AI 539: TRUSTWORTHY ML

ML OVERVIEW

Instructor: Sanghyun Hong

sanghyun.hong@oregonstate.edu

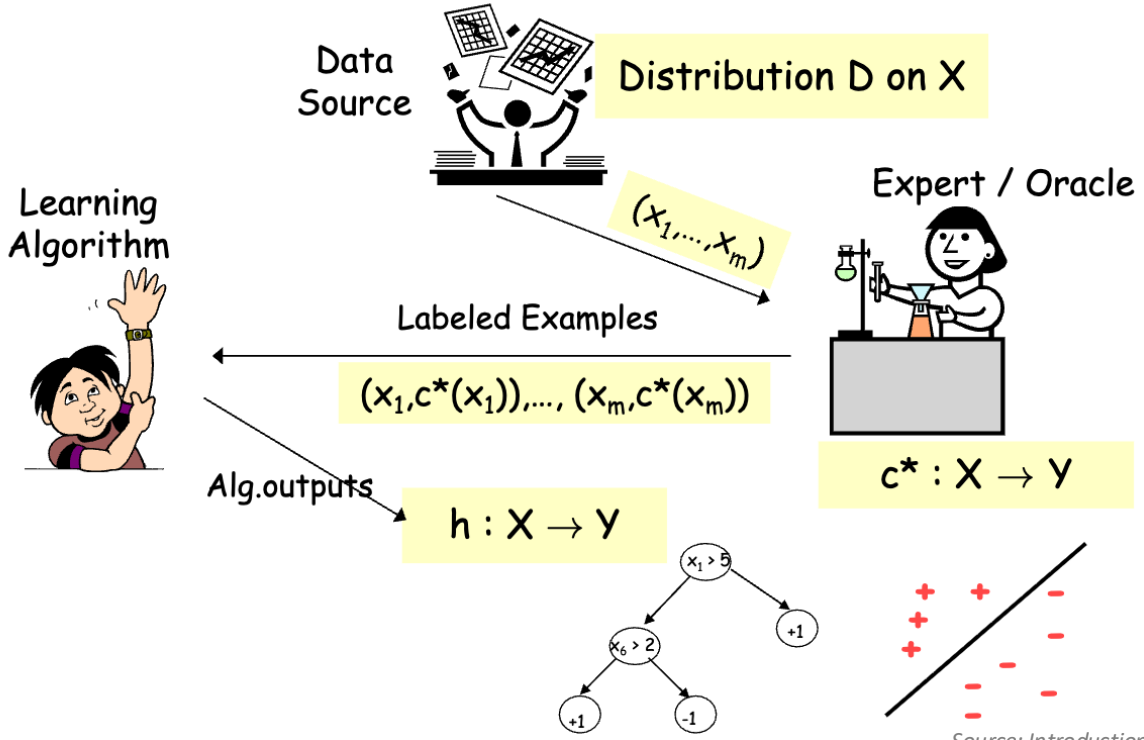
NOTES

- Call for actions
  - In-class presentation sign-ups
  - Term project team-up
  - On-boarding quiz on Canvas
  - GitHub classroom registration

PAC LEARNING

PAC LEARNING

- Probably approximately correct (PAC) learning in supervised setting



Source: Introduction to ML, CMU, Matt Gormley

PAC LEARNING

- Two types of errors
  - True error (expected risk)

$$R(h) = P_{\mathbf{x} \sim p^*(\mathbf{x})}(c^*(\mathbf{x}) \neq h(\mathbf{x}))$$

    - This error is unknown, unless we know the true data distribution

  - Train error (empirical risk)

$$\hat{R}(h) = P_{\mathbf{x} \sim \mathcal{S}}(c^*(\mathbf{x}) \neq h(\mathbf{x}))$$

    - We can quantify this error on the empirical data distribution
    - S denotes a set of data points x sampled from the empirical distribution

Source: Introduction to ML, CMU, Matt Gormley

PAC LEARNING

- PAC model
  - Generate samples from unknown distribution p^*, $x^i \sim p^*(x), \forall i$
  - Oracle labels each instance with *unknown* function c^*, $y^i = c^*\left(x^i\right), \forall i$
  - Learning algorithm chooses hypothesis $h \in H$ with lowest training error $\hat{R}(h)$

$$\hat{h} = \operatorname{argmin} \hat{R}(h)$$

  - Goal is to choose h with low generalization error $R(h)$

Source: Introduction to ML, CMU, Matt Gormley

PAC LEARNING

- Hypotheses of our interest
  - The *true function* c^* is the one we want to learn and that labeled the training data

$$y^i = c^*\left(x^i\right), \forall i$$

  - The *expected risk minimizer* has lowest true error:

$$h^* = \operatorname{argmin} R(h)$$

  - The *empirical risk minimizer* has lowest training error:

$$\hat{h} = \operatorname{argmin} \hat{R}(h)$$

Source: Introduction to ML, CMU, Matt Gormley

PAC LEARNING

- PAC learnable
  - PAC criterion: our learner produces a high accuracy learner with high probability

$$P\left(\left|R(h) - \widehat{R}(h)\right| \leq \varepsilon\right) \geq 1 - \delta$$

  - Suppose we have a learner that produces a hypothesis $h \in H$ given a sample of N training examples. The algorithm is called *consistent* if for every ε and δ, there exists a positive number of training examples N such that for any distribution p^*, we have:

$$P\left(\left|R(h) - \widehat{R}(h)\right| \leq \varepsilon\right) \leq \delta$$

  - The sample complexity is the min value of N for which this statement holds. If N is finite for some learning algorithm, then H is said to be learnable. If N is polynomial function of $1/\varepsilon$ and $1/\delta$ for some learning algorithm, then H is *PAC learnable*

Source: Introduction to ML, CMU, Matt Gormley

SUPERVISED LEARNING

SUPERVISED LEARNING

- Supervised learning
  - ML algorithms that are “supervised”
  - ML algorithms that learn from labeled data (x, y)
  - Unsupervised learning, e.g., generative models

- An example of supervised learning
  - Medical diagnosis
    - Doctor decides whether a patient is sick or not
    - Doctor looks at attributes (or features) to make a medical diagnosis

SUPERVISED LEARNING

- An example of supervised learning
  - Medical diagnosis
    - Doctor decides whether a patient is sick or not
    - Doctor looks at attributes (or features) to make a medical diagnosis
  - Setting
    - Input $x \in X$
    - Output $y \in Y$
    - Target fn $c^*: X \to Y$
    - Hypothesis $h: X \to Y$
  - Goal
    - A learner produces the hypothesis that best approximates the target fn
    - Classification: discrete y
    - Regression: real-valued y

	y	x_1	x_2	x_3	x_4	
i	allergic?	hives?	sneezing?	red eye?	has cat?	
1	$y^{(1)}$ -	$x_1^{(1)}$ Y	$x_2^{(1)}$ N	$x_3^{(1)}$ N	$x_4^{(1)}$ N	$x^{(1)}$
2	$y^{(2)}$ -	$x_1^{(2)}$ N	$x_2^{(2)}$ Y	$x_3^{(2)}$ N	$x_4^{(2)}$ N	$x^{(2)}$
3	$y^{(3)}$ +	$x_1^{(3)}$ Y	$x_2^{(3)}$ Y	$x_3^{(3)}$ N	$x_4^{(3)}$ N	$x^{(3)}$
4	$y^{(4)}$ -	$x_1^{(4)}$ Y	$x_2^{(4)}$ N	$x_3^{(4)}$ Y	$x_4^{(4)}$ Y	$x^{(4)}$
5	$y^{(5)}$ +	$x_1^{(5)}$ N	$x_2^{(5)}$ Y	$x_3^{(5)}$ Y	$x_4^{(5)}$ N	$x^{(5)}$

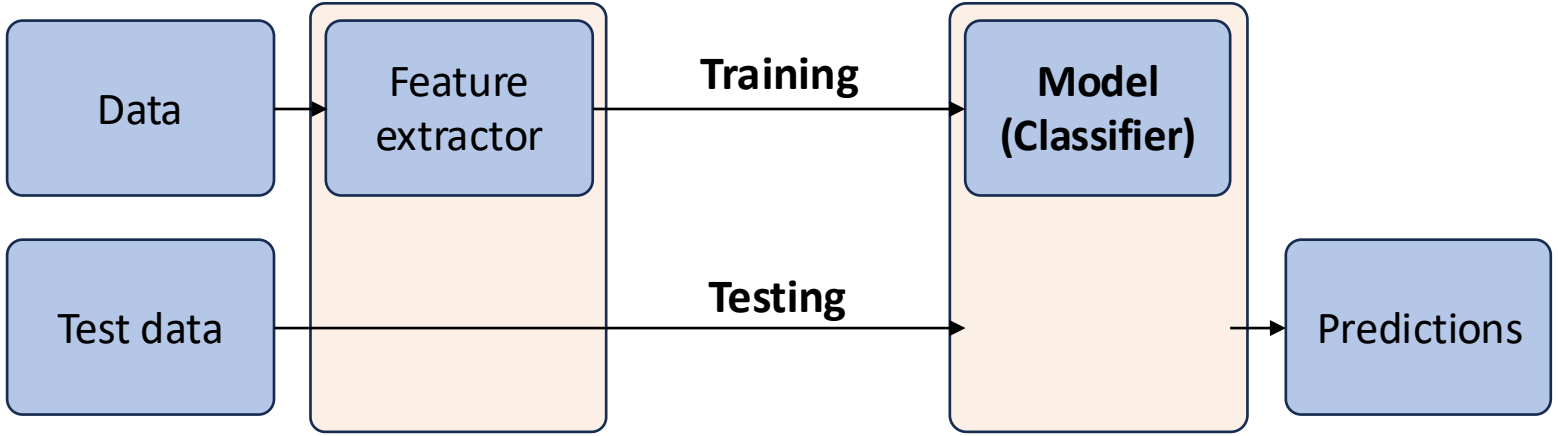
N = 5 training examples

M = 4 attributes

Source: Introduction to ML, CMU, Matt Gormley

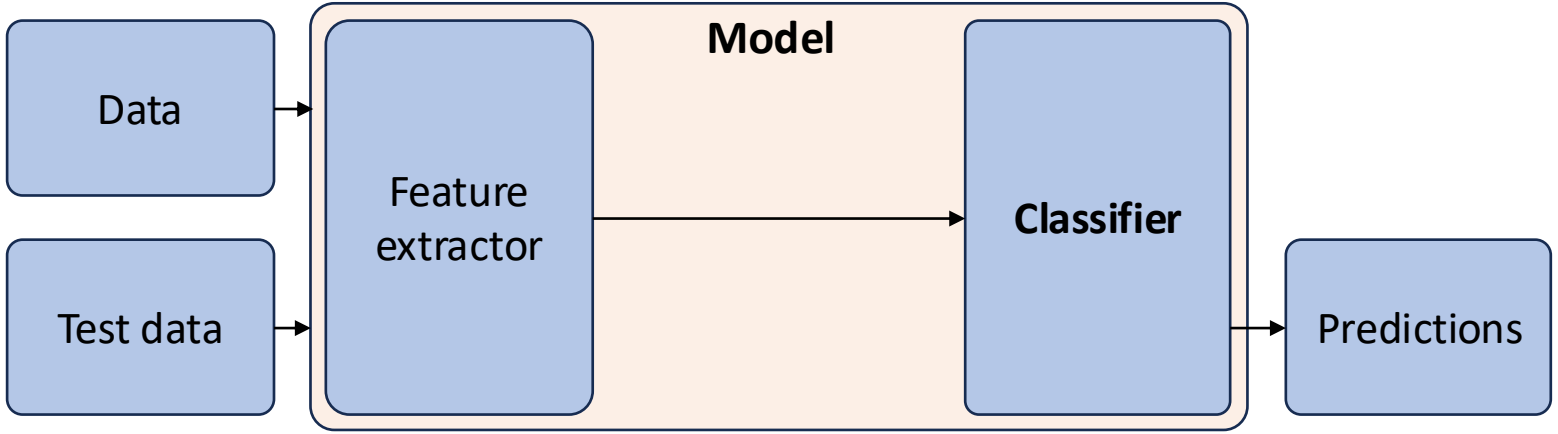
SUPERVISED LEARNING

- Traditional ML pipeline



SUPERVISED LEARNING

- Conventional deep learning pipeline



SUPERVISED LEARNING

- ML
  - Data $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^N, y^N)\}$
  - Model f is a hypothesis function f: $X \rightarrow Y$ and it has parameters θ

- Training
  - A procedure to find the best h: $X \rightarrow Y$
  - A loss (error) function L: $Y \times Y \rightarrow R$ that quantifies how bad h is
    - Common loss function that we will face is “cross-entropy”

SUPERVISED LEARNING

- Empirical risk minimization
  - Data $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^N, y^N)\}$
  - Model f is a hypothesis function f: $X \rightarrow Y$ and it has parameters θ

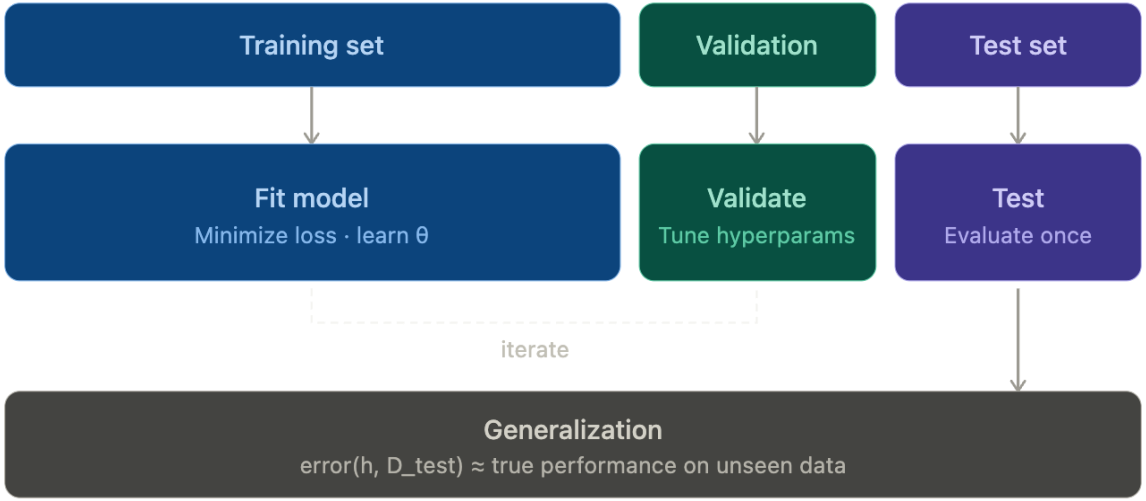
  - Training is a procedure to find the best h: $X \rightarrow Y$
  - The loss (error) function L: $Y \times Y \rightarrow R$ that quantifies how bad h is (cross-entropy)
  - The *empirical risk:* the average loss over the data points

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}^i, y^i) = \frac{1}{n}\sum_{i=1}^{n} \ell(g_\theta(x^i), y^i)$$

  - The *empirical risk minimization (ERM)* is a general method for choosing f $(or\ g)$

SUPERVISED LEARNING

- Validation and testing

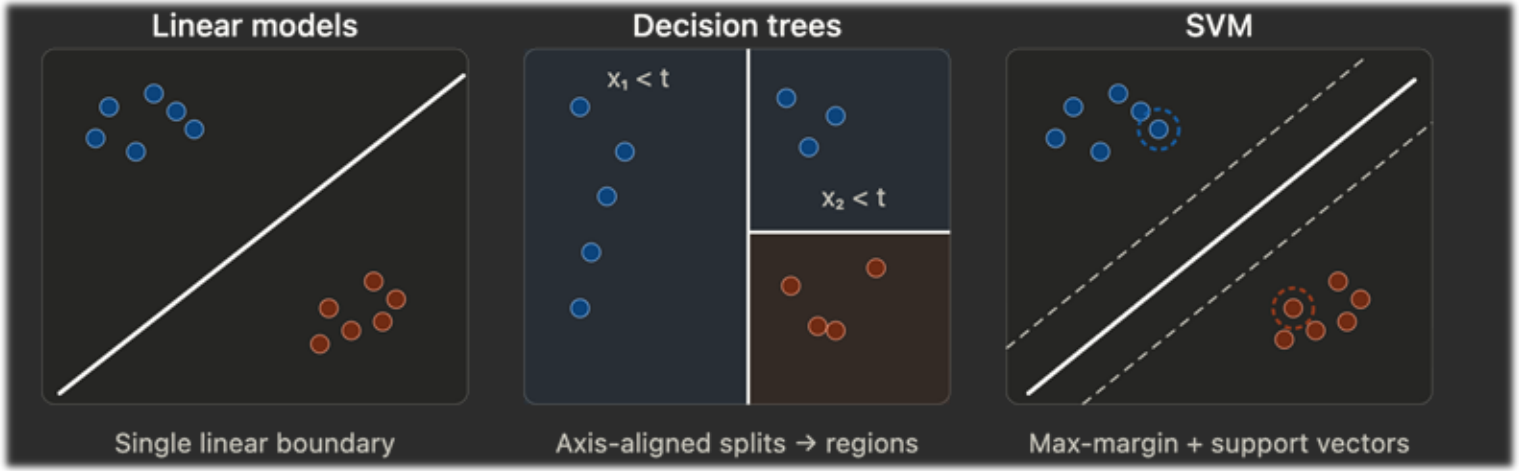


  - Generalization is to learn h with low error on unseen data S

ML MODELS AND OPTIMIZATION

ML MODELS: CONVENTIONAL MODELS

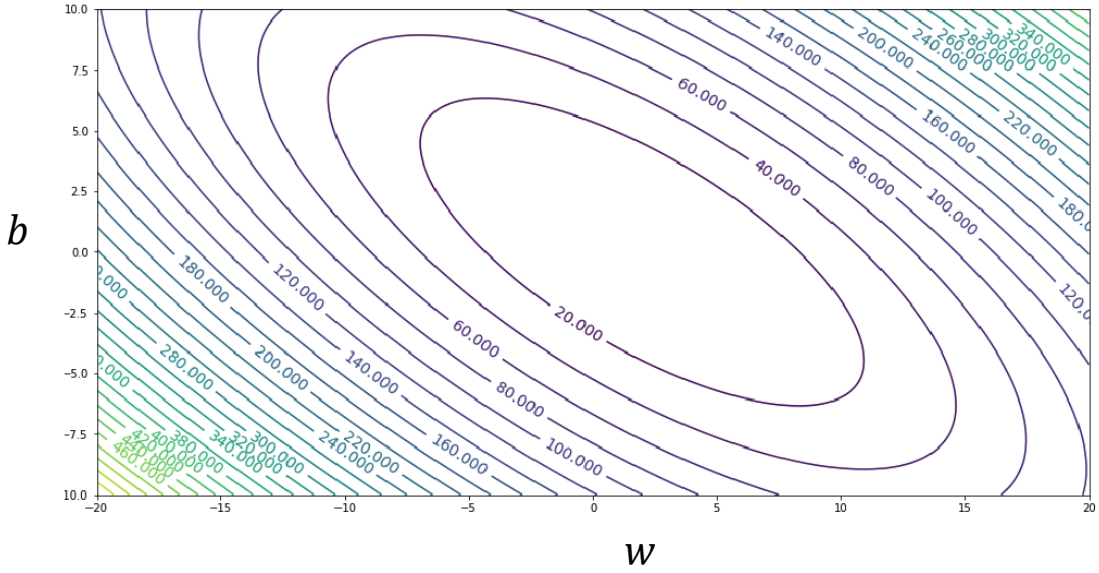
- Linear models, decision trees, and support-vector machines



- Linear models: $h_\theta(x) = \text{sign}(w^T x + b)$, where $l(y, \hat{y}) = \log(1 + e^{-yw^T x})$
- Decision trees: $h_\theta(x) = \sum_1^N c_k \, 1[x \in R_k]$, where $l(y, \hat{y}) = 1[y, \neq \hat{y}]$
- SVM: $h_\theta(x) = \text{sign}(w^T x + b)$, where $l(y, \hat{y}) = \max(0, 1 - yw^T x)$

OPTIMIZATION – GRADIENT DESCENT

- Linear models
  - $h_\theta(x) = \text{sign}(w^T x + b)$, where $l(y, \hat{y}) = \log(1 + e^{-yw^T x})$
  - ERM objective is to learn h that minimize the loss $l(y, \hat{y})$



OPTIMIZATION – GRADIENT DESCENT

- Linear models
  - $h_\theta(x) = \text{sign}(w^T x + b)$, where $l(y, \hat{y}) = \log(1 + e^{-yw^T x})$
  - ERM objective is to learn h that minimize the loss $l(y, \hat{y})$

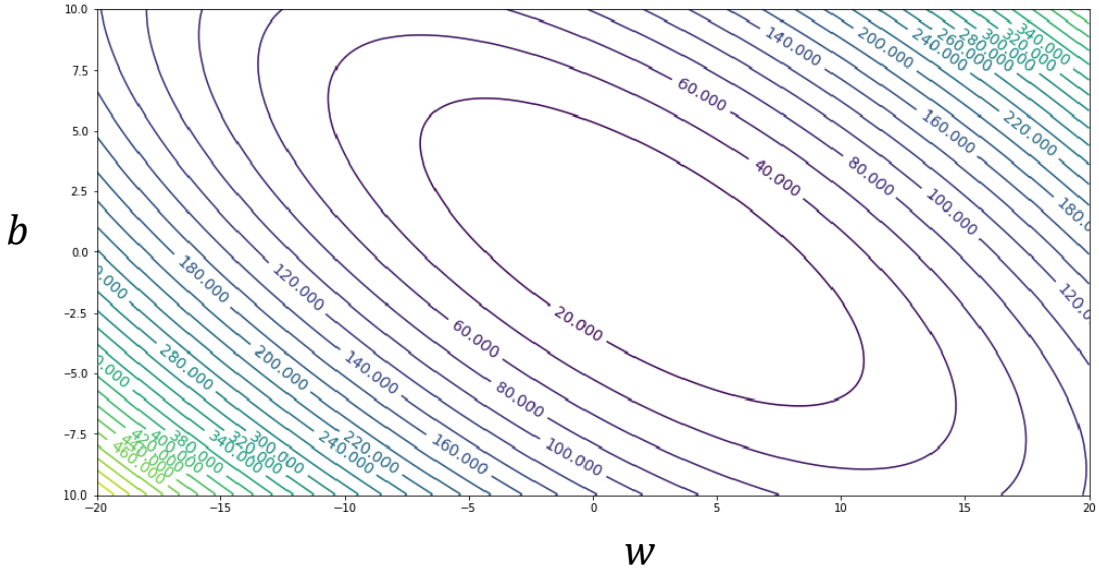
- Gradient descent
  - Compute gradients

$$\nabla l = [\frac{\partial l}{\partial w}, \frac{\partial l}{\partial b}]^{\mathrm{T}}$$

  - Update f parameters θ

$$w \leftarrow w - \gamma \frac{\partial l}{\partial w}$$

$$b \leftarrow b - \gamma \frac{\partial l}{\partial b}$$

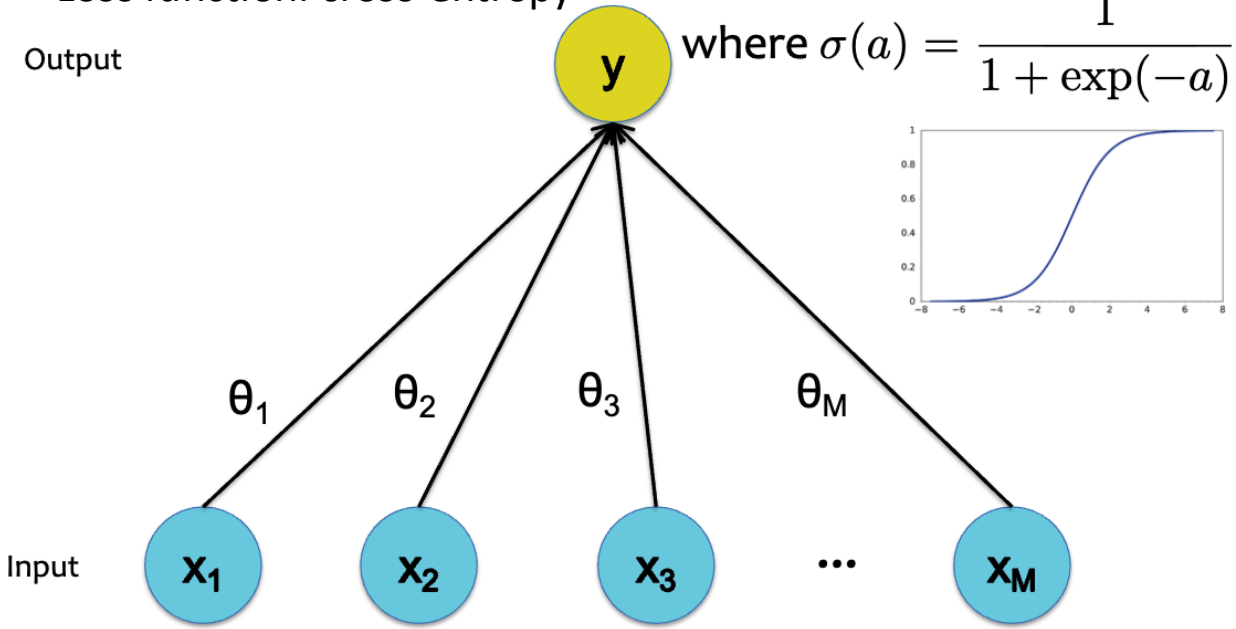


ML MODELS: NEURAL NETWORKS

- Linear model: logistic regression
  - Binary classification {0, 1}
  - Loss function: cross-entropy

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



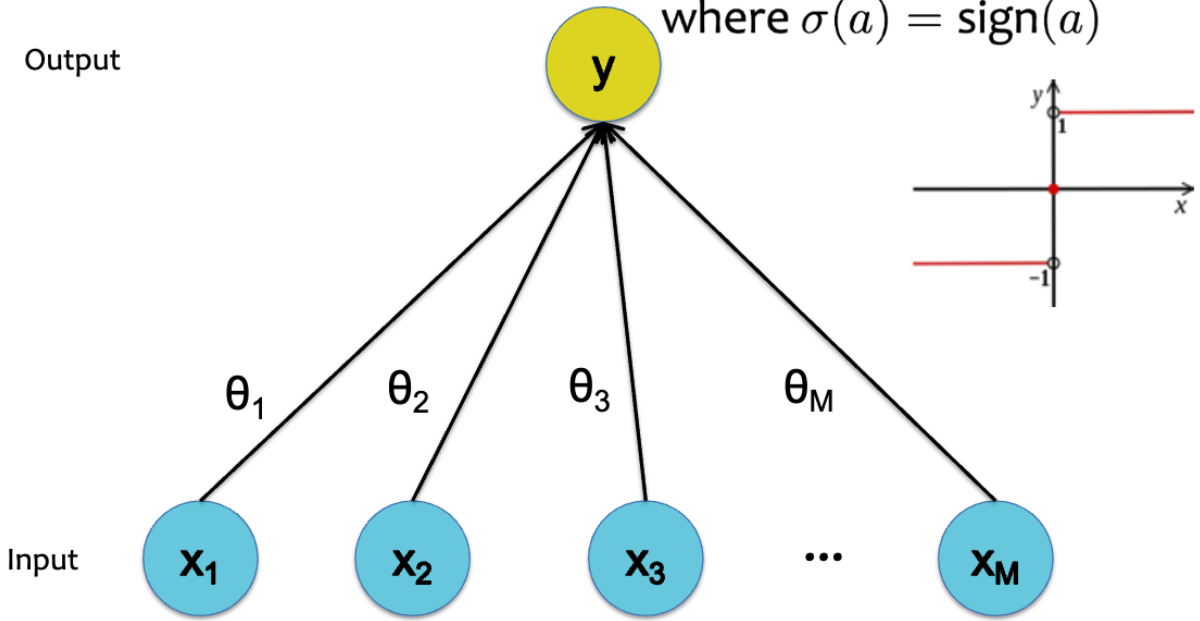
Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

- Linear model: perceptron
  - Binary classification [0, 1]
  - Loss function: 0-1 loss

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

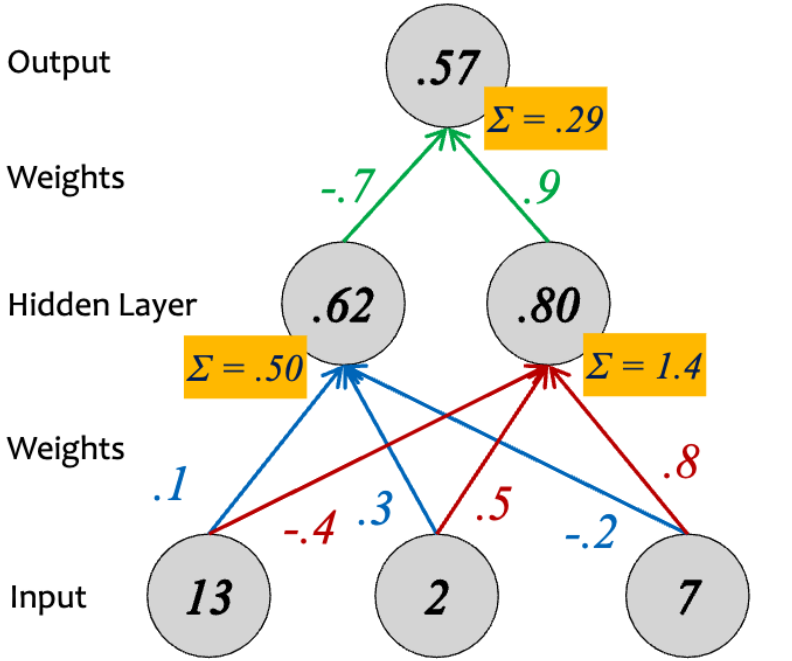
where $\sigma(a) = \text{sign}(a)$



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

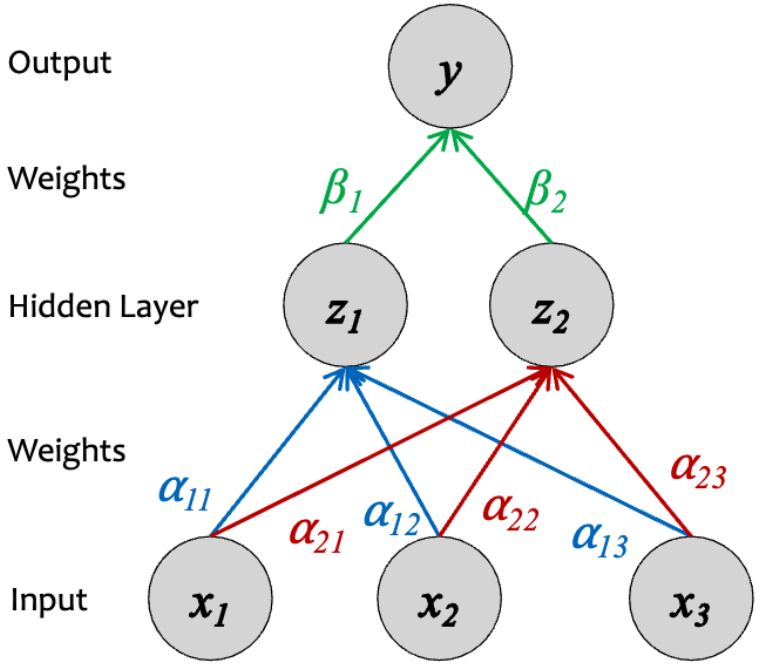
- Multi-layer perceptron
  - Neural network – each perceptron resembles binary logistic regression



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

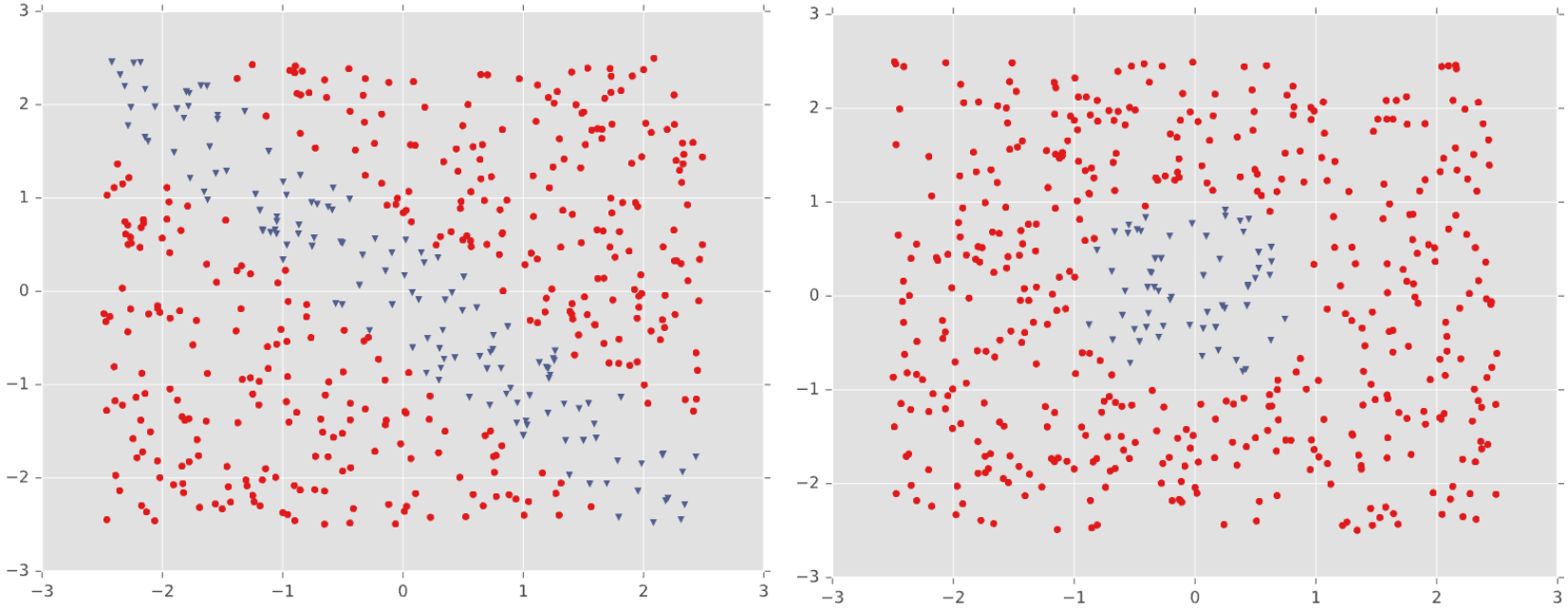
- Multi-layer perceptron
  - Neural network – each perceptron resembles binary logistic regression



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

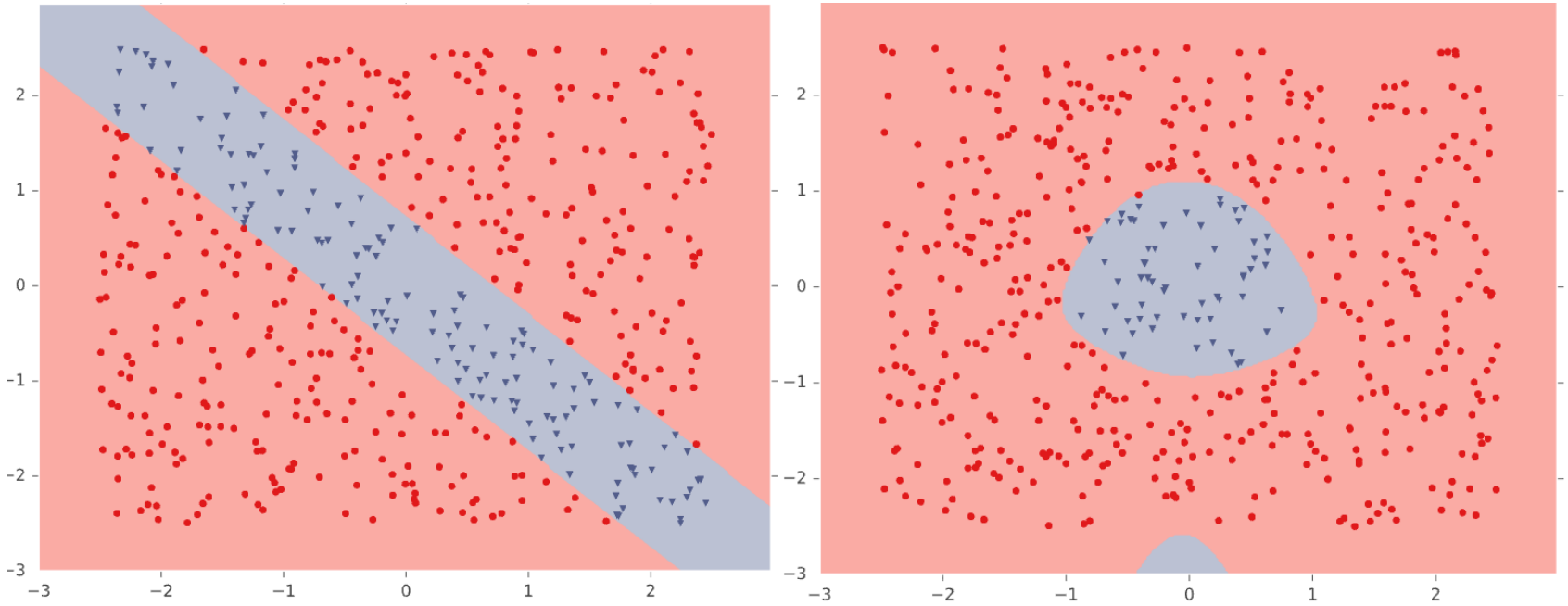
- Neural networks and linear models



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

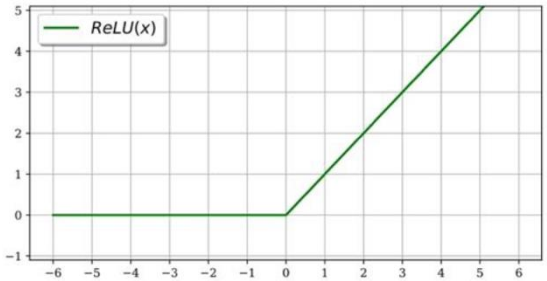
- Neural networks and linear models



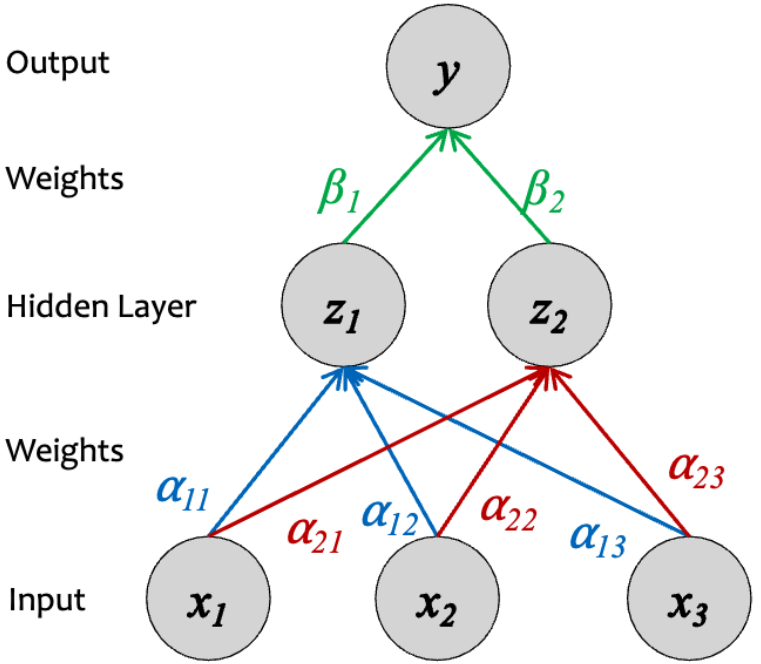
Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

- Neural network basics
  - Parameters: alpha and beta
  - Initialization: how to set alpha and beta first
  - Activation: z = $\sigma(\sum_k \alpha \cdot x)$
  - Activation function: $\sigma(\cdot)$



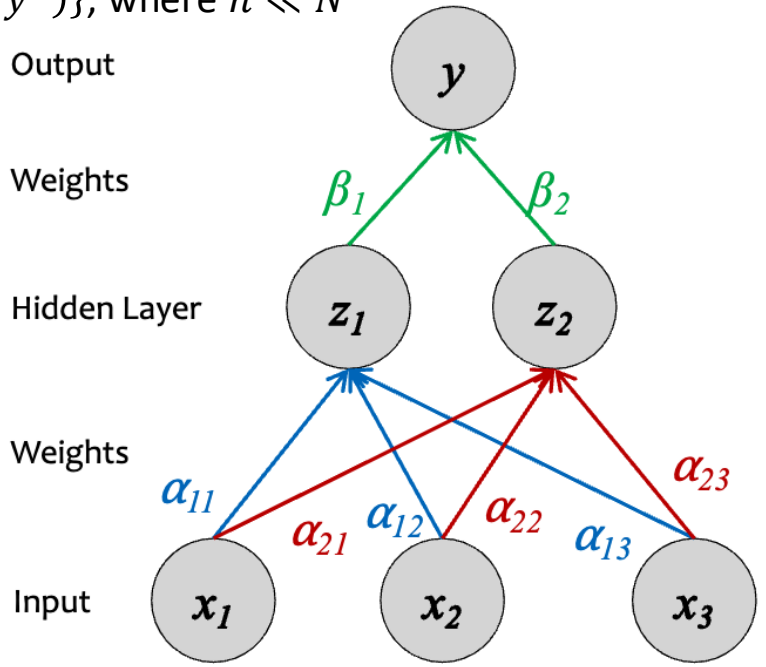
  - Depth: # of hidden layers
  - Width: # of units per hidden layer



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

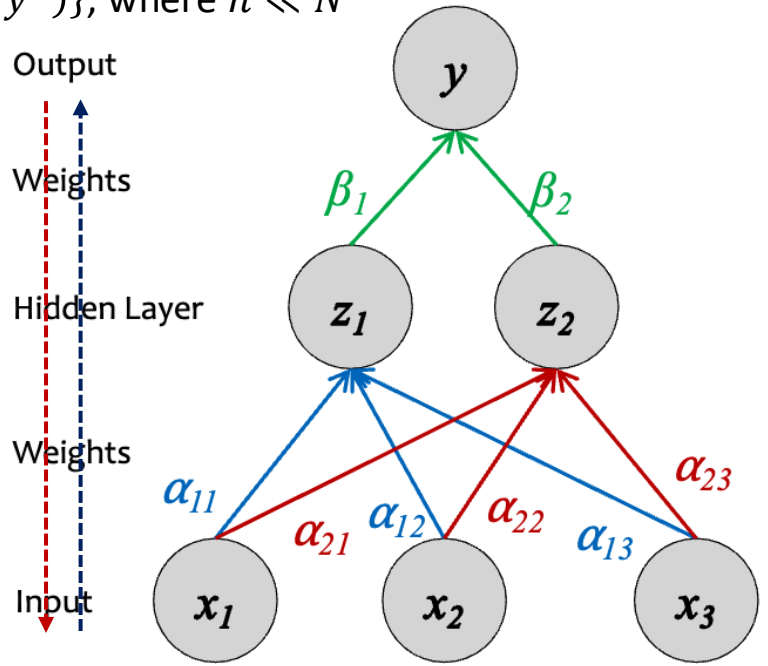
- Training neural networks
  - (Stochastic) gradient descent
    - Draw a mini-batch $B_D = \{(x^1, y^1), \ldots, (x^n, y^n)\}$, where $n \ll N$
    - Compute the loss on B_D
    - Compute gradients of l
    - Update f parameters θ
    - Do this sufficiently and store θ



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

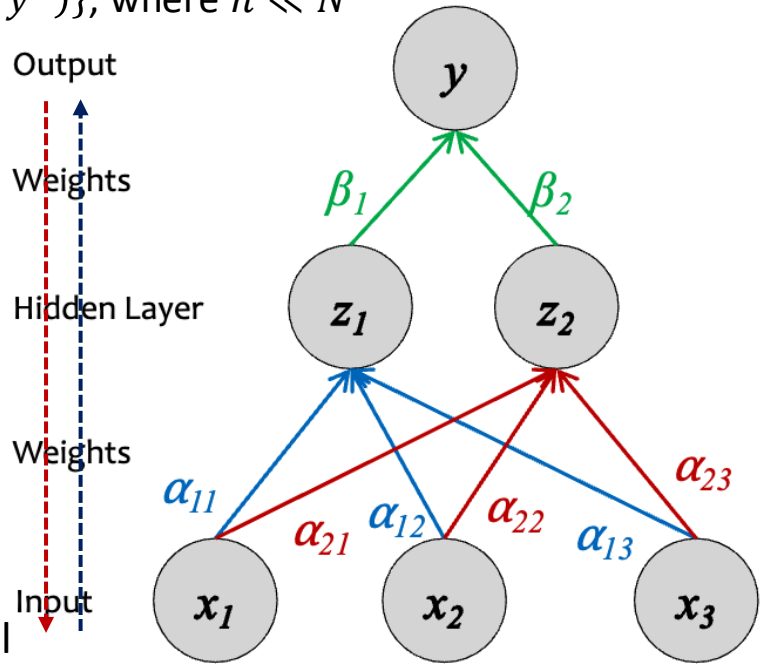
- Training neural networks
  - (Stochastic) gradient descent
    - Draw a mini-batch $B_D = \{(x^1, y^1), \dots, (x^n, y^n)\}$, where $n \ll N$
    - Compute the loss on B_D
    - Compute gradients of l
    - Update f parameters θ
    - Do this sufficiently and store θ
  - Forward and backward pass
    - Forward: compute the loss
    - Backward: compute the gradient



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

- Training neural networks
  - (Stochastic) gradient descent
    - Draw a mini-batch $B_D = \{(x^1, y^1), \dots, (x^n, y^n)\}$, where $n \ll N$
    - Compute the loss on B_D
    - Compute gradients of l
    - Update f parameters θ
    - Do this sufficiently and store θ
  - Forward and backward pass
    - Forward: compute the loss
    - Backward: compute the gradient
  - DIY
    - Backpropagation with a pen and a pencil



Source: Introduction to ML, CMU, Matt Gormley

ML MODELS: NEURAL NETWORKS

- How to update model parameters θ
  - SGD

$$\theta \leftarrow \theta - \gamma \frac{\partial l}{\partial \theta}$$

  - SGD + Momentum

$$\theta \leftarrow \theta - \alpha\, v, \text{ where } v \leftarrow \beta\, v - \frac{\partial l}{\partial \theta}$$

  - Adam

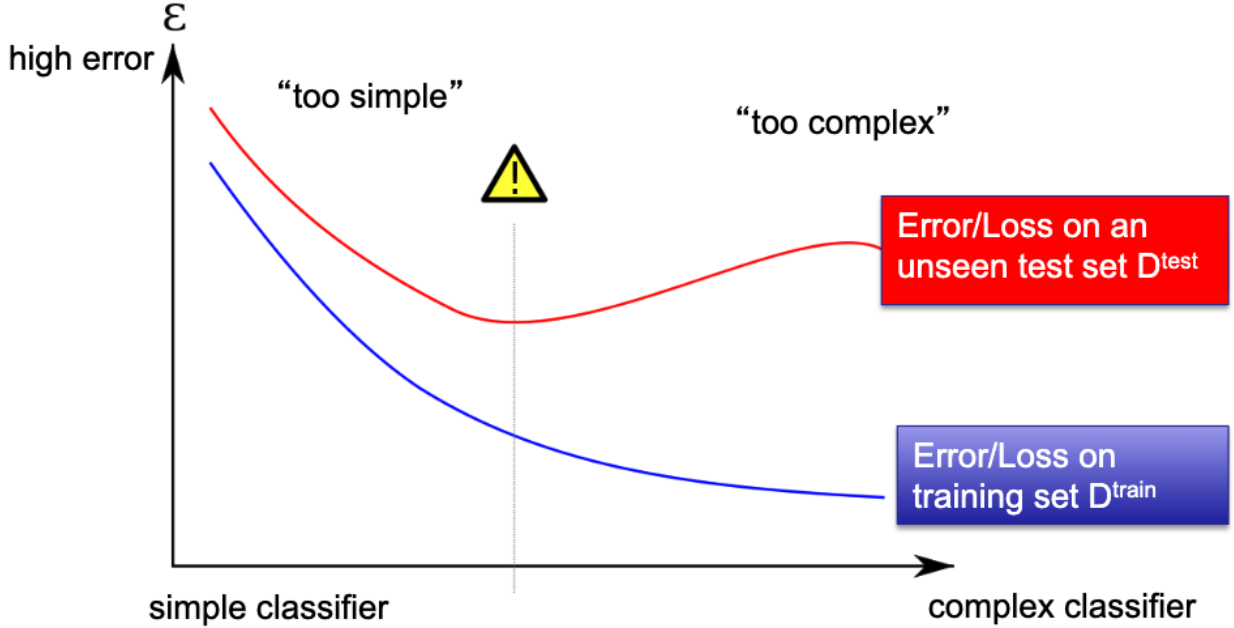
$$\theta \leftarrow \theta - \alpha \frac{\hat{m}}{\sqrt{\hat{v}}+\epsilon}, \text{ where } \hat{m} \leftarrow \frac{m}{1-\beta_1^t} \text{ and } \hat{v} \leftarrow \frac{v}{1-\beta_2^t}$$

$$m \leftarrow \beta_1\, m - (1-\beta_1)\frac{\partial l}{\partial \theta} \text{ and } v \leftarrow \beta_2\, v - (1-\beta_2)\left[\frac{\partial l}{\partial \theta}\right]^2$$

- Optimization hyper-parameters
  - Learning rate γ
  - Batch size $|B_D|$

OPTIMIZATION

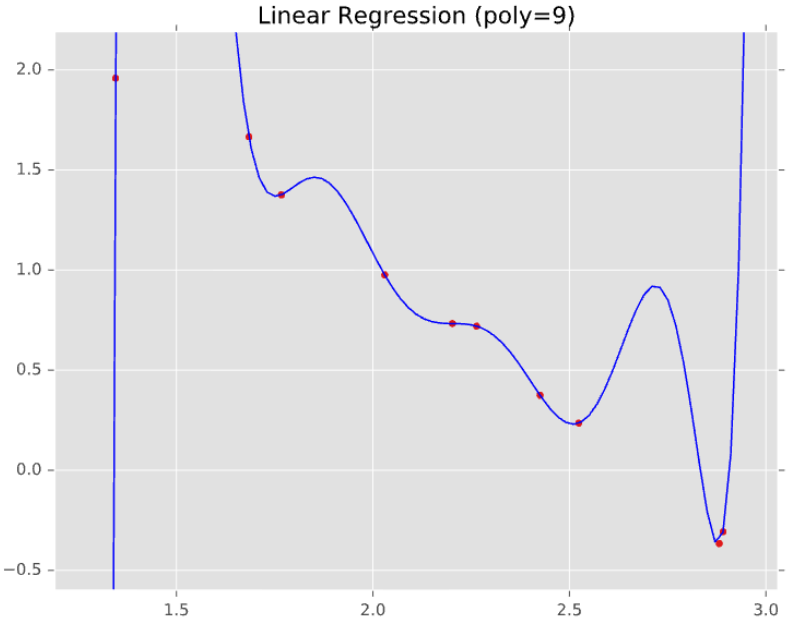
- Overfitting and underfitting
  - **Overfitting:** a model captures the noise in the data, not the underlying structure
  - **Underfitting:** a model is too simple to learn the structure

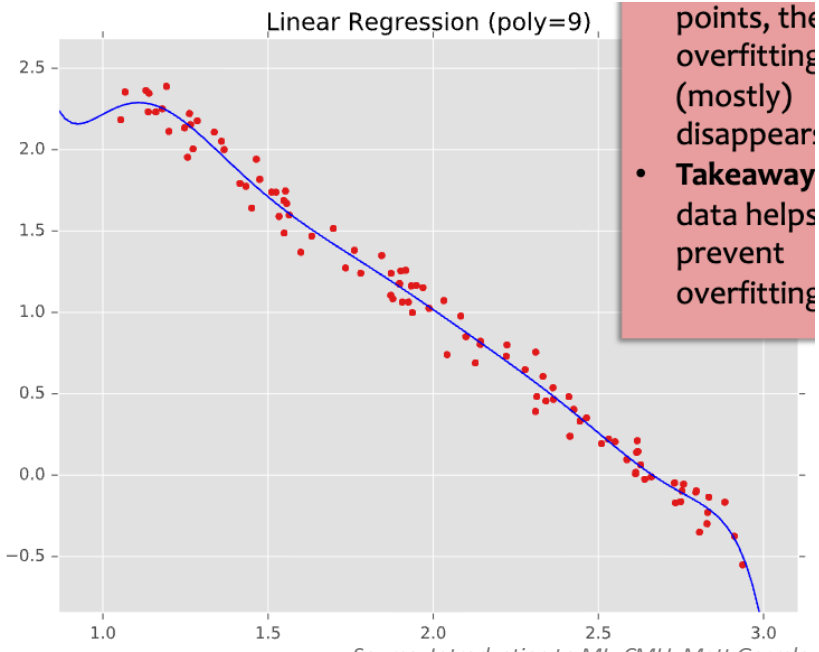


Source: Introduction to ML, CMU, Matt Gormley

OPTIMIZATION

- Overfitting and underfitting
  - **Overfitting:** a model captures the noise in the data, not the underlying structure
  - **Underfitting:** a model is too simple to learn the structure



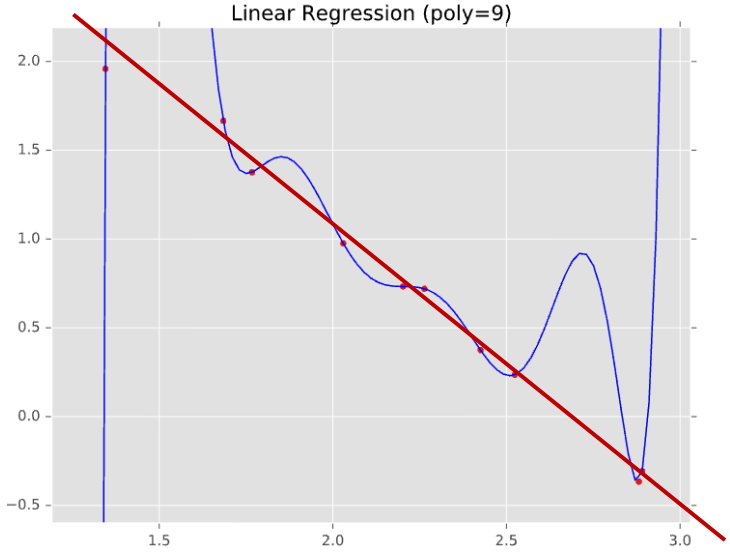


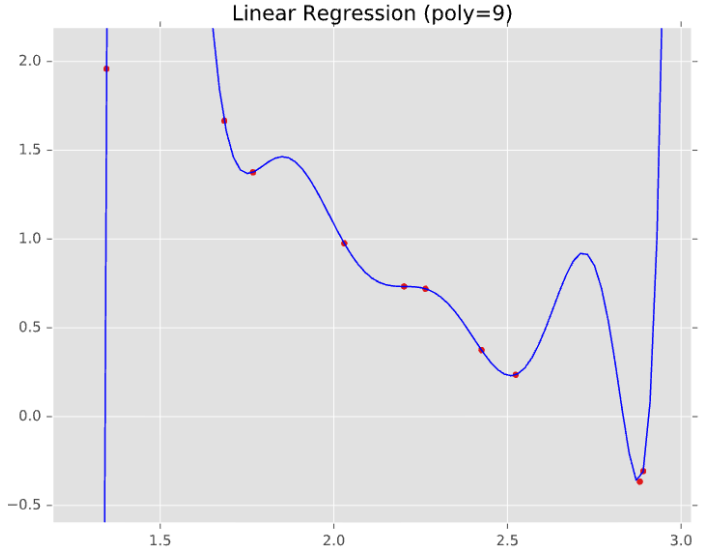
points, the
overfitting
(mostly)
disappear
- **Takeaway**
data helps
prevent
overfitting

Source: Introduction to ML, CMU, Matt Gormley

OPTIMIZATION

- Bias and variance
  - Decompose the error into two: bias and variance
  - Bias: a model cannot fit the data (more complex models are required)
  - Variance: a model can fit the data, but not the underlying structure





Source: Introduction to ML, CMU, Matt Gormley

OPTIMIZATION

- Regularization
  - During optimization (training), make a model prefer simplest hypothesis h
  - Regularization (typically) makes the model less overfit
  - Popular regularization
    - L_1-regularization: add sum of absolute parameter values, making them sparse
    - L_2-regularization: add sum of squared parameter values, making them smaller
  - Normally combined into the loss function

ML MODELS: NEURAL NETWORKS

- Modern neural networks
  - Convolutional neural networks
  - Transformers

NOW YOU ARE READY FOR HOMEWORK ASSIGNMENT 1

(WEBSITE, GITHUB CLASSROOM)

Thank You!

Instructor: Sanghyun Hong

https://true-lab.ai/courses/MLSec/current