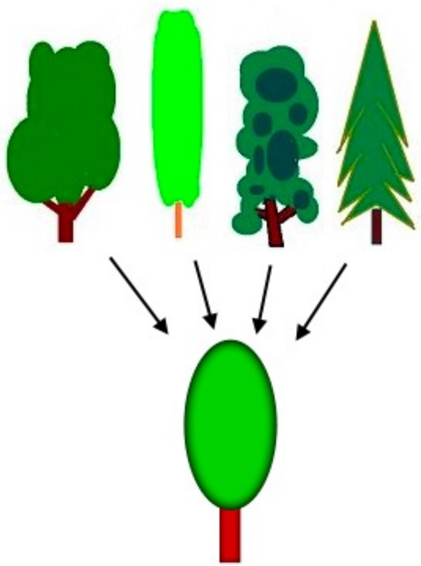
Generalization

Objective of learning

- Not to learn an exact representation of the training data itself
- To build a statistical model of the process that generates the data

Generalization

- A form of abstraction where common properties of specific instances are formulated as general concepts or claims
- It extracts the essence of a concept based on its analysis of similarities from many discrete objects
- If a toddler had never seen a willow tree or pine tree before he still might classify it as a tree because it is green



(Source) https://en.wikipedia.org/wiki/Generalization

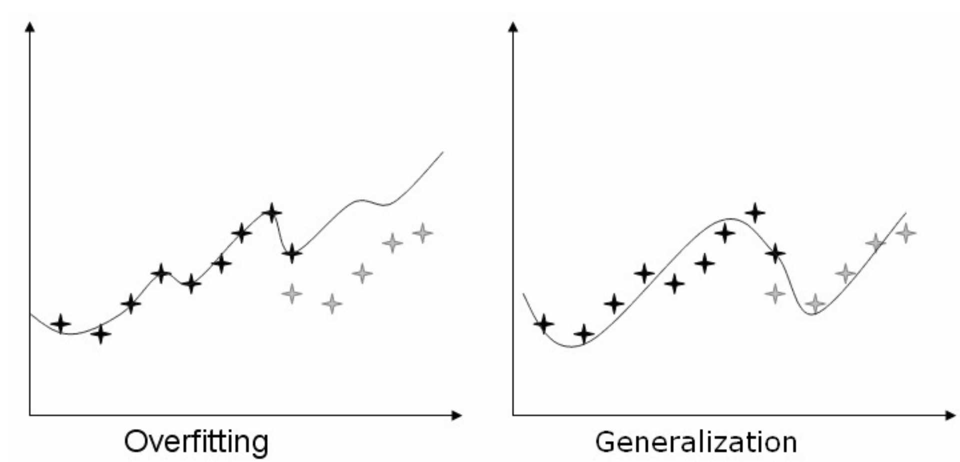
Generalization

Generalization in ML

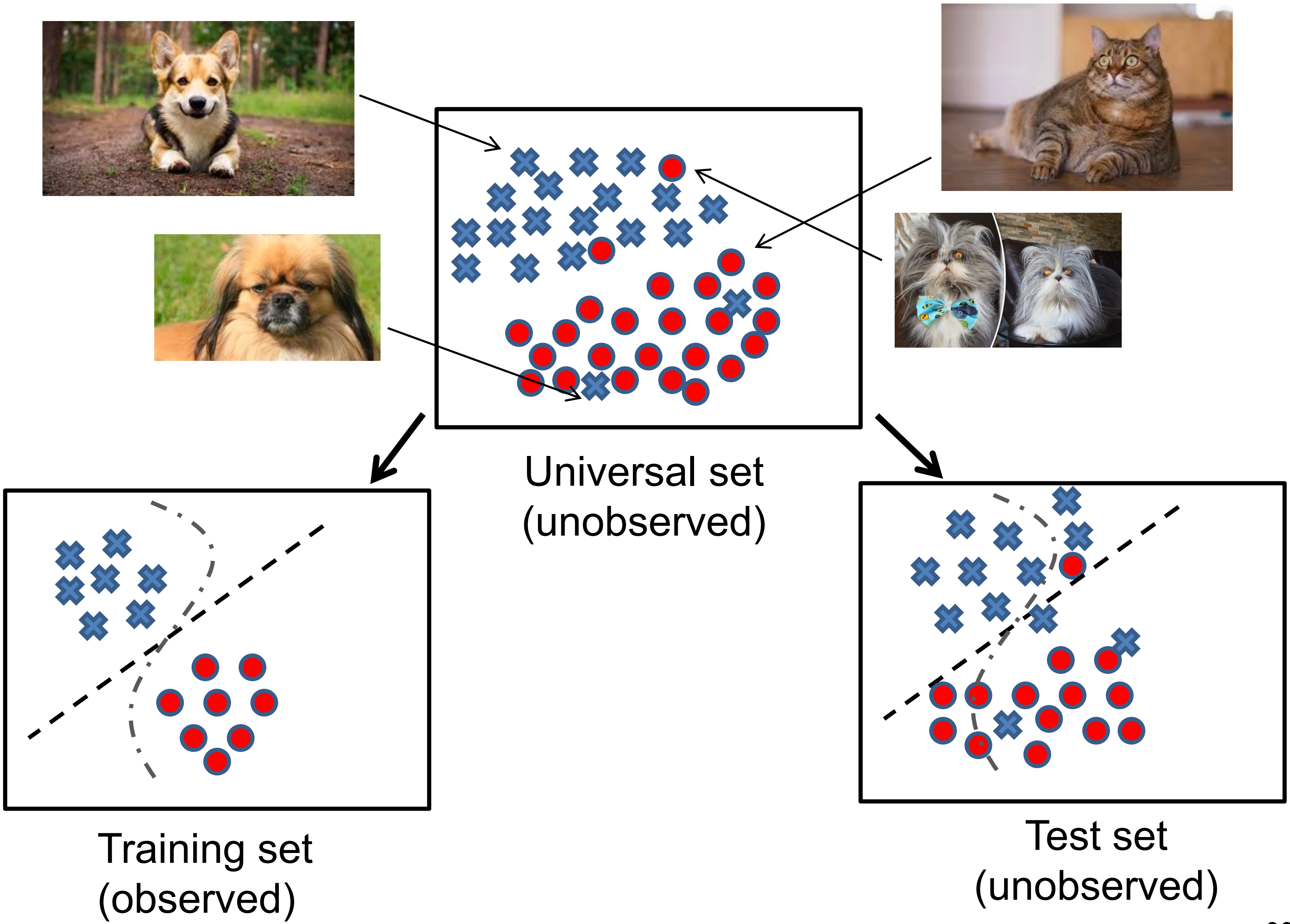
- An ML model's ability to perform well on new unseen data rather than just the data that it was trained on
- Learning algorithm maximizes accuracy on training examples
- How to generalize from training to test?

Strongly related to the concept of overfitting

- Overfitting = poor generalization



Training Data vs Test Data



Generalization Error

True distribution: $P(x, y)$

- All possible cases – unknown to us
- Training and test data are generated by $P(x, y)$
- Assumption: i.i.d (independent and identically distributed)

Training: fit an hypothesis $h(x)$

- Using training data $S = \{(x_i, y_i)\}_{i=1}^{N}$, sampled from $P(x, y)$

Generalization error: $L_P(h) = \mathbb{E}_{P(x,y)}[L(y, h(x))]$

- Prediction loss on all possible cases
- Overfitting**:** Generalization error > Training error
- Underfitting**:** Generalization error < Training error

Bias/Variance Decomposition

Test error $L_P(h) = \mathbb{E}_{P(x,y)}[L(y^*, h(x^*))]$

- Given a new test sample (x^*, y^*) where $y^* = f(x^*) + \varepsilon$
- Squared loss $L(a, b) = (a - b)^2$

$$\mathbb{E}[\left(y^* - h(x^*)\right)^2] = \mathbb{E}[\left(y^* - f(x^*) + f(x^*) - h(x^*)\right)^2]$$

$$= \mathbb{E}[\left(y^* - f(x^*)\right)^2 + \left(f(x^*) - h(x^*)\right)^2 + 2(y^* - f(x^*))(f(x^*) - h(x^*))]$$

$$= \mathbb{E}[\left(y^* - f(x^*)\right)^2] + \mathbb{E}\left[\left(f(x^*) - h(x^*)\right)^2\right] + \underbrace{2\mathbb{E}[(y^* - f(x^*))(f(x^*) - h(x^*))]}_{\text{goes to 0!}}$$

$$\mathbb{E}[(y^* - f(x^*))(f(x^*) - h(x^*)] = \mathbb{E}[y^* f(x^*) - f(x^*)^2 - y^* h(x^*) + f(x^*)h(x^*)]$$

$\mathbb{E}[y^* f(x^*)] = E[f(x^*)^2]$ because $E[f(x^*)^2] = f(x^*)^2$ and

$\mathbb{E}[y^* f(x^*)] = f(x^*)\mathbb{E}[y^*] = f(x^*)^2 \qquad (\mathbb{E}[y^*] = f(x^*))$

$\mathbb{E}[y^* h(x^*)] = \mathbb{E}[f(x^*)h(x^*)]$ because $\mathbb{E}[y^* h(x^*)] = f(x^*)\mathbb{E}[h(x^*)]$

$\mathbb{E}[f(x^*)h(x^*)] = f(x^*)\mathbb{E}[h(x^*)]$

Bias/Variance Decomposition

Test error $L_P(h) = \mathbb{E}_{P(x,y)}[L(y^*, h(x^*))]$

- Given a new test sample (x^*, y^*) where $y^* = f(x^*) + \varepsilon$
- Squared loss $L(a, b) = (a - b)^2$

$$\mathbb{E}[\left(y^* - h(x^*)\right)^2] = \mathbb{E}[\left(y^* - f(x^*)\right)^2] + \mathbb{E}\left[\left(f(x^*) - h(x^*)\right)^2\right]$$

- Take a look at the second term

$$\mathbb{E}\left[\left(f(x^*) - h(x^*)\right)^2\right] = \mathbb{E}\left[\left(f(x^*) - \mathbb{E}[h(x^*)] + \mathbb{E}[h(x^*)] - h(x^*)\right)^2\right]$$

$$= \mathbb{E}\left[(f(x^*) - \mathbb{E}[h(x^*)])^2 + \left(\mathbb{E}[h(x^*)] - h(x^*)\right)^2 + \underbrace{(f(x^*) - \mathbb{E}[h(x^*)])\left(\mathbb{E}[h(x^*)] - h(x^*)\right)}_{\text{goes to 0!}}\right]$$

$$= (f(x^*) - \mathbb{E}[h(x^*)])^2 + \mathbb{E}[(h(x^*) - \mathbb{E}[h(x^*)])^2]$$

$$\mathbb{E}\left[(f(x^*) - \mathbb{E}[h(x^*)])\left(\mathbb{E}[h(x^*)] - h(x^*)\right)\right]$$

$$= \mathbb{E}[f(x^*)\mathbb{E}[h(x^*)] - \mathbb{E}[h(x^*)]^2 - f(x^*)h(x^*) + \mathbb{E}[h(x^*)]h(x^*)]$$

$$= \mathbb{E}[f(x^*)]\mathbb{E}[h(x^*)] - \mathbb{E}[h(x^*)]^2 - \mathbb{E}[f(x^*)]\mathbb{E}[h(x^*)] + \mathbb{E}[h(x^*)]^2 = 0$$

Bias/Variance Decomposition

Test error $L_P(h) = \mathbb{E}_{P(x,y)}[L(y^*, h(x^*))]$

- Given a new test sample (x^*, y^*) where $y^* = f(x^*) + \varepsilon$
- Squared loss $L(a, b) = (a - b)^2$

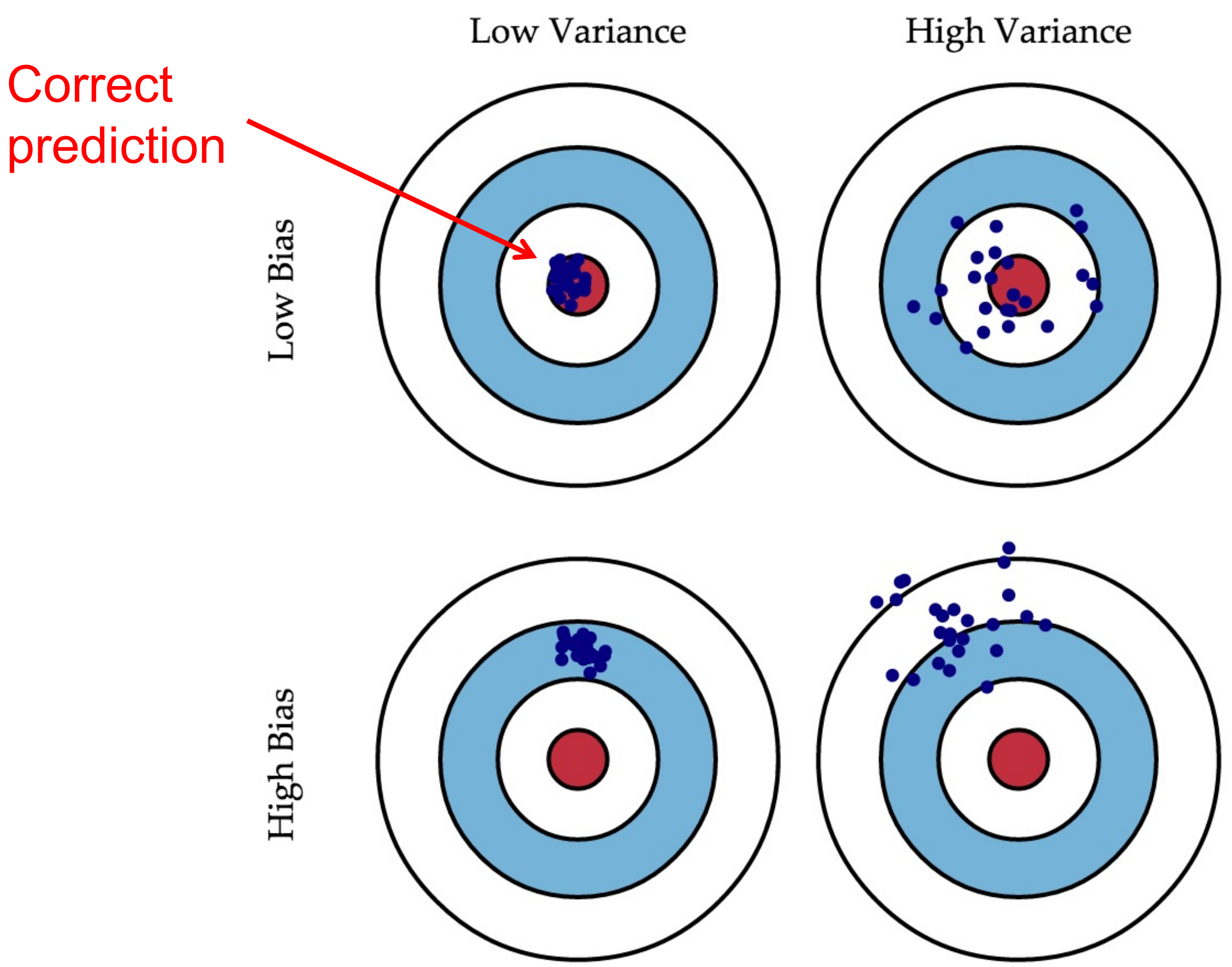
$$\mathbb{E}[\left(y^* - h(x^*)\right)^2] = \mathbb{E}\left[\left(y^* - f(x^*)\right)^2\right]$$ (noise)

$$+ (f(x^*) - \mathbb{E}[h(x^*)])^2$$ (Bias)^2

$$+ \mathbb{E}[(h(x^*) - \mathbb{E}[h(x^*)])^2]$$ (Variance)

- Average prediction $H(x^*) = \mathbb{E}[h(x^*)]$
- Variance term: how much each model varies from one training set to another
- Bias: describes the average error of each data

Bias/Variance Decomposition

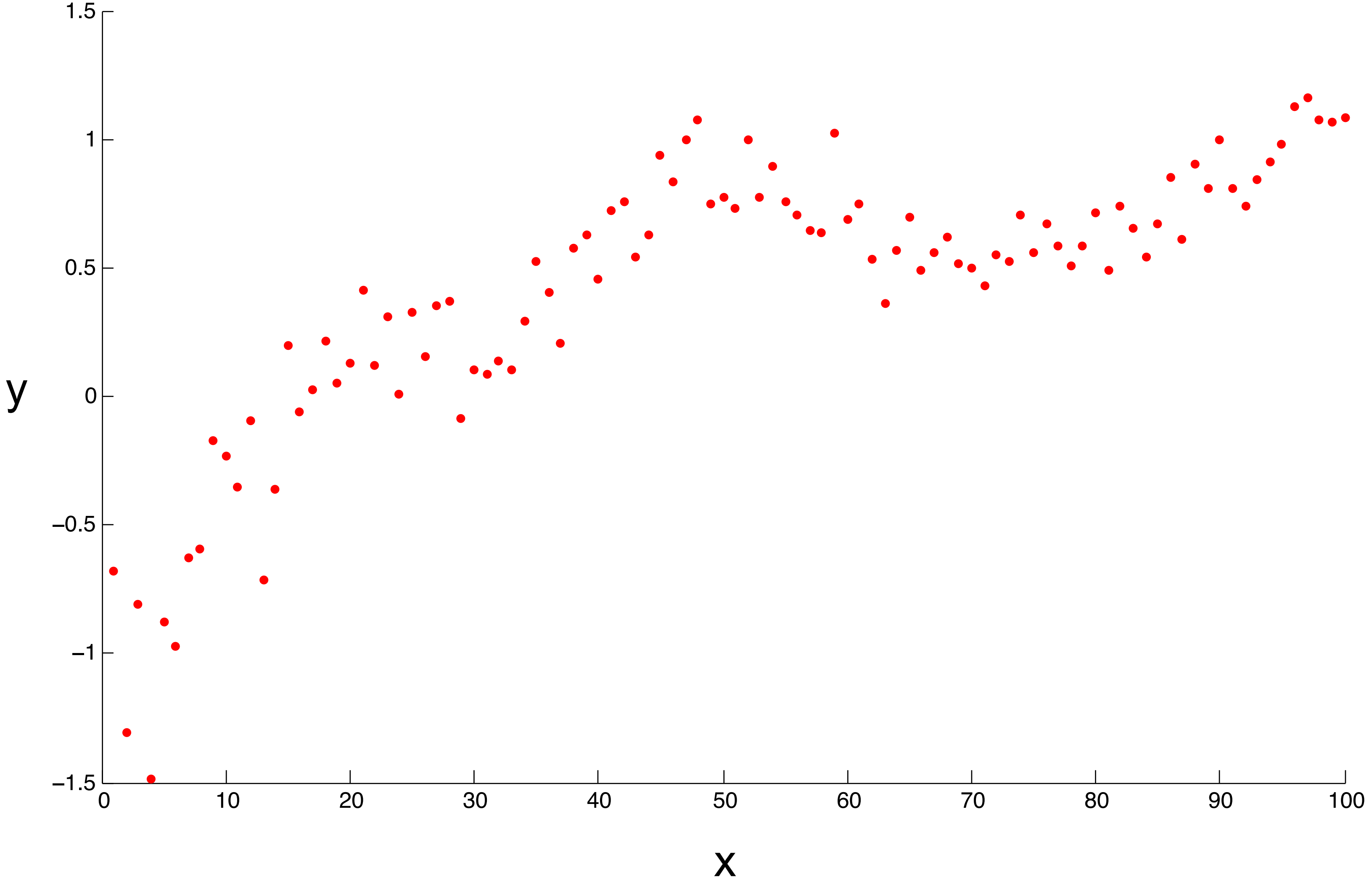


Low bias: predicted well

Low variance: Stable

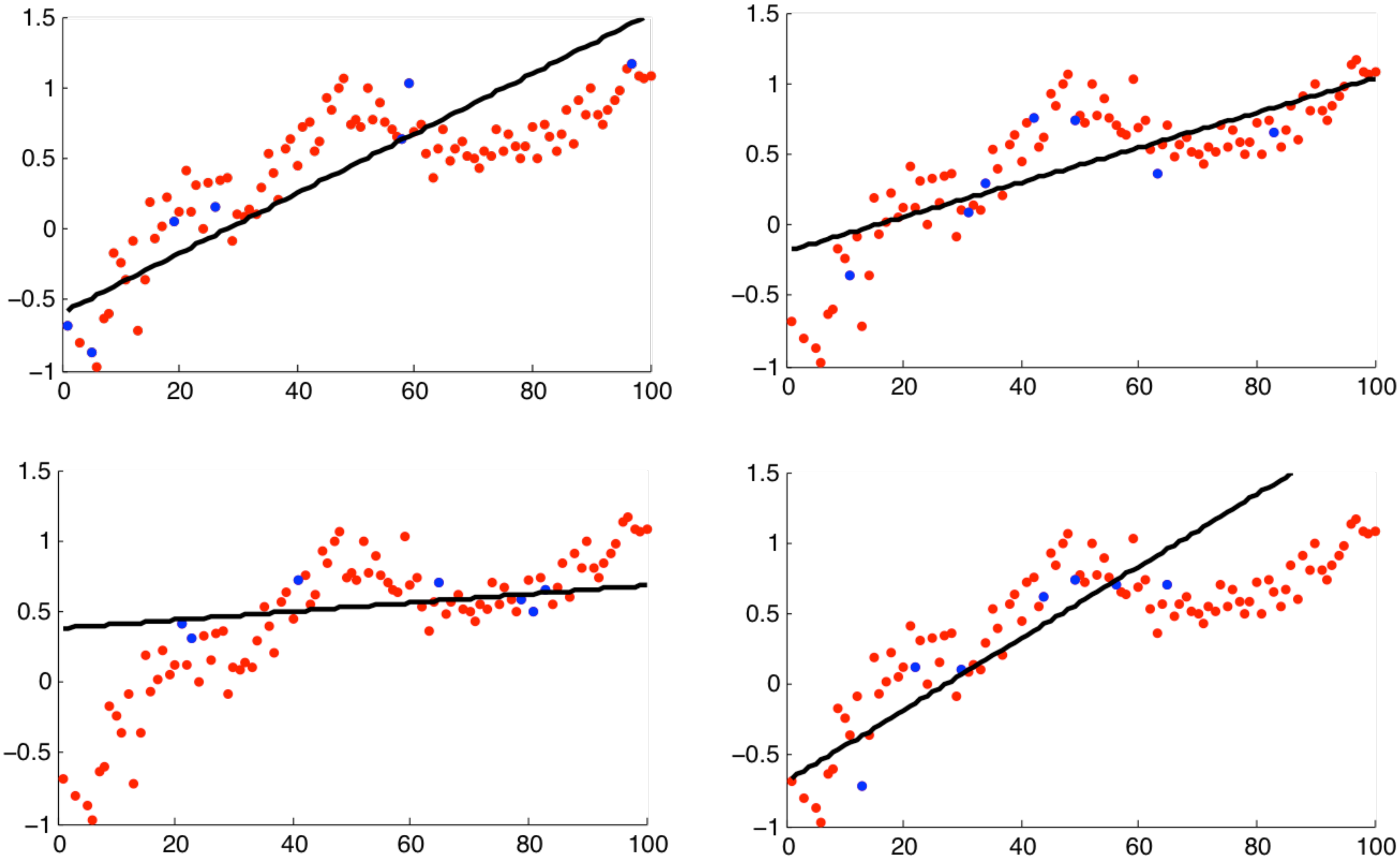
(Source) http://scott.fortmann-roe.com/docs/BiasVariance.html

Example $P(x, y)$



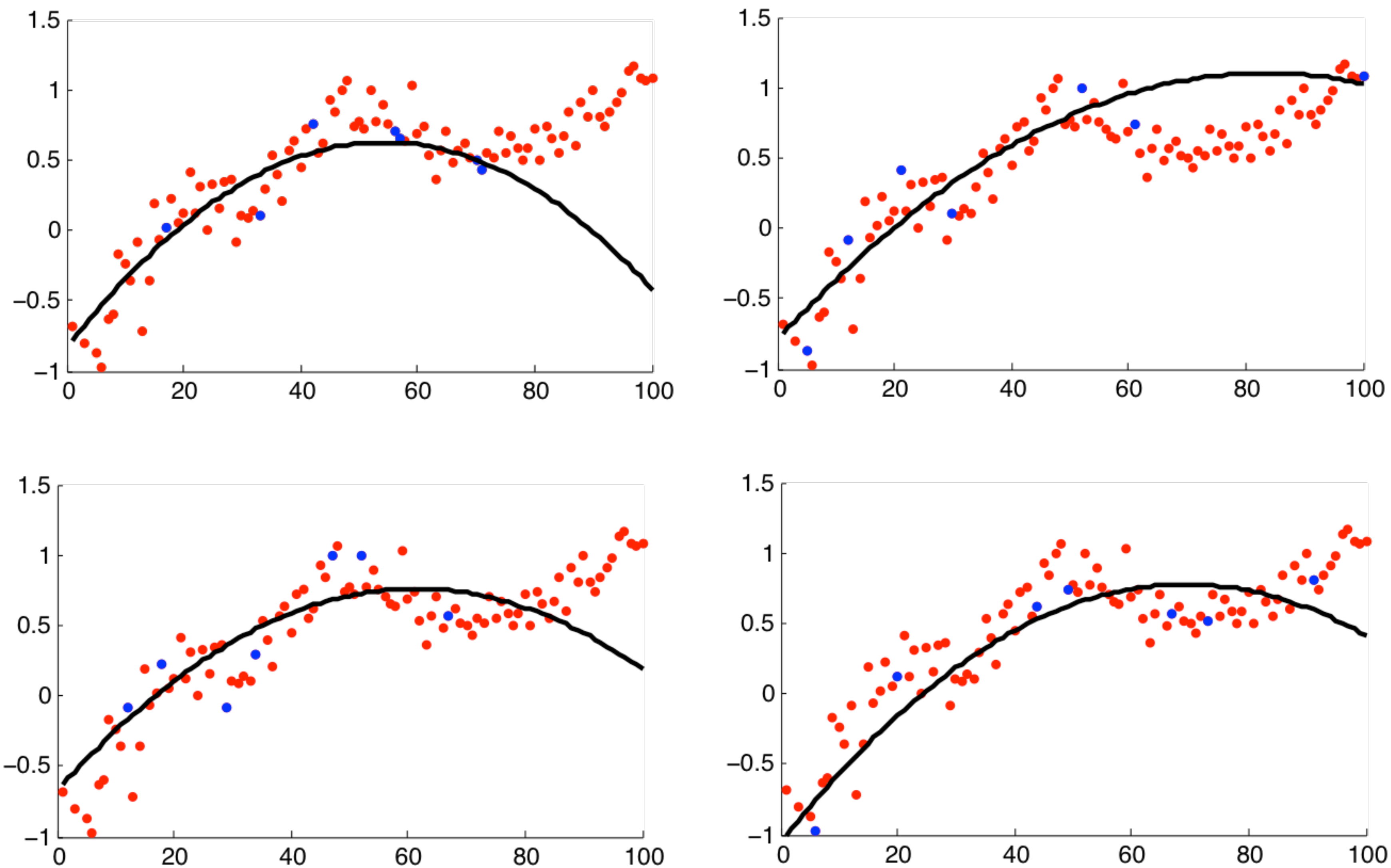
(Credits: Yisong Yue's Tutorial)

$h_s(x)$ Linear



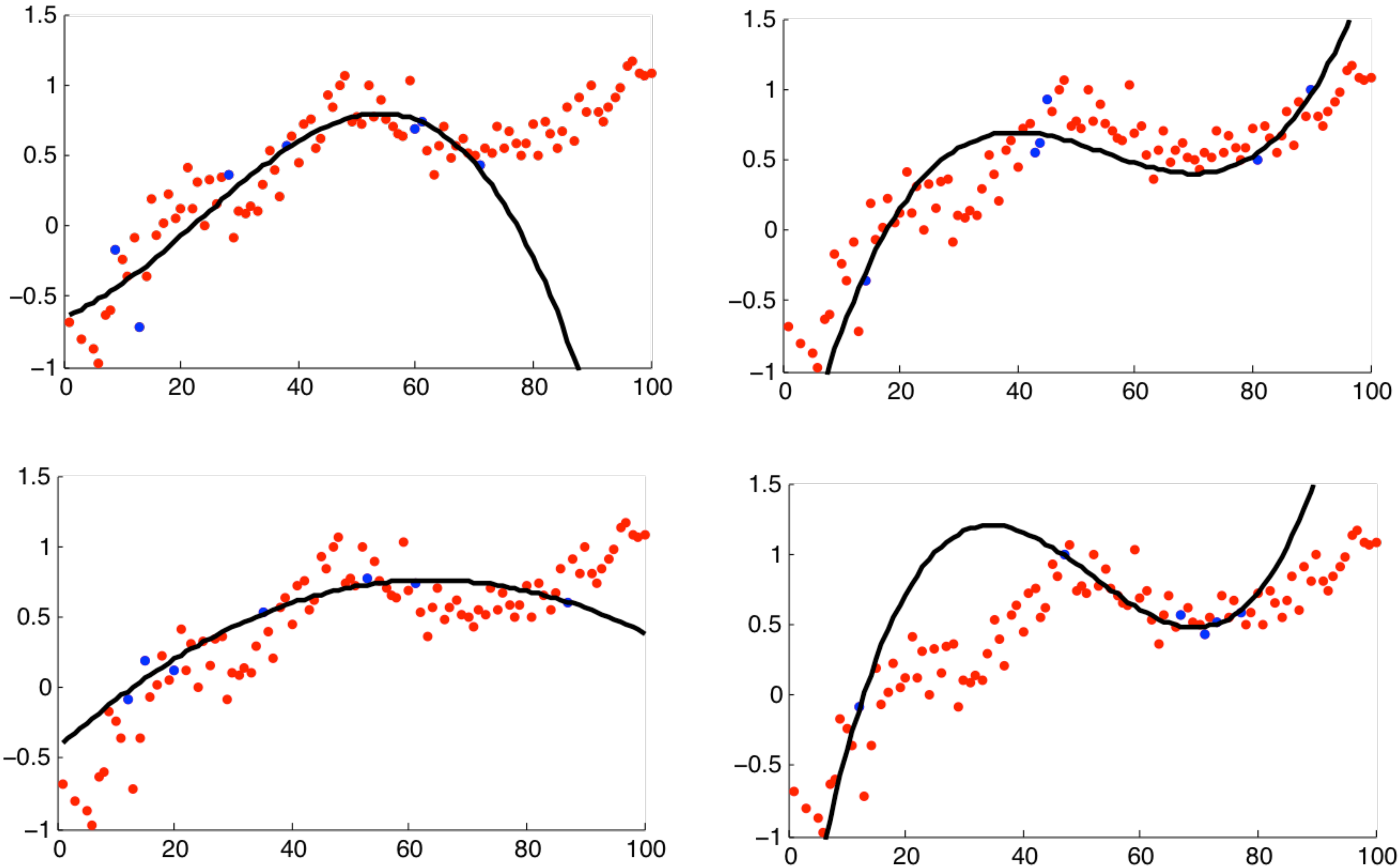
(Credits: Yisong Yue's Tutorial)

$h_s(x)$ Quadratic



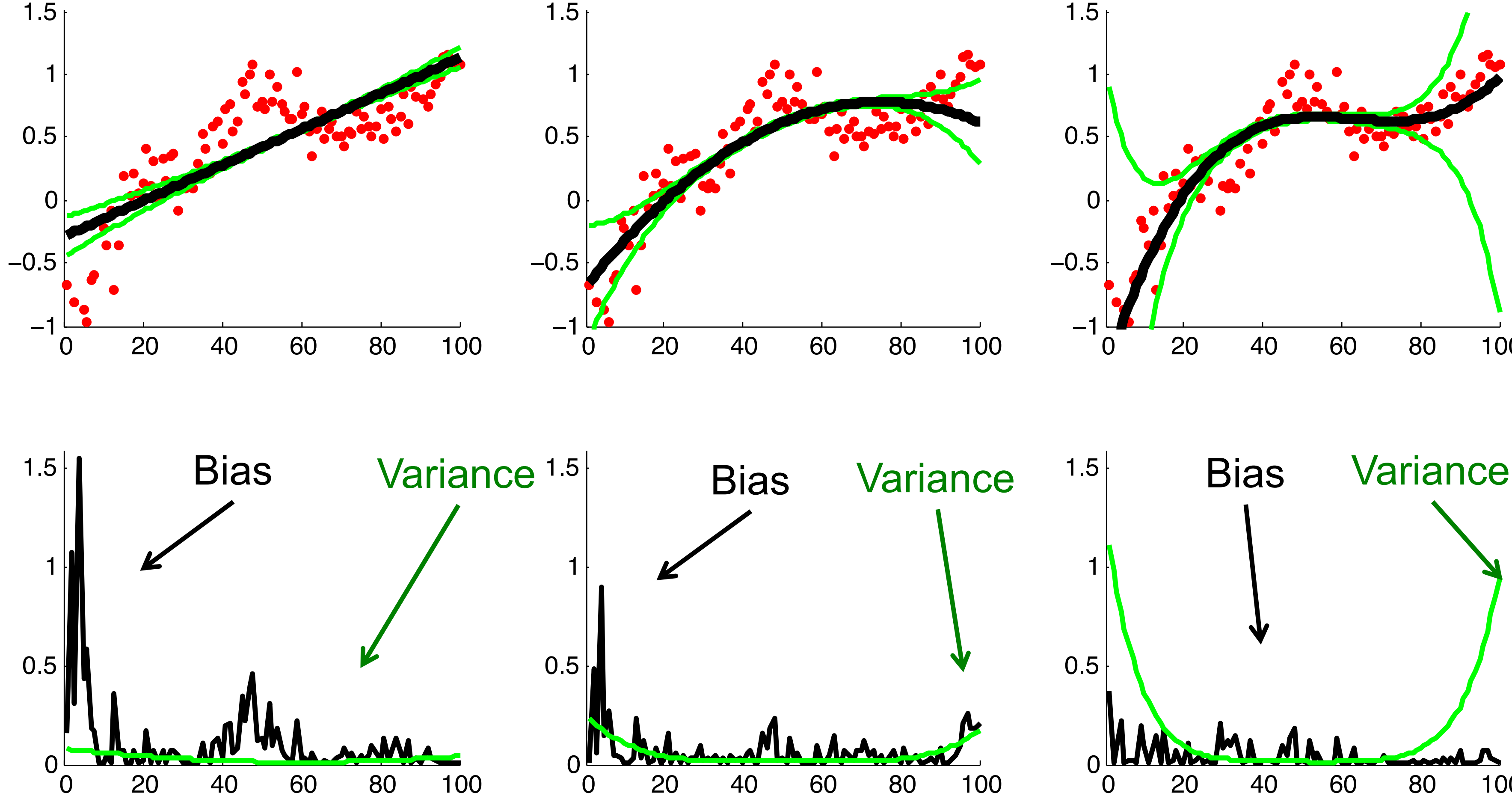
(Credits: Yisong Yue's Tutorial)

$h_s(x)$ Cubic



(Credits: Yisong Yue's Tutorial)

Bias-Variance Trade-off

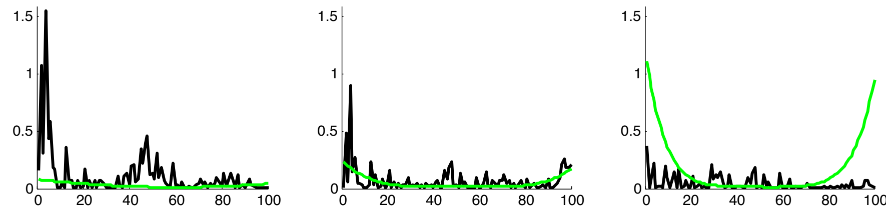


$$\mathbb{E}_S[L_P(h_S)] = \mathbb{E}_{(x,y)\sim P(x,y)}\left[\mathbb{E}_S\left[\left(h_S(x) - H(x)\right)^2\right] + (F(x) - y)^2\right]$$

Variance Bias

(Credits: Yisong Yue's Tutorial)

Overfitting vs Underfitting



High variance implies overfitting

- Model class unstable
- Variance increases with model complexity
- Variance reduces with more training data

High bias implies underfitting

- Even with no variance, model class has high error
- Bias decreases with model complexity
- Independent of training data size

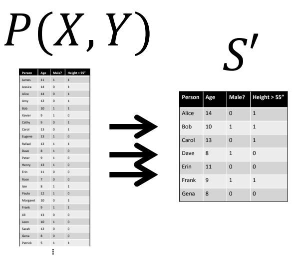
Bagging (Bootstrap Aggregating)

Goal: reduce variance

Expected error = **Variance** + Bias

Ideal setting: many independently sampled training sets S'

- Train model using each S'
- Average predictions



In practice: resample S' with replacement ($|S'| = S$)

- Called *Boostrapping*
- Variance reduces linearly and bias unchanged
- cf. Jackknife: Given a sample of size N, generate $N-1$ sets by ignoring one observation at each time

L. Breiman. Bagging predictors. Machine Learning 1996.

Bagging (Bootstrap Aggregating)

A general-purpose procedure for reducing the variance of a statistical learning method

- Given a set of n independent datasets $Z_1, \dots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n
- Averaging a set of training reduces variance
- It is not practical to multiple training sets, so instead we bootstrap (taking repeated samples from the single training set)

Bootstrap

The origination of the term

- From the idiom ***pull oneself up by one's bootstraps***
- Based on one of the 18C novel "The Surprising Adventures of Baron Munchausen" by Rudolph Erich Raspe:

  The Baron had fallen to the bottom of a deep lake. Just when it looked like all was lost, he thought to pick himself up by his own bootstraps

- It is not the same as the term "bootstrap" used in computer science meaning to "boot" a computer from a set of core instructions, though the derivation is similar.

Bagging (Bootstrap Aggregation)

Given: Training set S

Bagging: generate many bootstrap samples S'

Repeat B times

- Sampled with replacement from S ($|S'| = S$)
- Train minimally regularized classifier (ex. DT) on S'

Final Predictor: combine B predictors

- Voting for classification problems
- Averaging for estimation problems
- Averaging reduces variance

Ensemble Methods for Neural Networks

Highly recommended!

- Model averaging is an extremely powerful and reliable for reducing generalization error

NNs reach a wide variety of solution points

- Never obtain the same solution at each running
- random initialization, random selection of minibatches, different hyperparameter setting, …

Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

Dropout

A method of making bagging practical for ensembles of very many large NNs

- An inexpensive approximation to training and evaluating a bagged ensemble of exponentially many NNs

Suppose that we have a big NN model and optimize it with a minibatch-based learning algorithm (e.g. SGD)

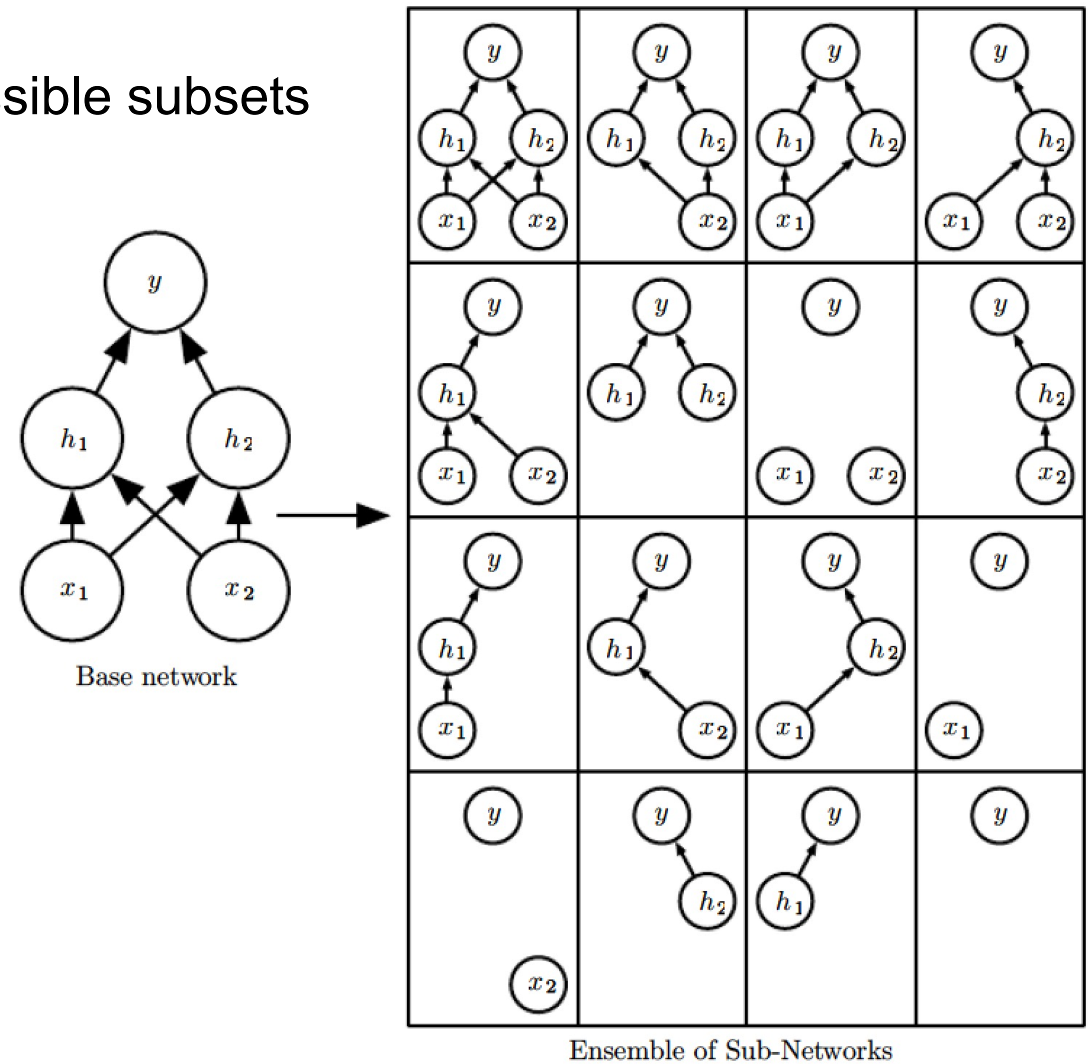
- For each minibatch, randomly sample a different binary mask to apply to all of the input and hidden units in the network
- In other words, randomly drop the output of units to zero
- e.g. dropout rate 0.5 for hidden units and 0.2 for input nodes
- Then use learning algorithm as usual

Dropout

A base network with two input and two hidden units

16 possible subsets

Ignore the ones with no input or no paths to output



Properties of Dropout

Comparison with Bagging

- In dropout, the models share parameters
- With each minibatch, a different subset of parameter is updated
- In bagging, the models are all independent

Dropout is very effective and highly recommended

- More effective than other regularizers (e.g. weight decay, sparsity)

Computationally very cheap

No limitation on model types or training procedures (e.g. CNNs, RNNs, and RBMs)

Properties of Dropout

Increase the size of the model to offset the regularizer

- Dropout reduces the effective capacity of a model

Do not use Dropout with very few training examples

Key insights

- Training a network with stochastic behavior and making predictions by averaging over multiple stochastic decisions
- Dropout power arises from that masking noise is applied to hidden units

Other relatives: dropconnect, batch normalization

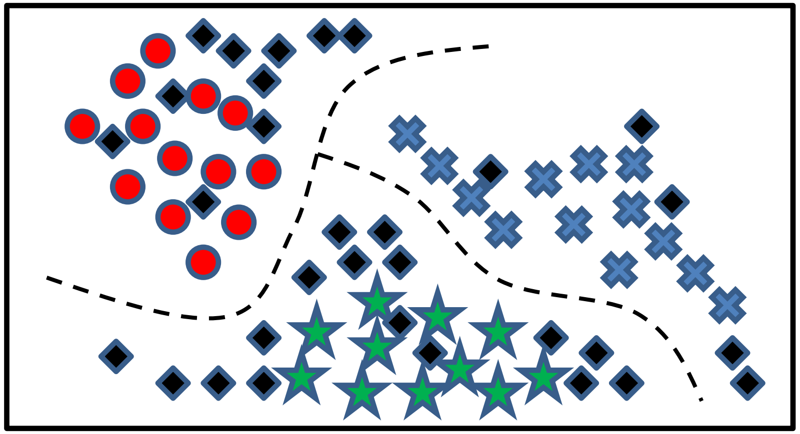
Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

Semi-supervised Learning

Use unlabeled data to augment a small labeled sample to improve learning

- Labeled data can be rare or expensive, while unlabeled data is much cheaper
- Model the generative process too
- One of active research areas



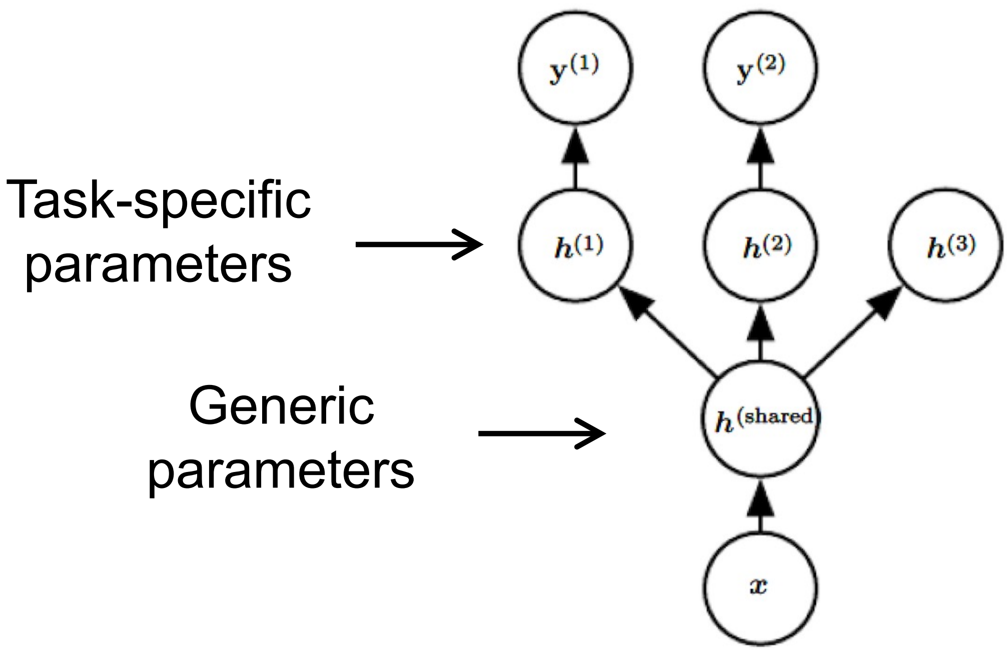
Semi-supervised learning

Multi-task Learning

Learn multiple-related problems together at the same time

- e.g. Image classification and object detection

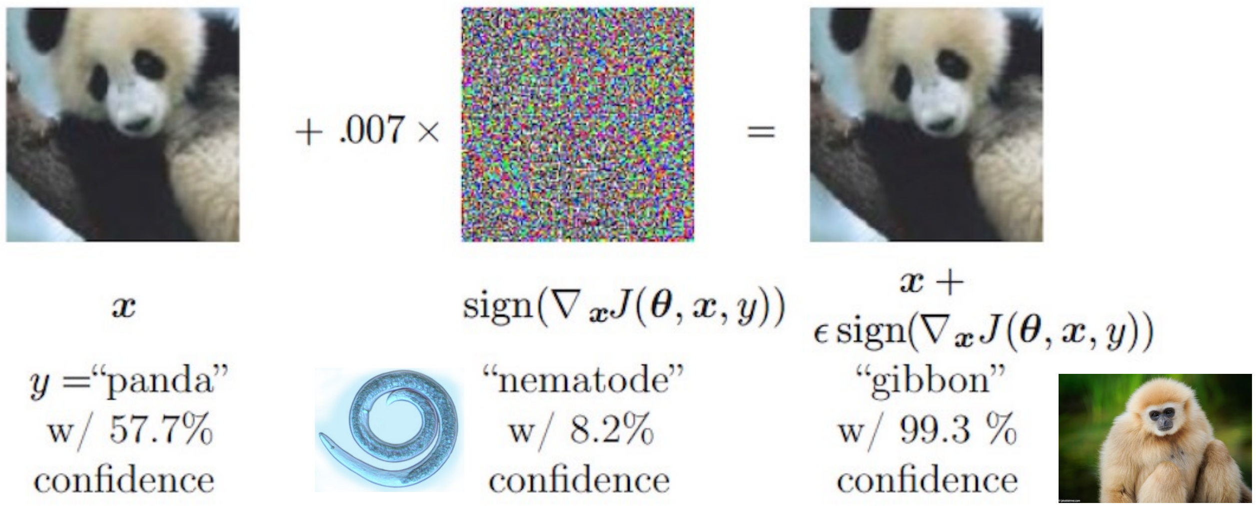
Two different supervised tasks (predicting $\boldsymbol{y}^{(i)}$), while sharing input $\boldsymbol{x}$ and some mid-level representation $\boldsymbol{h}^{(\text{shared})}$



Adversarial Training

Adversarial examples

- Human cannot tell the difference with the original example
- However, the network can make highly different predictions



Adversarial training: training on adversarially perturbed examples

- Purely linear models do not resist them