Lecture 3: ML review (2)

CS 182/282A (“Deep Learning”)

2022/01/26

Today's lecture

- Last lecture, we laid out the general machine learning method, and we defined **probabilistic models** (for classification), **likelihood based loss functions**, and **gradient based optimization**
- Now that we have a general recipe for how to learn parameters, we can ask:
  - If my learned parameters minimize the training loss, am I done? Should I deploy my model and move on?
  - How do I determine whether I am "satisfied" with the model?
  - What can I do if I am not satisfied with the model?

True risk and empirical risk

- **Risk** is defined as expected loss: $R(\theta) = \mathbb{E}[\ell(\theta; \mathbf{x}, y)]$
  - This is sometimes called **true risk** to distinguish from empirical risk below
- **Empirical risk** is the average loss on the training set: $\hat{R}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; \mathbf{x}_i, y_i)$
  - Supervised learning is oftentimes **empirical risk minimization (ERM)**
  - Is this the same as true risk minimization?

True risk and empirical risk

- The empirical risk looks just like a *Monte Carlo estimate* of the true risk, so shouldn't we have $\hat{R}(\theta) \approx R(\theta)$? Why might this not be the case?
  - Intuitively, the issue here is that we are already using the training dataset to learn θ — we can't "reuse" the same data to then get an estimate of the risk!
- When the empirical risk is low, but the true risk is high, we are **overfitting**
- When the empirical risk is high, and the true risk is also high, we are **underfitting**

Overfitting and underfitting

- When the empirical risk is low, but the true risk is high, we are overfitting
  - This can happen if the dataset is too small and/or the model is too “powerful”
- When the empirical risk is high, and the true risk is also high, we are underfitting
  - This can happen if the model is too “weak” and/or the optimization doesn’t work well (i.e., the training loss does not decrease satisfactorily)
  - What constitutes “high”? Often, that is up to the practitioner — that is, one must ask: “How well do I expect my model to work for this problem?”
- Generally, the true risk won’t be lower than the empirical risk

Model class and capacity

- We use the term **model class** to describe the set of all possible functions that the chosen model can represent via different parameter settings
  - E.g., the set of all linear functions, the set of all neural network functions with a certain network architecture, …
- Roughly speaking, the **capacity** of a model (class) is a measure of how many different functions it can represent
  - E.g., neural networks have greater capacity than linear models, because neural networks can represent linear functions and more

Questions for the rest of the lecture

- How do we know whether/if we are overfitting or underfitting?
- Given a dataset of a particular size, how do we select:
  - a model class?
  - an algorithm?
  - **hyperparameters**?

Diagnosing overfitting and underfitting

- As mentioned, we cannot rely on the empirical risk $\hat{R}(\theta)$ being an accurate estimate of the true risk $R(\theta)$
  - But we need to estimate $R(\theta)$ in order to diagnose overfitting and underfitting!
- What's the problem? We want to use the dataset for *two purposes*: learning θ and estimating $R(\theta)$
  - This suggests a natural solution: divide the dataset into two parts, one part for learning θ and one part for estimating $R(\theta)$

Training and validation sets

- We use the training set for training, i.e., learning θ
  - The loss on the training set also informs us of whether or not the empirical risk is “high” — if so, we are underfitting
  - Thus, we also use the training set for making sure that the optimization is working, i.e., decreasing training loss satisfactorily
- We reserve the validation set for diagnosing overfitting
  - The loss on the validation set should be an accurate estimate of the true risk, thus we can compare losses on these two sets

Remember: the machine learning method

(or, at least, the deep learning method)

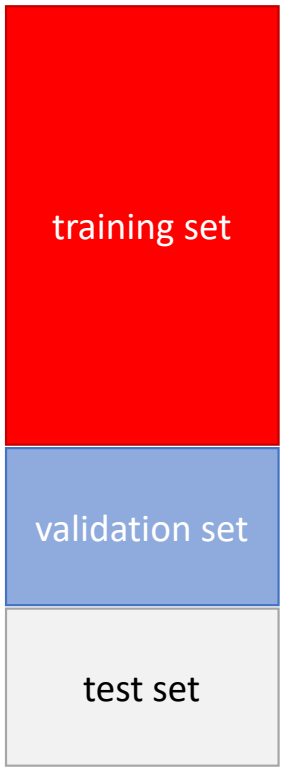
1. Define your **model** — which neural network, what does it output, …
2. Define your **loss function** — which parameters are good vs. bad?
3. Define your **optimizer** — how do we find good parameters?
4. Run it on a big GPU

Introducing: the machine learning *workflow*

1. Define your model
2. Define your loss function
3. Define your optimizer
4. Run it on a big GPU

1. Learn θ on the training set
   - if the training loss is not low enough…
   - you are **underfitting**! increase model capacity, improve optimizer, …
   - and go back to step 1
2. Measure loss on the validation set
   - if the training loss is much smaller than the validation loss…
   - you are **overfitting**! decrease model capacity, collect more data, …
   - and go back to step 1
3. Not overfitting or underfitting? You're done

You're done?



- What does "you're done" mean?
  - In industry, maybe it means: deploy your model
  - In research, competitions, this class, etc., it means: report your model's performance on a **test set**
- The test set is reserved for reporting final performance **only** and must never, ever be used for anything else

Combating overfitting

- Generally, underfitting is not as common of a concern as overfitting
  - Especially with deep learning, we can just keep making the network bigger…
  - … sometimes even without regard for overfitting! More on this later
- What tools and techniques do we have at our disposal if overfitting does occur?
  - Make the network smaller? But we like big models
  - Collect more data? This is a great option, *if possible*
  - Add more inductive biases — let's discuss how to do this via **regularization**

Regularization

- Broadly speaking, a **regularizer** is anything we add to the loss function and/or optimization that does not depend on the data
  - We add it to encode some prior belief about what a "good" model looks like — hence, it is a form of inductive bias
- Bayesian perspective: we can think of many forms of regularization as switching from a maximum likelihood approach to a **maximum a posteriori (MAP)** approach
  - I.e., from $\arg\max_{\theta} \sum_{i=1}^{N} \log p_\theta(y_i \mid \mathbf{x}_i)$ to $\arg\max_{\theta} \sum_{i=1}^{N} \log p(y_i \mid \mathbf{x}_i, \theta) + \log p(\theta)$

Maximum a posteriori estimation

- MLE is equivalent to optimizing the negative log likelihood (NLL) loss function
- MAP estimation is equivalent to adding a regularizer to the NLL loss function, in the form of $-\log p(\theta)$
  - What might be a reasonable choice for this regularizer?
- By far the most commonly used regularizer, when interpreted through the lens of MAP, can be thought of as setting $p(\theta) = \mathcal{N}(\theta; 0, \sigma^2 I)$
  - Then, we have $-\log p(\theta) = \sum_{i=1}^{D} \frac{1}{2}\frac{\theta_i^2}{\sigma^2} + \text{const.} = \lambda ||\theta||_2^2$, where $\lambda = \frac{1}{2\sigma^2}$

ℓ_2-regularization

- With this choice of regularization, our final summed loss becomes
  $\sum_{i=1}^{N} -\log p(y_i \mid \mathbf{x}_i, \theta) + \lambda \|\theta\|_2^2$ — we call this ℓ_2-regularization
  - We usually pick λ directly rather than specifying σ^2 — thus, λ is a hyperparameter
- Why is this a good idea? Smaller parameters typically correspond to *smoother functions* that change less dramatically as the input changes
- You may have already seen this regularizer before in *ridge regression*
- In classification, this is often (somewhat erroneously) referred to as **weight decay**

Perspectives on regularization

- From a Bayesian perspective, the regularizer encodes our prior beliefs about which parameters are (or should be) more likely vs. less likely
- We can also interpret regularization through other perspectives:
  - Numerical perspective: sometimes the regularizer makes an underdetermined problem well determined
  - Optimization perspective: sometimes the regularizer makes the loss function better conditioned and thus easier to “traverse”
    - Paradoxically, more regularization can actually lead to less *underfitting*!

Recap

- So far: how do we know whether/if we are overfitting or underfitting?
  - By measuring and comparing training set loss vs. validation set loss
  - Then, we “tune the knobs” of model capacity, optimization, regularization, …
- Next: given a dataset of a particular size, how do we select settings for these knobs?
  - There are two approaches to answering this question that seem somewhat at odds: the “traditional”/statistical approach, which posits a “bias-variance tradeoff”, and the “deep learning” approach, which suggests that we just keep cranking the knobs up
  - Resolving the apparent inconsistency between these two views is the subject of much ongoing research

A probabilistic model for continuous outputs

for this part, we'll focus on regression, where the outputs $y \in \mathbb{R}$ are real values

we are given $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$\varepsilon \sim \mathcal{N}(0, \sigma^2)$

we assume the data was sampled according to $X \sim p_x$, $Y|X = f(x) + \varepsilon$

how do we define a model that outputs a distribution over $y \,|\, \mathbf{x}$? one option:

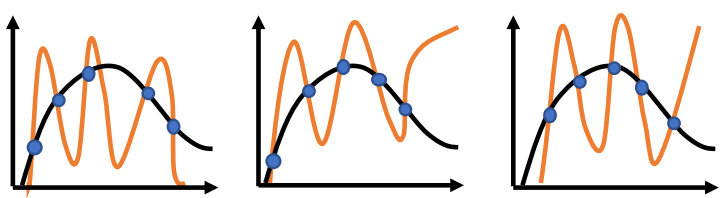
$Y|X \sim \mathcal{N}(f_\theta(x), 1)$ the negative log likelihood loss is

$$-\log p_\theta(y_i|x_i) = \frac{1}{2}(f_\theta(x_i) - y_i)^2 + (\text{const. w.r.t. } \theta)$$

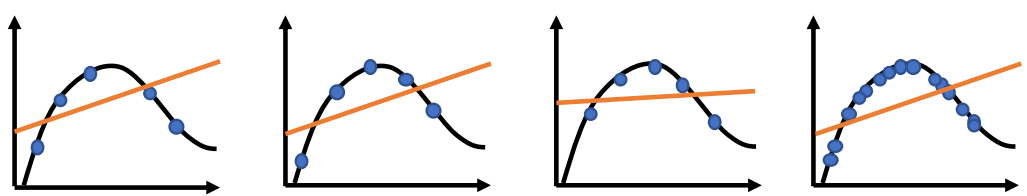
Intuition: bias and variance

- Since we assume the training data $\mathscr{D}$ was randomly sampled, we can ask the question: how does our model change for **different training sets**?
- If the model is overfitting, it will learn a different function for each training set
- If the model is underfitting, it learns similar functions, even if we combine all the training sets together — and all the learned functions are bad

overfitting



underfitting



The bias-variance decomposition ("tradeoff")

(let $\theta(\mathcal{D})$ be the MLE for $\mathcal{D}$, let $f_{\mathcal{D}} = f_{\theta(\mathcal{D})}$)

let's take a look at expected error for a test point $(\mathbf{x}^\star, \mathbf{y}^\star)$, where the expectation is over different training datasets $\mathscr{D}$:

$$\mathbb{E}\left[(f_{\mathcal{D}}(x') - y')^2\right]$$

↑ this is the only RV!

let $\bar{f}(\mathbf{x}^\star)$ be the expected prediction for $\mathbf{x}^\star$, where the expectation is again over the different training datasets (and the parameters that would be learned)

$$\bar{f}(x) = \mathbb{E}\left[f_{\mathcal{D}}(x)\right]$$

The bias-variance decomposition ("tradeoff")

$$\mathbb{E}[(f_{\mathcal{D}}(x') - y')^2]$$

$$= \mathbb{E}[(f_{\mathcal{D}}(x') - f(x') + f(x') - y')^2] \quad \mathbb{E}[y']$$

$$= \mathbb{E}[(f_{\mathcal{D}}(x') - f(x'))^2] + \underbrace{\mathbb{E}[(y' - f(x'))^2]}_{\sigma^2}$$

$$= \mathbb{E}[(f_{\mathcal{D}}(x') - \bar{f}(x') + \bar{f}(x') - f(x'))^2] + \sigma^2$$

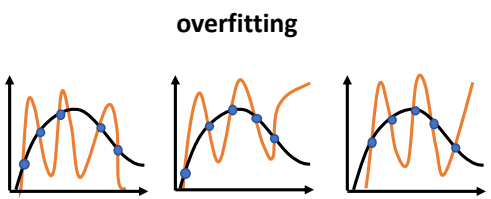
$$= (\bar{f}(x') - f(x'))^2 + \underbrace{\mathbb{E}[(f_{\mathcal{D}}(x') - \bar{f}(x'))^2]}_{\mathrm{Var}(f_{\mathcal{D}}(x'))} + \sigma^2$$

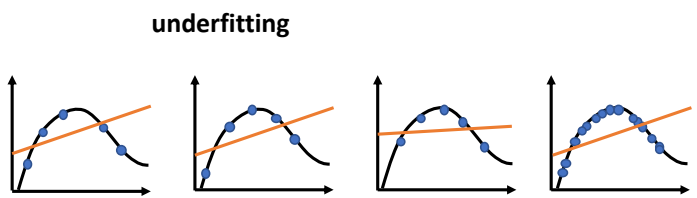
The bias-variance decomposition

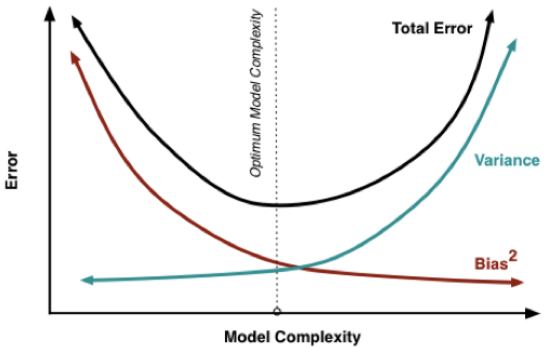
- So: $\mathbb{E}[(f_{\theta(\mathcal{D})}(\mathbf{x}') - y')^2] = (\bar{f}(\mathbf{x}') - f(\mathbf{x}'))^2 + \mathbb{E}[(f_{\theta(\mathcal{D})}(\mathbf{x}') - \bar{f}(\mathbf{x}'))^2] + \sigma^2$
  - The first term is called Bias^2 — how wrong is the model on expectation, regardless of the dataset it is trained on?
  - The second term is Variance — regardless of the true function f, how much does the model change based on the training dataset?
  - The last term is irreducible error — i.e., the noise in the data process itself
- So far, this is just a decomposition — where is the “tradeoff”?

The bias-variance tradeoff?

- Traditional statistics views bias and variance as “competing” sources of error that are regulated by model complexity
- High variance means insufficient data + a complex model class — overfitting
- High bias means an insufficiently complex model class — underfitting







Enter the deep learning perspective…

Allow me to quote Prof. Jitendra Malik 

- “Modern neural network practice doesn’t treat this as a tradeoff — go as high capacity as you can (e.g., networks like GPT-3 push the boundary of current computational hardware)”
- “We don’t fear overfitting!”

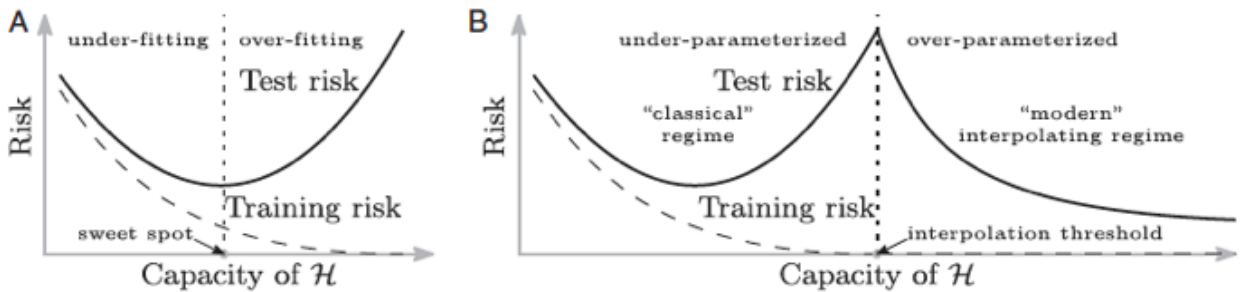


Fig. 1. Curves for training risk (dashed line) and test risk (solid line). (*A*) The classical U-shaped risk curve arising from the bias–variance trade-off. (*B*) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high-capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.