Machine Learning: Generalization 3

Hao Tang

March 18, 2024

Recap

- No free lunch theorem tells us we cannot PAC learn on the universe of functions.

- One error decomposition leads us to

$$L_{\mathcal{D}}(h) = \underbrace{L_{\mathcal{D}}(h) - \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)}_{\text{approximation error}} + \underbrace{\min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)}_{\text{estimation error}} . \tag{1}$$

- Choose a hypothesis class $\mathcal{H}$ to balance approximation error and estimation error.

Recap

- Another error decomposition leads us to

$$L_{\mathcal{D}}(h) = L_S(h) + L_{\mathcal{D}}(h) - L_S(h). \tag{2}$$

- Empirirical risk minimization (ERM) attempts to minimize the training error $L_S(h)$.

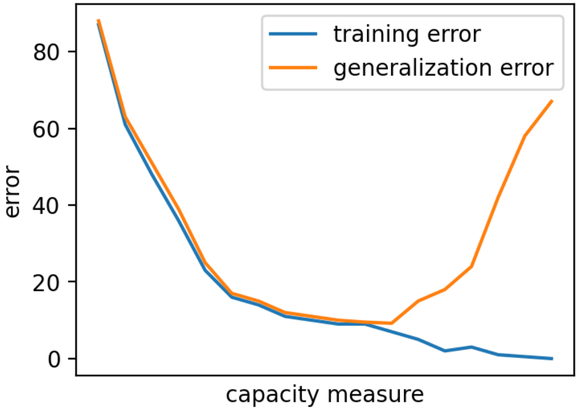
- Choose a hypothesis class such that we can uniform convergence, i.e., $L_{\mathcal{D}}(h) - L_S(h)$ is small.

- With probability $1 - \delta$, for all $h \in \mathcal{H}$

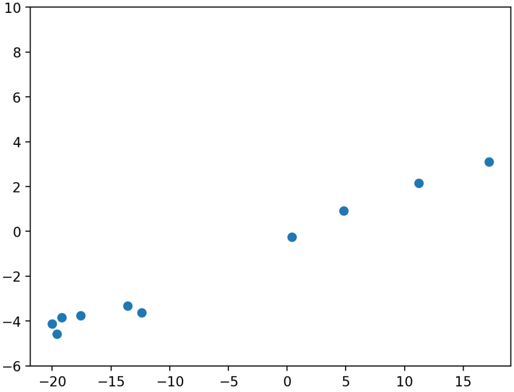
$$L_{\mathcal{D}}(h) \leq L_S(h) + 2\sqrt{\frac{8d\log(en/d) + 2\log(4/\delta)}{n}}, \tag{3}$$

  where d is the VC dimension of $\mathcal{H}$.

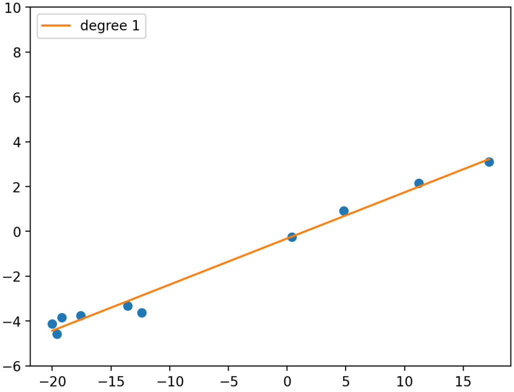
Capacity-generalization tradeoff



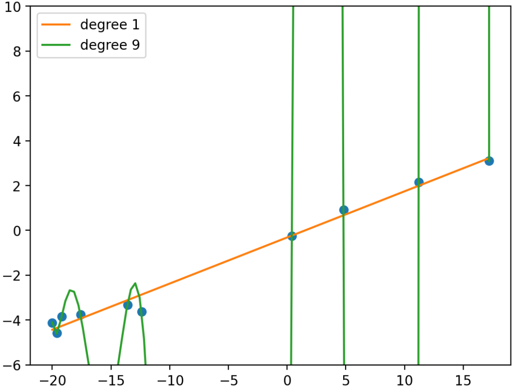
Capacity-generalization tradeoff



Capacity-generalization tradeoff



Capacity-generalization tradeoff



Sample complexity

- How many samples do we need to achieve a certain error?
- How large should n to get to ϵ?

$$\sqrt{\frac{C(\mathcal{H})}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}} \leq \epsilon \tag{4}$$

- In other words,

$$n = O\left(\frac{C(\mathcal{H}) + \log(1/\delta)}{\epsilon^2}\right) \tag{5}$$

Optimization

- We can only do ERM for a limited number of cases, for example, $w = (X^\top X)^{-1} X^\top y$ in linear regression.

- Recall that the convergence of an optimization algorithm tells us how many iterations we need (how large t should be) to get to

$$L_S(h_t) - \min_{h \in \mathcal{H}} L_S(h) < \epsilon. \tag{6}$$

Optimization

- We care about generalization of zero-one loss, not the cross entropy or the log likelihood.

- Cross entropy or the log likelihood are called **surrogate losses**.

- Surrogate losses are easier to optimize than the task loss, and usually have some connection to the task loss.

- For example, log loss is easier to optimize than zero-one loss, and is a smooth approximation of zero-one loss.

Error decomposition

- Optimization error
  - Mismatch between the surrogate loss and the task loss
  - Controlled by the optimization algorithm
- Estimation error
  - Controlled if we do ERM and have uniform convergence
  - Controlled by the capacity of $\mathcal{H}$ and the size of the training set
- Approximation error
  - Controlled by the capacity of $\mathcal{H}$

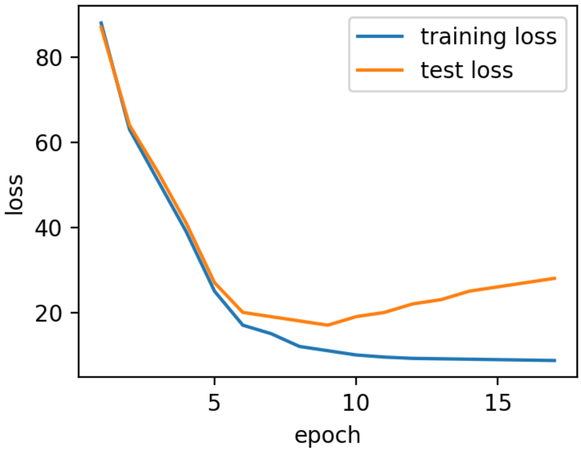
Underfitting

- A model is **underfitting** if there is another model that has a lower training.
- A model h is underfitting if there is f such that $L_S(f) < L_S(h)$.
- The better f is unknown unless we find it.
- All models are underfitting with respect to ERM.
- When people say a model is underfitting, they simply mean there is room to improve the training error.

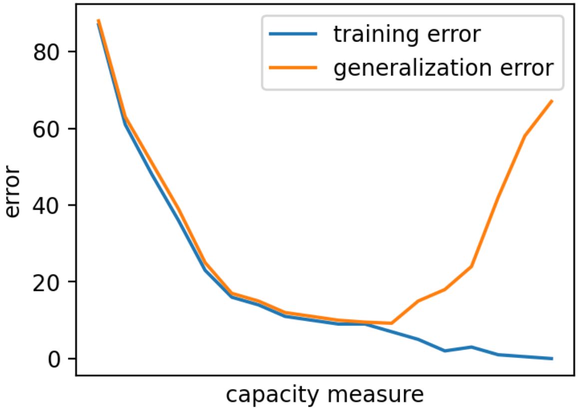
Overfitting

- A model is **overfitting** if there is another model that has a higher training error but a lower test eror.
- A model h is overfitting if there is f such that $L_S(f) > L_S(h)$ and $L_{S'}(f) < L_{S'}(h)$.
- The better f is unknown unless we find it.
- Models can overfit even when the gap $|L_S(h) - L_{S'}(h)|$ between training and test is not large.
- When people say a model is overfitting, they simply mean there is a large gap between the training and test error.

Overfitting



Capacity-generalization tradeoff



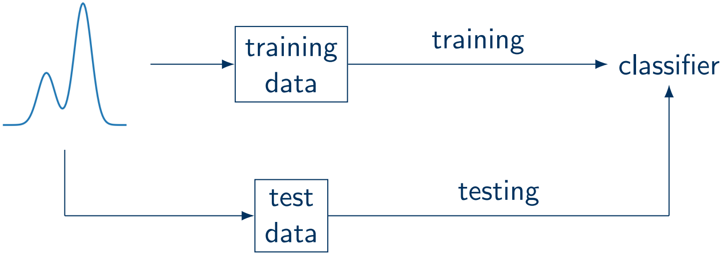
In practice

- We minimize a *surrogate* loss on the training set S, i.e., doing ERM.
- We can only do ERM approximately most of the time, because of optimization difficulty.
- Suppose training gives us $\hat{h}$.
- We use a test set S' and measure *task* loss $L_{S'}(\hat{h})$ to approximate generalization error.
- We hope $L_{\mathcal{D}}(\hat{h})$ is small when $L_{S'}(\hat{h})$ is small.

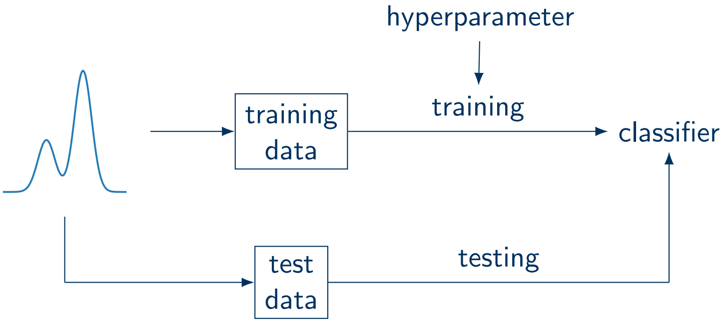
Test set

- Test error on a test set is used to approximate generalization error.
- Test set is supposed to be considered as an indepdent data drawn from the unknown distribution.
- Sometimes we have hyperparameters (not learned from data) we need to tune, for example, the step size in stochastic gradient descent.
- What's the problem of using the test set to tune hyperparameters?

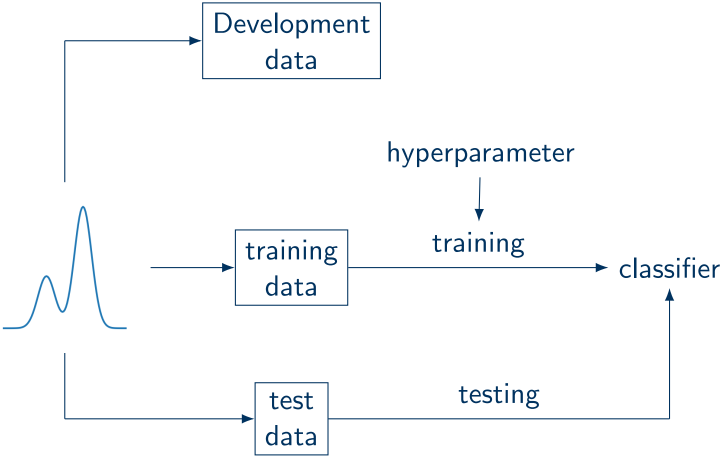
Generalization



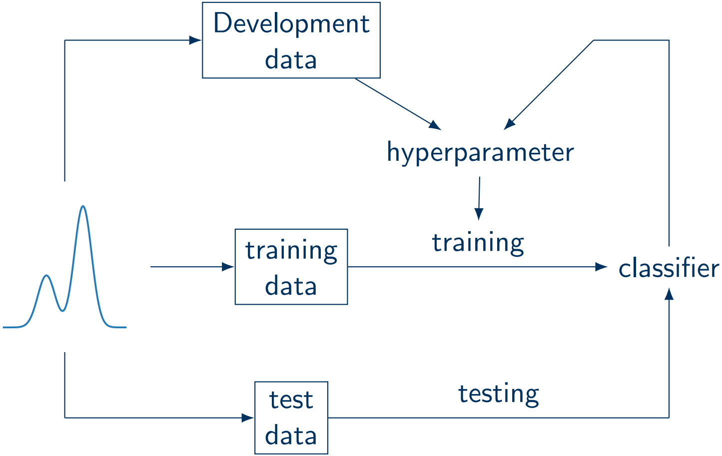
Generalization



Generalization



Generalization



Reusing test sets

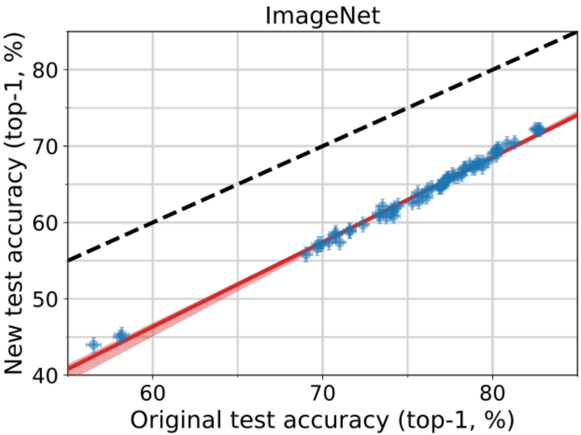


Image credit: (Recht *et al.*, 2019)

Large hypothesis classes

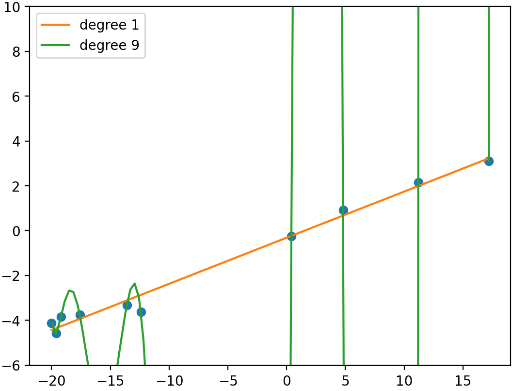
- Compare

$$\mathcal{H}_1 = \text{the set of two-layer neural networks with 512 hidden units} \tag{7}$$

$$\mathcal{H}_2 = \text{the set of all two-layer neural networks} \tag{8}$$

- $\mathcal{H}_1$ has a finite VC dimension, while the VC dimension of $\mathcal{H}_2$ is infinite!
- It is much easier (and tempting) to reduce the training error by increasing the hypothesis class.

Overfitting



Overfitting

- Compare

  $w_2 = [0.206, -0.317]$

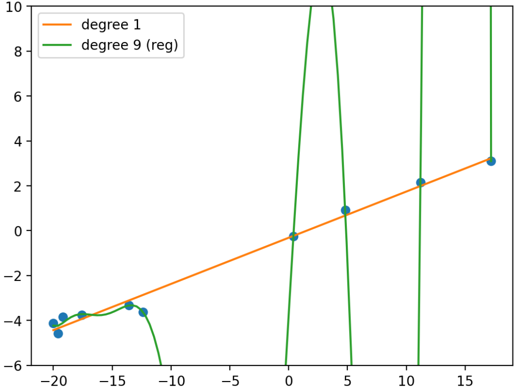
  $w_9 = [-30.69, 93.27, -2.65, -3.29, -0.124, 0.0248, 0.0017, 0.0000245, -0.00000423, -0.0000000857]$

- The learned weights are either too large or too small for degree 9.

- What if instead we optimize

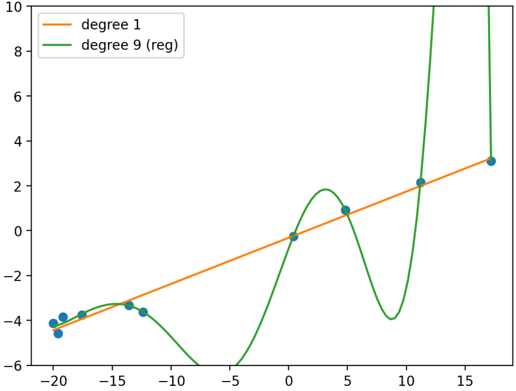
$$\min_{w \in \mathcal{H}} L_S(w) + \frac{\lambda}{2} \|w\|_2^2 \tag{9}$$

Regularization



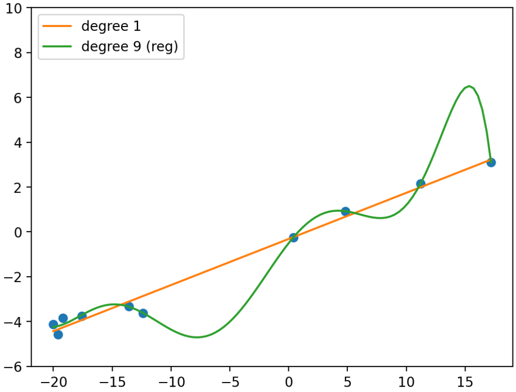
$$\lambda = 0.002$$

Regularization



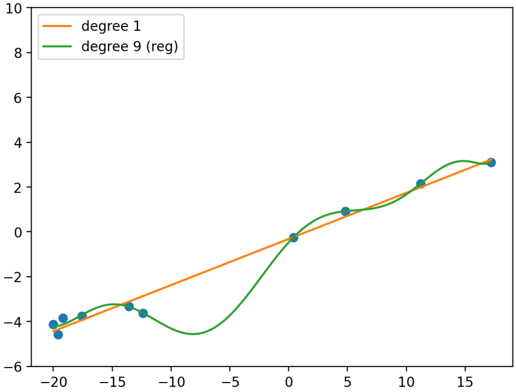
$$\lambda = 0.02$$

Regularization



$$\lambda = 0.1$$

Regularization



$$\lambda = 0.2$$

L_2 Regularization

- The term $\frac{\lambda}{2}\|w\|_2^2$ is called an L_2 regularizer.
- It is also known as weight decay.
- The expression

$$L_S(w) + \frac{\lambda}{2}\|w\|_2^2 \tag{10}$$

is the Lagrangian of

$$\min_{w} \quad L_S(w) \tag{11}$$

$$\text{s.t.} \quad \|w\|_2 \leq B \tag{12}$$

L_2 Regularization

- The L_2 regularizer has an effect of controlling the capacity of the hypothesis class.
- Compare

$$\mathcal{H} = \{x \mapsto w^\top x : w \in \mathbb{R}^d\} \tag{13}$$

$$\mathcal{H} = \{x \mapsto w^\top x : \|w\|_2 \leq B\} \tag{14}$$

Generalization bound for bounded linear classifier

- With probability $1-\delta$, for all $h \in \mathcal{H}$,

$$L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{\frac{r^2 B^2}{n}} + 3\sqrt{\frac{\log(2/\delta)}{2n}}, \tag{15}$$

where $\|x\|_2 \leq r$ for any $x \in S$ and $\mathcal{H} = \{x \mapsto w^\top x : \|w\|_2 \leq B\}$.

Stability

- A learning algorithm is **stable** if the learned program does not change much in performance when we change the data set slightly.
- The slight change in data set is by swapping out a data point.

$$S = \{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)\} \tag{16}$$
$$S^i = \{(x_1, y_1), \ldots, (x', y'), \ldots, (x_n, y_n)\} \tag{17}$$

- A learning algorithm is stable is $A(S)$ and $A(S^i)$ is "similar," or

$$\ell(A(S)(x), y) - \ell(A(S^i)(x), y) \tag{18}$$

is small.[1]

[1] Recall that $L_S(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(x_i), y_i)$.

Stability

- Stable learning algorithms don't overfit.

$$\mathbb{E}_{S\sim\mathcal{D}^n}[L_{\mathcal{D}}(A(S)) - L_S(A(S))] = \mathbb{E}_{\substack{i\sim U(n)\\ S\sim\mathcal{D}^n\\ (x,y)\sim\mathcal{D}}}[\ell(A(S^i)(x_i), y_i) - \ell(A(S)(x_i), y_i)] \tag{19}$$

- Proof

$$\mathbb{E}_S[L_{\mathcal{D}}(A(S))] = \mathbb{E}_S[\mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(A(S)(x), y)]] = \mathbb{E}_S[\mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(A(S^i)(x_i), y_i)]] \tag{20}$$

$$\mathbb{E}_S[L_S(A(S))] = \mathbb{E}_S[\mathbb{E}_{i\sim U(n)}[\ell(A(S)(x_i), y_i)]] \tag{21}$$

Stability

- If the loss function ℓ is convex, then $L(w) + \lambda\|w\|_2^2$ is λ-strongly convex.
- If ℓ is ρ-Lipschitz[2] and $A_{\text{ERM}}(S) = \text{argmin}_{h\in\mathcal{H}}[L_S(w) + \lambda\|w\|_2^2]$, then

$$\|A(S^i) - A(S)\|_2 \leq \frac{2\rho}{\lambda n}. \tag{22}$$

- In the end, we have

$$\mathbb{E}_{S\sim\mathcal{D}^n}\left[L_{\mathcal{D}}(A(S)) - L_S(A(S))\right] \leq \frac{2\rho^2}{\lambda n}. \tag{23}$$

[2] A function f is ρ-Lipschitz if $|f(x) - f(y)| \leq \rho\|x - y\|_2$ for any x and y.